frontiers | Frontiers in Industrial Engineering

TYPE Original Research
PUBLISHED 14 November 2025
DOI 10.3389/fieng.2025.1686126

OPEN ACCESS

EDITED BY
Mitsuo Gen,
Tokyo University of Science, Japan

REVIEWED BY
Wenqiang Zhang,
Henan University of Technology, China
Hiroaki Tohyama,
Maebashi Institute of Technology, Japan

*CORRESPONDENCE
Huizhen Zhang,
huizhenzhang@usst.edu.cn

RECEIVED 14 August 2025
REVISED 22 September 2025
ACCEPTED 29 September 2025
PUBLISHED 14 November 2025

CITATION
Zhang H, Zhou X and Rios Insua D (2025) Efficiently solving open capacitated location-routing problems through a discrete fireworks algorithm.
Front. Ind. Eng. 3:1686126.
doi: 10.3389/fieng.2025.1686126

Efficiently solving open capacitated location-routing problems through a discrete fireworks algorithm

Huizhen Zhang[1]*, Xun Zhou[1] and David Rios Insua[2]

[1]School of Management, University of Shanghai for Science and Technology, Shanghai, China, [2]Institute of Mathematical Sciences ICMAT-CSIC, Madrid, Spain

In the open capacitated location-routing problem (OCLRP), a fleet of distribution vehicles departs from selected depots to satisfy customers' demands, but they do not need to return to their starting depots after serving all customers. Comparing the solutions of the OCLRP and its corresponding capacitated location-routing problem (CLRP) can provide valuable insights for companies considering whether to outsource their delivery activities. To effectively solve the OCLRP, this study proposes a novel discrete fireworks algorithm (DFWA) with two key innovations. (1) An adaptive search mechanism: embedding swap, insertion, and reverse operations into explosion/mutation breaks the limitations of traditional fireworks algorithms in discrete optimization by expanding the search space while enhancing local tuning, thus boosting global optimal solution discovery. (2) A diversified selection strategy: integrating fitness value and Hamming distance improves the "premature convergence from single fitness selection" defect in existing algorithms. It retains high-performance solutions while maintaining population diversity. Evaluated on OCLRP instances adapted from standard CLRP benchmarks, the DFWA shows strong competitiveness, consistently generating high-quality solutions within reasonable computation time. A real-world OCLRP case further verifies its practical applicability in complex industrial scenarios.

KEYWORDS

logistics, open capacitated location-routing problem, capacitated location-routing problem, fireworks algorithm, discrete

1 Introduction

In the current highly competitive economic environment, companies aim for improved supply chain management, including efficient decision-making about depot locations and delivery route design, as these may entail important cost and time savings (Dai, Aqlan, Gao and Zhou, 2019; Zhang et al., 2022b). As a consequence, there is extensive research on both facility location (FLP) (Garcia-Diaz and Smith, 2024) and vehicle routing (VRP) (Niu, Shao, Xiao, Song and Cao, 2022; Zhang et al., 2022a; Garside, Ahmad and Muhtazaruddin, 2024) problems. However, solving these problems as silos may lead to inferior solutions. Location routing problems (LRP) combine both problems and typically lead to more flexible and efficient logistics systems (Mohamed, Klibi, Sadykov, Sen and Vanderbeck, 2022; Yu and Lin, 2015).

A major consequence of the current competitive environment is that companies are increasingly focusing on their core value-added capabilities and activities and tend to

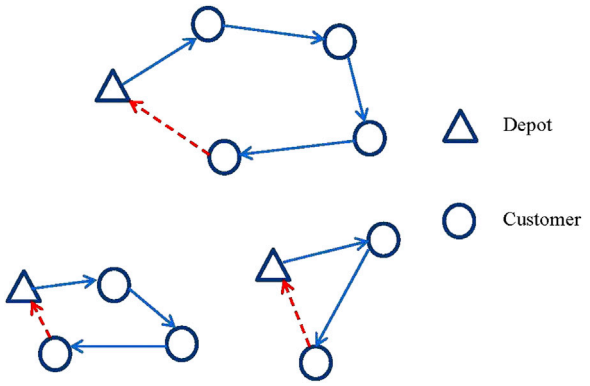

FIGURE 1
CLRP and OCLRP solutions with three depots, three routes, and nine customers.

outsource their logistics to third-party (TPL) providers (Yu and Lin, 2015). In such cases, TPL provider vehicles usually start their service from the company's depots (or distribution centers) and return to the TPL installations instead of the depots or distribution centers after delivering to customers; consequently, classic capacitated location-routing problems (CLRPs) are not applicable. To handle these situations, open capacitated location routing problems (OCLRPs) were proposed by Yu and Lin (2015). OCLRPs do not consider the costs associated with the starting trip from the TPL company to its depot nor the return trip from a vehicle's last customer to the TPL facilities. Thus, the company only considers the costs of locating its depots and those associated with the number of hired vehicles and the trips between depots and the last customer that the vehicle served (Nucamendi-Guillén, Padilla and Moreno-Vega, 2021).

This study provides a novel approach to OCLRPs. We assume that each candidate depot and vehicle have limited capacities and that vehicles return to TPL installations instead of the company's depots after having served their last customer. OCLRPs simultaneously search for depot locations and vehicle routes to satisfy customers' demands. The main difference between CLRPs and OCLRPs is that each route in an OCLRP is a Hamiltonian path (vehicles do not return to the starting depot), whereas each route in a CLRP is a Hamiltonian cycle (vehicles return to the starting depot) (Yu and Lin, 2015). Figure 1 illustrates the differences between CLRP and OCLRP solutions in a problem with three depots (triangles), nine customers (circles), and three routes. Both solutions are constructed on a complete directed graph $G = (V, E)$ where the vertex set $E = \{ < i, j > | i, j \in V, i \neq j \}$. Critically, the cost of each arc $< i, j > \in E$ corresponds to the straight-line distance between the spatial positions of vertex i and vertex j (spatial positions of all points are meaningful for route cost calculation). In Figure 1, for visual clarity (especially in printed form), we use solid arcs to represent the OCLRP solution (Hamiltonian paths starting from depots and ending at customers) and solid arcs plus red dashed arcs to represent the corresponding CLRP solution (Hamiltonian cycles that return to the starting depot after serving all customers on the route).

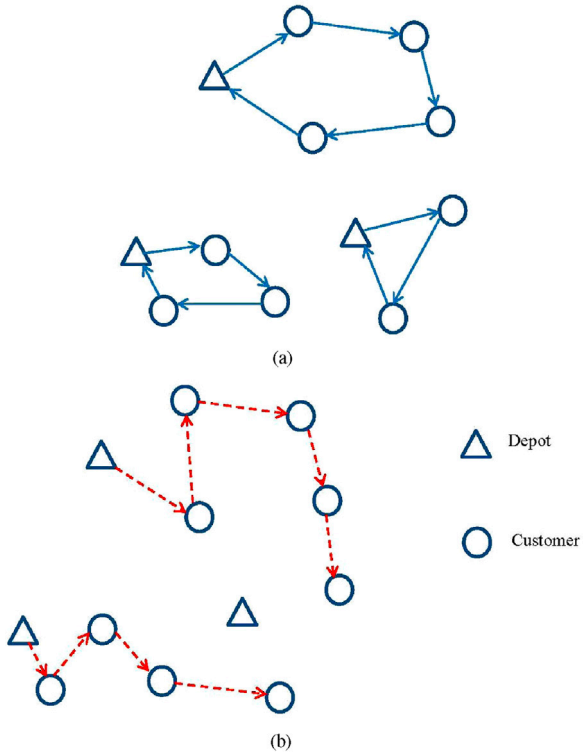

FIGURE 2
Illustration of the differences between the optimal solutions of a CLRP and that of its corresponding OCLRP. **(a)** Optimal solution of CLRP instance. **(b)** Optimal solution of OCLRP instance.

Importantly, the difference between the solution values of a CLRP and its corresponding OCLRP provides a benchmark to support a company's decision on whether to outsource delivery activities (Yu and Lin, 2015). While OCLRP and CLRP solutions only differ in the "depot–last customer" arc cost (OCLRP omits this cost), a good CLRP cycle does not guarantee a good OCLRP path; this is because the optimal OCLRP path depends on both inter-customer distances and customer–depot distance, rather than just the closed-loop cost of CLRP. Figure 2 further validates this discrepancy, where 2(a) shows the optimal solution of a CLRP and 2(b) shows the optimal solution of its corresponding OCLRP (both with three candidate depots and nine customers). In Figure 2, depots are marked as triangles and customers are marked as circles. Both 2(a) and 2(b) are built on the same weighted complete directed graph, so that their vertex set, spatial positions of all vertices, and arc costs (straight-line distances) are identical. Solid arcs in Figure 2a form three CLRP cycles, while the red dashed arcs in Figure 2b form two OCLRP paths. This figure helps illustrate that the optimal solution for an OCLRP cannot simply be derived by removing the edge between the depot and the last customer from the optimal solution of the CLRP.

As the VRP (Ergao and Mingyong, 2009) and FLP (Garcia-Diaz and Smith, 2024) are both NP-hard problems, so the CLRP is also NP-hard (Dai et al., 2019). Given its important practical applications and high computational complexity, the CLRP has attracted the interest of numerous researchers who have proposed a wide variety of algorithms, including genetic algorithms (Bootaki and Zhang,

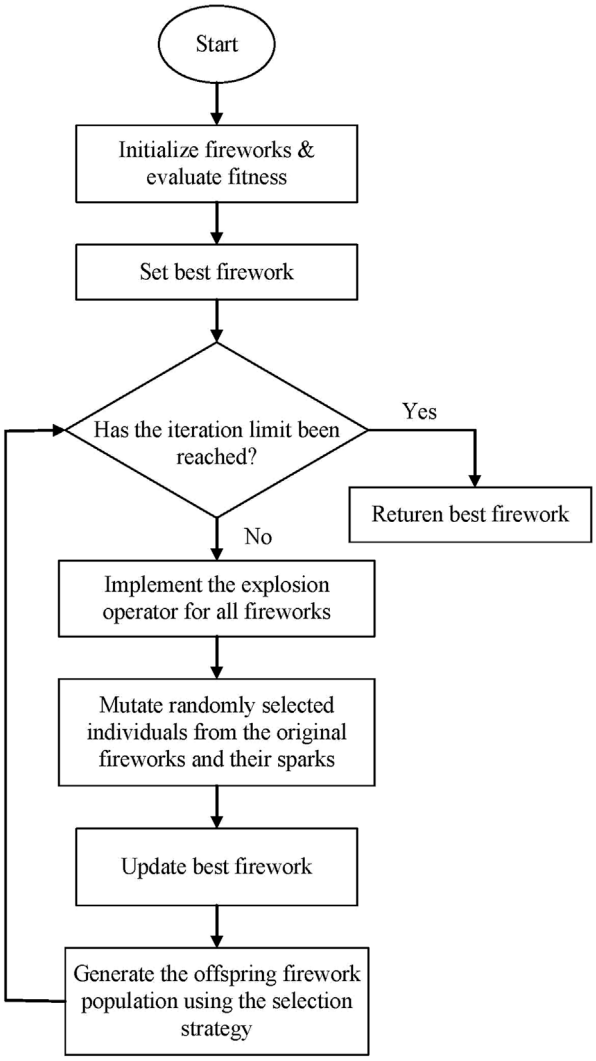

FIGURE 3
Flowchart of discrete fireworks algorithm for OCLRP.

2024; Arifuddin, Utamima, Mahananto, Vinarti and Fernanda, 2024), simulated annealing (SA) (Ferreira and Alves de Queiroz, 2022), particle swarm optimization (Kechmane, Nsiri and Baalal, 2018; Wang et al., 2021), tabu search (Kyriakakis, Sevastopoulos, Marinaki and Marinakis, 2022), ant colony optimization (Ting and Chen, 2013), greedy randomized adaptive search procedures (Prins, Prodhon and Calvo, 2006), and memetic algorithms with population management (Shi, Zhou, Boudouh and Grunder, 2022). Moreover, some exact algorithms have been also proposed, based on dynamic programming, column generation (Farham, Süral and Iyigun, 2018), branch-and-cut (Marques, Sadykov, Deschamps and Dupas, 2020), and branch pricing algorithms. In contrast, the OCLRP has received comparatively less attention, and there is still much scope to improve available models and solutions. Nucamendi-Guillén et al. (2021) proposed a multi-start metaheuristic algorithm to solve the multi-depot open location routing problem with a heterogeneous fixed fleet. Yu and Lin (2015) proposed an effective simulated annealing heuristic for solving OCLRPs based on a special solution representation enlarging the search space and facilitating its exploration. Toro, Franco, Granada-Echeverri, Guimarães, and Rendón (2016) took into account current environmental issues and proposed the green open location-routing problem. Nucamendi-Guillén et al. (2021) were inspired by a real case and proposed a multi-depot OLRP with a fixed heterogeneous fleet.

As a variant of the CLRP, the OCLRP retains the same computational challenge. Moreover, existing mainstreams for similar routing problems face notable limitations when adapted to the OCLRP's unique characteristics (i.e., Hamiltonian path constraints and depot–customer–vehicle coupling). For instance, simulated annealing heuristics such as proposed by Yu and Lin (2015) expand the search space via special solution representation but often stagnate in local optima for large-scale instances (e.g., 100+ customers), as their single-solution iterative update lacks sufficient local exploitation. Genetic algorithms, though widely applied to the CLRP (Bootaki and Zhang, 2024), struggle with the OCLRP's path-specific constraints, as their crossover operators often disrupt valid route structures, leading to a high proportion of infeasible solutions that require additional repair mechanisms. Even neighborhood-based methods such as adaptive large neighborhood search (ALNS) and tabu search (TS), while excellent at neighborhood exploration for VRPs, lack the targeted handling of the OCLRP's triple coupling of depot selection, customer assignment, and vehicle routing, resulting in inefficient search processes and suboptimal cost performance.

In contrast, fireworks algorithms (FWAs) have shown accurate and consistent performance in complex optimization (Y. Tan and Zhu, 2010; Chen et al., 2019) but are rarely applied to NP-hard combinatorial problems like OCLRP, FLP, VRP, or LRP and their variants. This study aims to fill this gap by proposing a novel discrete fireworks algorithm (DFWA) for solving the OCLRP, motivated by three key advantages of the FWA. First, as a heuristic approach, the FWA provides an effective algorithmic framework that delivers acceptable solutions in reasonable computing time to NP-hard optimization problems (Niu et al., 2022). Second, as a meta-heuristic, it allows the flexible integration of local search operators (Zhang et al., 2019; Tan et al., 2025)—a feature that can mitigate SA's local stagnation and GA's infeasibility issues. Third, the performance of FWAs have been found accurate and consistent, and they can achieve competitive results in comparison to other well-known optimization algorithms (Tan and Zhu, 2010; Chen et al., 2019). They have been applied in domains such as multimodal function optimization (Meng and Tan, 2024), radar deployment for UAV swarms defense coverage (Ding et al., 2025), and heterogeneous multi-project multi-task allocation in mobile crowdsensing (Shen et al., 2023), indicating their potential to handle the OCLRP's complex constraints.

The key contributions of this study are the following. Firstly, it proposes a mathematical programming model for the OCLRP. Second, it presents an efficient DFW algorithm for solving the OCLRP. This entails providing an effective initialization and convenient explosion and mutation operators to respectively enhance the exploitation and exploration of the search space. Third, in-depth numerical experiments are conducted to facilitate parameter choice and evaluate the operational gains obtained using the proposed OCLRP model and DFWA. Such experiments and a real case study clearly showcase the effectiveness of our proposal.

FIGURE 4
Example of solution representation.

The rest of this study covers the following issues. Section 2 states the details of the OCLRP model. Section 3 proposes and discusses our DFWA. Section 4 then provides an in-depth computational analysis, including a real case study. Conclusions and future research are presented in Section 5.

2 Mathematical model for the OCLRP

We present here the formulation of the OCLRP for which we shall develop an efficient algorithm. A set of customers with known locations is available. Their known demands should be satisfied from a set of depots with known locations and opening costs. For this, there is a fleet of homogeneous vehicles with known capacity. Customers must be serviced once, and only once, by a single vehicle. A route load cannot exceed the vehicle capacity, with each route starting from a depot and ending at a customer. Similarly, the demands allocated to a depot cannot surpass its capacity. The OCLRP is typically formulated to minimize total distribution costs, which cover the costs of opening the depots, the fixed vehicle costs, and the routing costs.

The underlying structure to formulate the problem is a complete directed graph (V, E). Its vertex set V is $V_p \bigcup V_d \bigcup V_c$ where V_p, V_d, and V_c, respectively, represent the set of locations of the parking lots of the TPL company, of m candidate depots, and of n customers. Its arc set is $E = \{\langle i, j\rangle | i, j \in V \text{and } i \neq j\}$. The rest of the notation is as follows.

Sets and parameters:

V_d: $= \{1, 2, \ldots, m\}$.
V_c: $= \{m+1, m+2, \ldots, m+n\}$.
K: set of homogeneous vehicles with common capacity P.
O_i: i-th depot fixed opening cost.
d_i: i-th customer demand.
Q_i: i-th depot capacity.
c_{ij}: distance from node i to node j.
e: transportation cost per unit distance.
w: vehicle fixed cost.

Decision Variables:

$$x_{ijk} = \begin{cases} 1, & \text{if } k-\text{th vehicle travels directly from } i-\text{th to } j-\text{th node.} \\ 0, & \text{otherwise} \end{cases}$$
$$y_{ij} = \begin{cases} 1, & \text{if } j-\text{th customer is served by } i-\text{th depot.} \\ 0, & \text{otherwise} \end{cases}$$
$$z_i = \begin{cases} 1, & \text{if } i-\text{th depot is opened.} \\ 0, & \text{otherwise} \end{cases}$$
u_{jk} : Auxiliary variables for sub − tour elimination constraints in route k.

Using the above notation, the OCLRP may be modeled through the following integer linear programming problem:

$$\min \sum_{i\in V_d} O_i z_i + \sum_{i\in V_d} \sum_{j\in V_c} \sum_{k\in K} w x_{ijk} + \sum_{i\in V_d \bigcup V_c} \sum_{j\in V_c} \sum_{k\in K} c_{ij} e x_{ijk} \quad (1)$$

subject to:

$$\sum_{i\in V_d \bigcup V_c} \sum_{k\in K} x_{ijk} = 1 \qquad \forall j \in V_c \quad (2)$$

$$\sum_{i\in V_d \bigcup V_c} \sum_{j\in V_c} d_j x_{ijk} \leq P \qquad \forall k \in K \quad (3)$$

$$\sum_{j\in V_c} d_j y_{ij} \leq Q_i z_i \qquad \forall i \in V_d \quad (4)$$

$$\sum_{i\in V_d \bigcup V_c} x_{ijk} - \sum_{i\in V_c \bigcup V_p} x_{jik} = 0 \qquad \forall j \in V_c,\ \forall k \in K \quad (5)$$

$$\sum_{i\in V_d} \sum_{k\in K} x_{ijk} = \sum_{i\in V_d} \sum_{k\in K} x_{jik} = 0 \qquad \forall j \in V_d \quad (6)$$

$$\sum_{j\in V_d} x_{ijk} = 0 \forall i \in V_c, \qquad \forall k \in K \quad (7)$$

$$\sum_{i\in V_d} y_{ij} = 1 \qquad \forall j \in V_c \quad (8)$$

$$\sum_{j\in V_c} x_{ijk} - z_i \leq 0 \qquad \forall i \in V_d,\ \forall k \in K \quad (9)$$

$$u_{lk} - u_{jk} + n x_{ljk} \leq n - 1 \qquad \forall l \in V_c,\ \forall j \in V_c,\ \forall k \in K \quad (10)$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i \in V_d \cup V_c,\ \forall j \in V_p \cup V_c,\ \forall k \in K \quad (11)$$

$$y_{ij} \in \{0, 1\} \qquad \forall i \in V_d,\ \forall j \in V_c \quad (12)$$

$$z_i \in \{0, 1\} \qquad \forall i \in V_d \quad (13)$$

$$u_{jk} \in \{0, 1, 2, \ldots, n\} \qquad \forall j \in V_c,\ \forall k \in K \quad (14)$$

As mentioned, the objective function (1) aggregates the costs of opening the depots, the fixed costs of employing vehicles, and the routing costs. Constraint (2) reflects the fact that customers are serviced only once by one vehicle and that they should have just one predecessor. Constraints (3) and (4) ensure, correspondingly, that vehicle and depot capacities are respected. Route continuity is guaranteed through constraint (5). In turn, constraint (6) models the fact that there should be no connection between any two depots, and constraint (7) that each route ends at the last customer. The group of constraints (8) models the issue that each and any customer should be assigned to exactly one depot, whereas (9) guarantees the fact that vehicles dispatch from depots that are open. Constraint (10)

eliminates sub-tours within vehicle routes. Finally, constraints (11)–(13) specify the binary character of some of the variables, and (14) is auxiliary variables.

3 Discrete fireworks algorithm for the OCLRP

To facilitate understanding of the proposed algorithm, we first clarify two core technical terms specific to a fireworks algorithm (FWA). (a) Firework: each firework represents a feasible solution within the search space. (b) Spark: a spark is a new candidate solution generated by applying algorithm-specific operators (e.g., explosion) to an existing firework. Sparks represent potential improvements over the original firework, and their quality is evaluated via the fitness function, which in this study is defined as the OCLRP's objective function.

Tan and Zhu (2010) and Tan (2015) introduced the FWA for optimization problems, inspired by how fireworks explosions resemble optimal solution searches in swarm intelligence algorithms, with each spark generated by a firework explosion assimilated to a feasible solution. It runs iteratively according to the following scheme until certain stopping conditions are met. For each explosion generation, N fireworks are set off from N given locations, from which a certain number of sparks are generated and their fitness assessed. The number and amplitude of the sparks depend on the explosion operator implemented, with some further sparks generated through a mutation operator. A mapping rule eventually maps infeasible sparks into feasible ones. Lastly, a selection strategy is used to select N sparks for the next explosion generation. The original FWA was designed for continuous optimization problems and cannot be directly used to solve the OCLRP (and other discrete optimization problems). For this, we need to modify the explosion and mutation operators used to explore the feasible space.

We now present a DFWA to efficiently solve OCLRPs. The basic steps of our DFWA for solving the OCLRP are presented as pseudocode in Algorithm 1. The input to the algorithm includes the maximum number G of iterations, the number N of fireworks, the number $Musparks$ of mutation sparks, the maximum explosion amplitude $\hat{A}$, and a coefficient $\hat{M}$ controlling the number of explosion sparks generated by each firework. After the fireworks' initialization (line 1), each firework's fitness is assessed (line 2) and the best one is determined (line 3). Lines 4–22 cover the DFWA iterations, including the initialization of the sets of sparks generated by the explosion and mutation operators (line 5), the generation of explosion sparks (lines 6–10), the update of individuals (lines 11 and 19), the calculation of the number of fireworks and explosion sparks (line 12), the generation of mutation sparks (lines 13–18), the resetting of the best individual (line 20), and the selection of the next generation of sparks (line 21). Note that the mapping rule is called upon in lines 9 and 16. To facilitate understanding, Figure 3 illustrates the overall logical flow of the algorithm. The forthcoming subsections provide core algorithmic details.

Input: G = no. iterations, N = no. fireworks, $Musparks$ = no. mutation sparks, $\hat{A}$ = max. explosion amplitude, $\hat{M}$ = parameter controlling no. explosion sparks generated by each firework.
Output: best = best OCLRP solution.
1: Initialize set $\mathcal{F}$ of fireworks;
2: Evaluate each firework $sol_i \in \mathcal{F}$ $(i = 1,2,\ldots,N)$;
3: best\coloneq firework with best fitness in $\mathcal{F}$;
4: for $g \coloneqq 1$ to G do
5: $Esparks \coloneqq \varnothing$; $Msparks \coloneqq \varnothing$;//At beginning of each iteration, $Esparks$ and $Msparks$, sets of sparks generated respectively by the explosion and mutation procedures, initialized as empty.
6: for $i \coloneqq 1$ to N do
7: Calculate no. sparks $\widehat{ns}_i$ for firework sol_i;
8: Calculate amplitude of sparks A_i for firework sol_i;
9: $Esparks \coloneqq Esparks \bigcup$ Explosion $(sol_i, A_i, \widehat{ns}_i)$;//Firework sol_i explodes within amplitude A_i generating $\widehat{ns}_i$ sparks.
10: end for
11: $\mathcal{F} \coloneqq \mathcal{F} \bigcup Esparks$;
12: $M \coloneqq$ size$(\mathcal{F})$;//Calculating no. sparks in set $\mathcal{F}$.
13: $j \coloneqq 0$; //j counts no. mutation sparks.
14: while $j < Musparks$ do
15: $i \coloneqq$ randint$(1,M)$; //Randomly generate an integer in range $(1,M)$.
16: $Msparks \coloneqq Msparks \bigcup$ Mutation (sol_i);//Generating sparks using mutation operator.
17: $j \coloneqq j + 1$;
18: end while
19: $\mathcal{F} \coloneqq \mathcal{F} \bigcup Msparks$;
20: best:= firework with best fitness in $\mathcal{F}$;//Updating best firework.
21: $\mathcal{F} \coloneqq$ select$(\mathcal{F}, N)$;//Select N fireworks for next-generation according to selection strategy.
22: end for
return best

Algorithm 1. Discrete fireworks algorithm for the OCLRP.

3.1 Representation of OCLRP solutions

A compact solution representation is crucial for efficiently implementing a DFWA. In the OCLRP, it should characterize the customer–vehicle assignment, the chosen depots, and the sequence of customers served by a specific vehicle starting at a given depot. Given m and n, our representation uses N_{dummy} dummy zeros (with $N_{\text{dummy}} \leq \lceil \sum_{i \in V_c} (d_i)/P \rceil$, where $\lceil x \rceil$ represents the ceiling of x) and all of the elements within sets V_d and V_c. The first number in the representation is in V_d and indicates the first depot in the route; then the customers assigned to each depot appear between it and the next different depot in the order they are serviced, respecting the vehicle capacity constraint. No customers after a depot indicates that such a depot is not open. The end of a route and the start of a

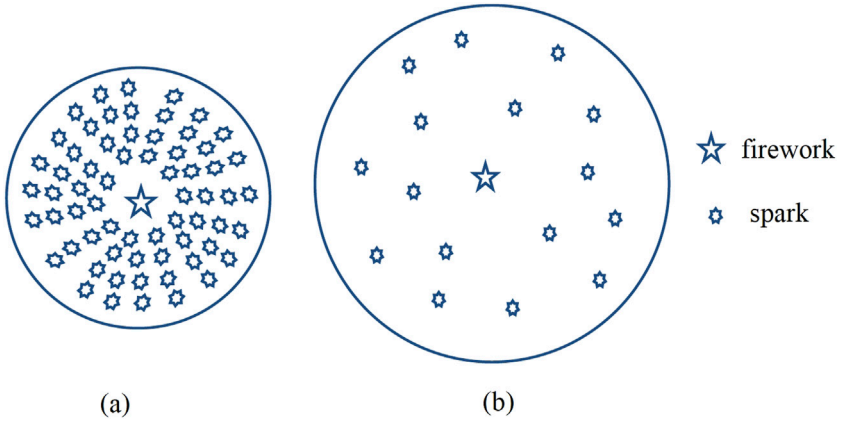

FIGURE 5
Fireworks explosion. **(a)** Good explosion. **(b)** Bad explosion.

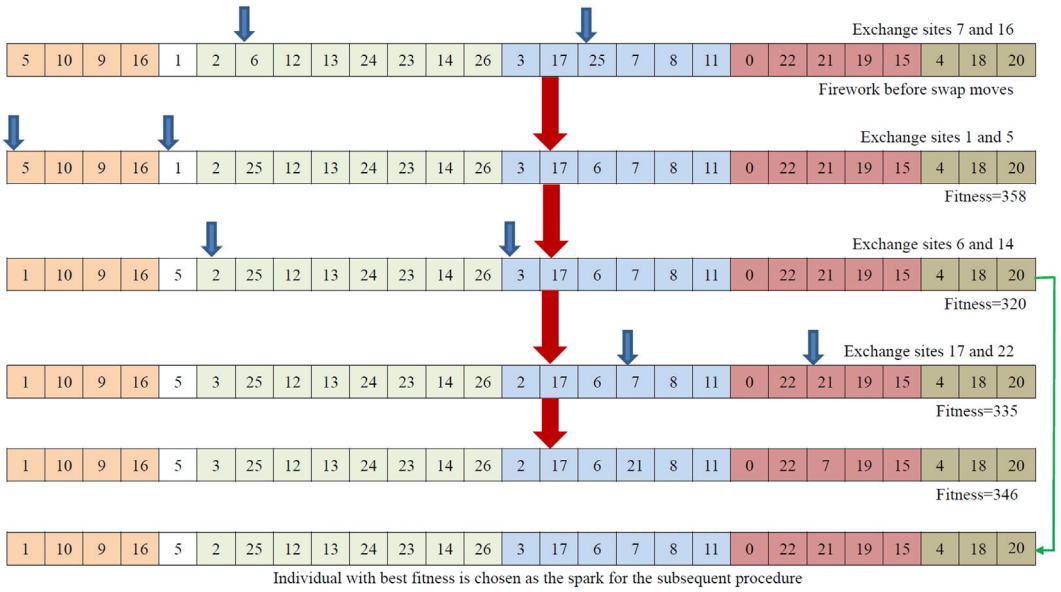

FIGURE 6
Illustration of spark generation with $A_i = 4$.

new one are marked with dummy zeros, even if additional customers may be served by a vehicle with sufficient capacity. Thus, vehicle routes can be randomly terminated with dummy zeros in the representation. As a consequence, the search space is larger and better solutions can be found (Yu and Lin, 2015; Yu and Lin, 2016).

As an illustration, Figure 4 presents a feasible solution of an OCLRP with five depots and 21 customers. Observe that depot 1 is not open (no customers after it). The routes served from the four open depots (2, 3, 4, 5) are

- D 5→c 10→c 9→c 16.
- D 2→c 6→c 12→c 13→c 24→c 23→c 14→c 26.
- D 3→c 17→c 25→c 7→c 8→c 11.
- D 3→c 22→c 21→c 19→c 15.
- D 4→c 18→c 20.

3.2 DFWA initial solutions

We construct a set of good initial solutions by randomly generating the open depots and assigning their customers with a greedy approach that promotes use of the maximum capacity of a vehicle; this initialization process corresponds to line 1 of Algorithm 1 ("Initialize set $\mathcal{F}$ of fireworks"). The detailed steps are as follows:

Step 1. Randomly generate a permutation *Per* of the m depots in V_d. Denote the i-th element in *Per* as per_i and set $i = 1$. Let C be the full set of all customers in V_c. Since no customer is assigned initially, set $Cus := C$ (where *Cus* represents the set of unassigned customers).

Step 2. Designate $C(per_i) \subseteq Cus$ as the subset of unassigned customers whose individual demands do not exceed the

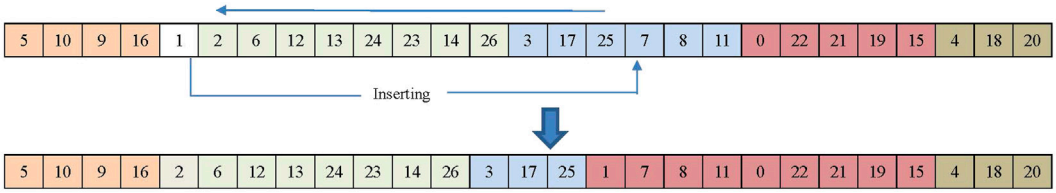

FIGURE 7
Illustration of insertion move.

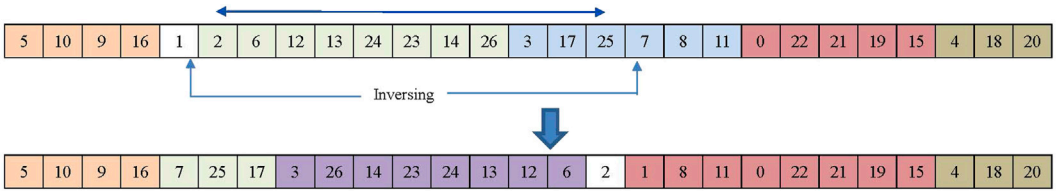

FIGURE 8
Illustration of inversion move.

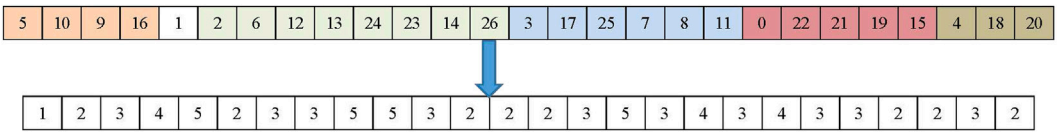

FIGURE 9
Transformation from sol_i to $\widetilde{sol}_i$.

remaining capacity of per_i. To initially allocate the unassigned customers and start a new route from per_i, its closest customer in $C(per_i)$ is assigned, followed by the nearest customer to the one last selected. This is repeated while the vehicle's capacity may satisfy the demand of the selected nearest customer. Otherwise, such a customer will be excluded from the current route, and the closest to the last assigned customer will be searched among those unassigned until the vehicle's remaining capacity is not sufficient to serve new customers. A new route is launched for unassigned customers. This is repeated until $C(per_i)$ is empty.

Step 3. If Cus is not empty, increment i returning to step 2 based on depot per_i in the permutation Per is considered. Otherwise, the solution found is codified as in Subsection 3.1.

Step 4. While the number of solutions/fireworks is smaller than the required N, return to step 1; otherwise, terminate the process.

3.3 Explosion operator

Based on the N initial fireworks, the explosion operator is used to generate sparks within different amplitudes. This has a key role in FWAs (Tan, 2015) and is inspired by how fireworks actually explode. As Figure 5 depicts, well-designed fireworks blast and produce numerous sparks around the explosion center; in turn, bad fireworks explode and generate few sparks scattered around. Algorithmically, a good firework portrays a firework with good fitness value located in a promising area possibly close to an optimal solution; thus, it seems sensible to generate additional sparks around it. On the other hand, bad fireworks suggest that the optimal solution may be far away and, consequently, fewer sparks should be generated at a larger search radius.

3.3.1 Explosion amplitude and number of sparks

In our DFWA, the explosion amplitude and number of sparks associated with a firework (a feasible solution of formulation (1)–(14)) depends on its fitness (objective function value) through Formulas 15 and 16 (Tan, 2015; Tan and Zhu, 2010)

$$ns_i = \hat{M} \cdot \frac{obj_{worst} - f(sol_i) + \xi}{\sum_{j=1}^{N}\left(obj_{worst} - f\left(sol_j\right)\right) + \xi} \tag{15}$$

$$A_i = \left[\hat{A} \cdot \frac{f(sol_i) - obj_{best} + \xi}{\sum_{j=1}^{N}\left(f\left(sol_j\right) - obj_{best}\right) + \xi}\right] \tag{16}$$

TABLE 1 Average results with different number of fireworks for Gaskell67-21×5 and Gaskell67-29×5.

No. Of fireworks	Gap_{ave}		Average computing time (s)		Number of runs that generated the best-known solution	
	Gaskell67-21×5	Gaskell67-29×5	Gaskell67-2×15	Gaskell67-29×5	Gaskell67-21×5	Gaskell67-29×5
$0.5(n+m)$	1.89	2.05	15.41	23.78	6	3
$n+m$	1.23	1.64	17.28	27.19	7	4
$1.5(n+m)$	0.23	0.50	24.09	32.76	11	7
$2(n+m)$	0.29	0.53	31.36	39.58	10	7
$2.5(n+m)$	0.22	0.52	39.15	48.93	12	8
$3(n+m)$	0.23	0.46	47.66	56.42	11	8

TABLE 2 Average results using different values of $\hat{M}$ for Gaskell67-21×5 and Gaskell67-29×5.

$\hat{M}$	Gap_{ave}		Average computing time (s)		Number of runs that generated the best-known solution	
	Gaskell67-21×5	Gaskell67-29×5	Gaskell67-21×5	Gaskell67-29×5	Gaskell67-21×5	Gaskell67-29×5
$0.5N$	2.40	2.33	16.87	22.15	4	4
N	1.22	1.01	23.25	26.39	6	5
$1.5N$	0.47	0.23	29.16	32.18	10	7
$2N$	0.94	0.26	36.37	40.73	11	6
$2.5N$	0.51	0.82	42.33	46.9	9	8
$3N$	0.58	0.37	49.12	57.88	10	7

TABLE 3 Average results using different number of mutation sparks for Gaskell67-21×5 and Gaskell67-29×5.

No. Of mut. sparks	Gap_{ave}		Average computing time (s)		Number of runs that generated the best-known solution	
	Gaskell67-21×5	Gaskell67-29×5	Gaskell67-21×5	Gaskell67-29×5	Gaskell67-21×5	Gaskell67-29×5
$0.5(n+m)$	1.73	1.79	24.91	28.13	6	4
$n+m$	0.60	0.46	28.12	32.55	10	8
$1.5(n+m)$	0.69	0.85	36.48	39.03	9	7
$2(n+m)$	1.02	1.05	39.19	45.28	11	7
$2.5(n+m)$	0.55	1.10	46.78	50.77	10	6
$3(n+m)$	1.25	0.49	53.04	58.29	8	8

where sol_i refers to the i-th firework $(i = 1, 2, \ldots, N)$, $f(sol_i)$ represents its fitness, $\hat{M}$ controls the number of sparks generated by each firework, $\hat{A}$ denotes the maximum explosion amplitude, ξ is a small constant that avoids division by zero errors, and $obj_{worst} = \max_{1\le j\le N}\{f(sol_j)\}$ and $obj_{best} = \min_{1\le j\le N}\{f(sol_j)\}$ are, respectively, the worst and best values of the objective function among the N incumbent fireworks.

To avoid too many or too few sparks using Equation 15, an upper bound $\widehat{ns_i}$ on the number of sparks (Tan, 2015) is defined through (with $round()$ the rounding function)

$$\widehat{ns_i} = \begin{cases} round\left(a \times \hat{M}\right), & \text{if } ns_i < a \times \hat{M} \\ round\left(b \times \hat{M}\right), & \text{if } ns_i > b \times \hat{M}, a < b < 1 \\ round\left(ns_i\right), & \text{otherwise} \end{cases} \tag{17}$$

for properly chosen constants a and b.

3.3.2 Generation of sparks

Our algorithm generates a spark from sol_i through the swap move. We use a simple swap by randomly selecting two sites and swapping the corresponding elements. This is repeated A_i times.

As mentioned, good sparks suggest closeness to an optimal or near-optimal solution. Thus, the explosion process should preserve features of good solutions, and hence only the best-fit spark is selected from the A_i swapped solutions. One spark can be obtained from each swap move, and $\widehat{ns_i}$ sparks are generated thereafter, iteratively carrying out A_i*$\widehat{ns_i}$ swap moves. Note that the iterative swap move used in the explosion operator essentially entails a local search to generate relatively better sparks, possibly improving the performance of our DFWA. Figure 6 illustrates the generation of a spark with $A_i = 4$. Pseudo-code for generating one spark is available in Algorithm 2, whereas Algorithm 3 details pseudo-code for the explosion procedure called by Algorithm 1.

```
function GenerateSpark (sol_i, A_i, ñ) //ñ–the number of elements in the individual sol_i.
  fitness:= ∞; ~sol_i := sol_i;
  for i := 1 to A_i do
    t1 :=randint(1, ñ); //Randomly generate an integer number in the range [1, ñ].
    if t1 == 1 then //If the first position is selected, then the other position must be a depot to maintain feasibility.
      repeat
        t2 :=randint(1, ñ);
      until t2 ≠ t1 and ~sol_{i,t2} ∈ V_d
      temp := ~sol_{i,t1}; ~sol_{i,t1} := ~sol_{i,t2}; ~sol_{i,t2} := temp; //Swapping the elements assigned to sites t1 and t2.
      ~sol_i :=mapping(~sol_i); //If the generated spark is infeasible, it will be mapped back to the feasible space.
    else
      t2 :=randint(1, ñ);
      if t2 == 1 then
        repeat
          t1 :=randint(1, ñ);
      until t1 ≠ t2 and ~sol_{i,t1} ∈ V_d
      temp := ~sol_{i,t1}; ~sol_{i,t1} := ~sol_{i,t2}; ~sol_{i,t2} := temp;
      ~sol_i :=mapping(~sol_i);
      end if
    end if
    if fitness > f(~sol_i) then
      fitness:= f(~sol_i);  ~sol_i := ~sol_i;  //The generated spark with better fitness is saved as ~sol_i.
    end if
  end for
  return ~sol_i;
end function
```

Algorithm 2. Pseudo-code to generate one spark.

```
function Explosion (sol_i, A_i, ^ns_i)
  ñ := size(sol_i); //Calculating the number of elements in the individual sol_i.
  Spark_i = ∅; //Initializing the set of sparks generated by the firework sol_i.
  for i := 1 to ^ns_i do
    SingleSpark :=GenerateSpark(sol_i, A_i, ñ); //Generating one spark.
    Spark_i = Spark_i ⋃ SingleSpark;
  end for
  return Spark_i;
end function
```

Algorithm 3. Explosion procedure pseudo-code.

Recall that once a move is executed, the new solution might need to be recodified to ensure its feasibility. For example, if the first position is selected for a swap, it should be replaced by another depot.

3.4 Mutation operator

As with other swarm intelligence algorithms, maintaining population diversity is crucial for the efficiency of the DFWA to promote full exploration of the most interesting regions of the search space. Mutation is a major diversification technique employed within swarm heuristic algorithms. It refers to the process of increasing population diversity to mitigate stagnation in unpromising areas and promote the exploration of new search regions by introducing random variations in the individual solutions (Zhang et al., 2022b). In this study, the mutation operator is implemented through two search mechanisms—insertion and inversion moves—which are chosen with equal probabilities. Algorithm 4 presents pseudo-code for the mutation procedure.

```
function Mutation (sol_i)
  ñ := size(sol_i); //Calculating the number of elements in the individual sol_i.
  r := rand(0,1); //Randomly generating a number in the range (0,1);
  f r≤0.5 then
    MSpark := InsertionMove(sol_i, ñ);
  else
    MSpark := InverseMove(sol_i, ñ);
  end if
  return MSpark;
end function
```

Algorithm 4. Pseudo-code for the mutation procedure.

3.4.1 Insertion move

This move randomly selects the i-th and j-th positions ($i \neq 1$ and $i < j$) from a solution permutation and then inserts the element in the i-th position in front of the element in the j-th position. Figure 7 and Algorithm 5 show an illustration and pseudo-code of the insertion move, respectively.

```
function InsertionMove (sol_i, ñ)
  InsertSpark ≔ sol_i ;
  repeat
    t1 ≔randint(1, ñ − 1);
    t2 ≔randint(1, ñ − 1);
  until t2 − t1 > 1
  t1 ≔ t1 + 1; t2 ≔ t2 + 1; //Updating t1 and t2. To maintain feasibility, the first position is not selected in insertion move.
  temp ≔ InsertSpark_t1 ;
  for t ≔ t1 + 1 to t2 − 1 do
    InsertSpark_t−1 ≔ InsertSpark_t ;
  end for
  InsertSpark_t2−1 ≔ temp ;
  InverseSpark ≔mapping(InsertSpark);
  return InsertSpark ;
end function
```

Algorithm 5. Pseudo code for insertion move.

```
function InverseMove (sol_i, ñ)
  InverseSpark ≔ sol_i ;
  t1 ≔randint(1, ñ);
  if t1 == 1 then // If the first position is selected, then the other position must be a depot to maintain feasibility.
    repeat
      t2 ≔randint(1, ñ);
    until t2 ≠ t1 and InverseSpark_t2 ∈ V_d
  else
    t2 ≔randint(1, ñ);
    if t2 == 1 then
      repeat
        t1 ≔randint(1, ñ);
      until t1 ≠ t2 and InverseSpark_t1 ∈ V_d
    end if
  end if
  t ≔ min{t1, t2}; t2 ≔ max{t1, t2};
  t1 ≔ t ; t3 ≔ ⌈(t1+t2)/2⌉;
  for t ≔ t1 to t3 do
    temp ≔ InverseSpark_t ;
    InverseSpark_t ≔ InverseSpark_t2 ;
    InverseSpark_t2 ≔ temp ;
    t2 = t2 − 1 ;
  end for
  InverseSpark ≔mapping(InverseSpark);
  return InverseSpark ;
end function
```

Algorithm 6. Pseudo code for inversion move.

3.4.2 Inversion move

Inversion move. This move randomly selects the i-th and j-th positions ($i \neq 1$ and $i < j$) from a solution permutation and then inverts the sub-permutation between both positions. Figure 8 and Algorithm 6 illustrate the inversion move and its pseudo-code, respectively.

3.5 Mapping rule

Some of the generated sparks using the swap move and mutation procedures could be infeasible, and they should be mapped back to the feasible region. Indeed, if a spark generated by a move is infeasible, at least one of the following two cases holds.

Case 1. The resulting customers' demand assigned to the i-th depot exceeds its capacity. In this case, a customer is randomly selected from the set $Cusd(i)$ of customers allocated to depot i and assigned to a randomly selected open depot j, the remaining capacity of which can satisfy the demand of the selected customer. If no open depot can satisfy the selected customer, a closed depot is randomly selected to serve such customer. This procedure is repeatedly carried out until all open depots may satisfy their customers' demand.

Case 2. The resulting customers' demand served by the k-th vehicle exceeds its capacity. In this case, a customer is randomly selected from the set $Cusv(k)$ of customers served by the k-th vehicle and assigned to a randomly selected vehicle starting at the same depot as vehicle k, with remaining capacity available to satisfy the selected customer's demand. If no such vehicle exists, a new vehicle will be launched in the same parking lot as vehicle k to serve the selected customers. This procedure is repeated until all vehicles may satisfy the demands of the customers they serve.

Algorithm 7 presents pseudo-code for dealing with both cases. The random selection mechanism in the mapping rule ensures simplicity and avoids excessive computation for small-to-medium OCLRP instances but may lead to inefficiencies in large-scale cases (e.g., 100+ customers or 10+ depots). Repeated random selections can result in redundant reallocations, such as frequently reassigning small-demand customers or choosing depots and vehicles with minimal residual capacity, thus increasing the need for additional repair steps. To address this issue, two optimized strategies are proposed for future implementation while preserving the core logic of the mapping rule:

1. Residual capacity-greedy selection: for depot-level capacity violations (Case 1), prioritize reallocating the customer with the highest demand from the overloaded depot to an open depot with the largest available capacity; for vehicle-level violations (Case 2), assign the largest-demand customer to the vehicle within the same depot that has the maximum residual capacity.
2. Distance-aware selection: when reallocating, combine residual capacity with routing distance by selecting the depot or vehicle that offers sufficient capacity and the smallest additional travel distance (either from the customer or from the last customer in the vehicle's current route).

It is worth noting that the current random selection plays a crucial role in maintaining population diversity in DFWA, thus helping avoid premature convergence to local optima. For large-scale problems, a hybrid approach (e.g., 80% greedy selection and 20% random selection) could potentially balance computational efficiency and solution diversity. However, such an approach would require further parameter tuning and validation using large-scale benchmarks (e.g., instances with 200+ customers); this is reserved for future research to avoid complicating the current DFWA framework.

```
function Mapping ($\hat{S}$)
  $Odep$ ≔ the set of open depots included in the solution permutation $\hat{S}$;
  $Cdep$ ≔ the set of closed depots included in the solution permutation $\hat{S}$;
  $Onum$ ≔ the number of open depots included in $Odep$;
  for $i$ ≔ 1 to $Onum$ do
    $Sum\_Cus\_dep$ ≔ the total demand of the customers allocated to depot $Odep_i$;
    $Set\_Cus\_dep$ ≔ the set of customers allocated to depot $Odep_i$;
    while $Sum\_Cus\_dep > Q_i$ do //Eliminating the case 1: the total demand of the customers assigned to depot $i$ ($i \in V_d$) exceed its capacity $Q_i$.
      $Cus\_dep$ ≔ randomly select one customer from the set $Set\_Cus\_dep$;
      $Sum\_Cus\_dep$ ≔ $Sum\_Cus\_dep - d_{Cus\_dep}$;
      $Set\_Cus\_dep$ ≔ $Set\_Cus\_dep - \{Cus\_dep\}$;
      $\overline{Set\_Cus\_dep}$ ≔ the set of opening depots whose remaining capacity satisfy the demand $d_{Cus\_dep}$;
      if $\overline{Set\_Cus\_dep} \neq \varnothing$ then
        $\widetilde{dep}$ ≔ randomly select one depot from $\overline{Set\_Cus\_dep}$;
        Randomly insert customer $Cus\_dep$ into one of the distribution routes starting at depot $\widetilde{dep}$;
      else
        $\widetilde{dep}$ ≔ randomly select one depot from $Cdep$;
        Allocate the customer $Cus\_dep$ to depot $\widetilde{dep}$;
        $Odep$ ≔ $Odep \bigcup \{\widetilde{dep}\}$;
      end if
    end while
    $Vdepot$ ≔ the set of vehicles which start at the depot $Odep_i$;
    $Vnum$ ≔ the number of the vehicles which start at the depot $Odep_i$;
    for $j$ ≔ 1 to $Vnum$ do
  $Sum\_Cus\_V$ ≔ the total demand of the customers assigned to vehicle $Vdepot_j$;
  $Set\_Cus\_V$ ≔ the set of customers assigned to vehicle $Vdepot_j$;
  while $Sum\_Cus\_V > P$ do //Eliminating the case 2: the total demand of the customers served by vehicle $k$ ($k \in K$) exceed its capacity $P$.
  $Cus\_V$ ≔ randomly select one customer from the set $Set\_Cus\_V$;
  $Sum\_Cus\_V$ ≔ $Sum\_Cus\_V - d_{Cus\_V}$;
  $Set\_Cus\_V$ ≔ $Set\_Cus\_V - \{Cus\_V\}$;
  $\overline{Set\_Cus\_V}$ ≔ the set of vehicles that start at depot $Odep_i$ and whose remaining capacity satisfy the demand $d_{Cus\_V}$;
  if $\overline{Set\_Cus\_V} \neq \varnothing$ then
    $\widetilde{EV}$ ≔ randomly select one vehicle from $\overline{Set\_Cus\_V}$;
    Randomly insert customer $Cus\_V$ into the distribution route of vehicle $\widetilde{EV}$;
  else
    A new vehicle denoted as $\widetilde{VN}$ is employed to serve the customer $Cus\_V$;
    $Set\_Cus\_V \bigcup \{\widetilde{VN}\}$;
  end if
end while
    end for
    end for
    return the mapped feasible solution;
end function
```

Algorithm 7. Pseudo-code of the mapping rule.

3.6 Improving the routes

For each solution generated using the explosion and mutation operators, the distribution routes included in it are improved using a greedy search method. The customers assigned to the same distribution route may be rerouted in this step. First, for each route, the customer nearest the depot which provides service to it is routed. Next, the customer nearest the last rerouted is selected. This is repeated until all customers assigned to the current vehicle are rerouted. Algorithm 8 presents pseudo-code to improve the distribution routes.

```
function ImprovingRoutes ($\hat{S}$)
  $Odep$ ≔ the set of open depots included in the solution permutation $\hat{S}$;
  $Onum$ ≔ the number of open depots included in $Odep$;
  for $i$ ≔ 1 to $Onum$ do
    $Vdepot$ ≔ the set of vehicles which start at the depot $Odep_i$;
    $Vnum$ ≔ the number of the vehicles which start at the depot $Odep_i$;
    for $j$ ≔ 1 to $Vnum$ do
      $Set\_Cus\_V$ ≔ the set of customers assigned to vehicle $Vdepot_j$;
      $Cusnum$ ≔ the number of the customers assigned to vehicle $Vdepot_j$;
      for $k$ ≔ 1 to $Cusnum$ do
        if $k$ == 1 then
          the nearest customer from the set $Set\_Cus\_V$ to the depot $Odep_i$, denoted as $Ncus$, is routed;
        else
          the nearest customer from the set $Set\_Cus\_V$ to the last rerouted customer, denoted as $Ncus$, is selected;
        end if
        $Set\_Cus\_V$ ≔ $Set\_Cus\_V - \{Ncus\}$;
      end for
    end for
  end for
  return the improved solution;
end function
```

Algorithm 8. Pseudo-code to improve routes.

TABLE 4 Average results using different values of $\hat{A}$ for Gaskell67-21×5 and Gaskell67-29×5.

$\hat{A}$	Gap_{ave}		Average computing time (s)		Number of runs that generated the best-known solution	
	Gaskell67-21×5	Gaskell67-29×5	Gaskell67-21×5	Gaskell67-29×5	Gaskell67-21×5	Gaskell67-29×5
$0.5N$	1.21	1.32	19.12	25.73	6	4
N	1.05	1.01	24.46	29.3	9	5
$1.5N$	0.89	0.63	29.1	32.45	10	6
$2N$	0.93	0.59	33.96	37.67	9	7
$2.5N$	1.02	0.78	39.14	42.82	9	7
$3N$	0.72	0.75	44.33	49.59	10	6

3.7 Selection strategy

Based on the explosion and mutation sparks and the current fireworks, N solutions are selected to form the next generation of fireworks. The selection uses a fitness-and-Hamming-distance-based strategy aimed at promoting good solution features as well as diversity.

Prior to calculating the Hamming distance, each candidate sol_i is rewritten as $\widetilde{sol}_i$, encoded through the depots' indices and including $m+n$ elements. In $\widetilde{sol}_i$, the m depots (numbered 1 to m) are placed in the first m positions in order, with the element $\widetilde{sol}_{ij}$ $(j>m)$ representing the depot to which the j-th customer is allocated. Figure 9 illustrates the transformation from sol_i to $\widetilde{sol}_i$. In it, customers 10, 9, and 16 are allocated to depot 5 in sol_i. Thus, the $\widetilde{sol}_i$ elements located in positions 10, 9, and 16 are five. Then, for each solution sol_i, an index $R(sol_i)$ is computed defined as the sum of Hamming distances from sol_i to the other individuals, computed through

$$R(sol_i)=\sum_{j\in\mathcal{F},\ j\neq i} Dis\left(sol_i,sol_j\right)=\sum_{j\in\mathcal{F},\ j\neq i} Dis\left(\widetilde{sol}_i,\widetilde{sol}_j\right), \tag{18}$$

where $Dis(sol_i,sol_j)$ is the Hamming distance between sol_i and sol_j (the Hamming distance between two permutations is the number of positions at which the corresponding elements are different) and $\mathcal{F}$ denotes the set of candidate solutions (consistent with the notation in line 21 of Algorithm 1 and line 1 of Algorithm 9).

Our implementation passes down the individual with the best fitness value down to the next generation. The remaining $N-1$ next-generation individuals are selected using the roulette wheel principle, with probability $Pro(sol_i)$ of choosing the individual sol_i computed through Formula 19:

$$Pro(sol_i)=\frac{\frac{1}{\left(f(sol_i)\right)^2}+R(sol_i)}{\sum_{i\in\mathcal{F}}\frac{1}{\left(f(sol_i)\right)^2}+\sum_{i\in\mathcal{F}}R(sol_i)} \tag{19}$$

From Equation 19, observe that individuals with larger distances and better fitness will have more chances of being passed on to the next generation. Thus, this selection strategy promotes both the good characteristics and diversity of the population. In Equation 19, the quadratic power increases the probability of selecting good individuals compared to the first power. Algorithm 9 presents the pseudo-code implementing the selection strategy.

```
1:  function Select (ℱ, N)//ℱ is the set of candidate individuals.
2:    ℱ̃ = ∅; //ℱ̃ is the set of selected individuals for next generation.
3:    sumfitness := 0;
4:    Num :=size(ℱ); //Calculating the number of sparks in the set ℱ.
5:    best := the firework with best fitness in the set ℱ;
6:    ℱ̃ := ℱ̃ ⋃ {best}; //The individual with best fitness value is passed down to the next generation.
7:    ℱ := ℱ − {best};
8:    for i := 1 to Num − 1 do
9:      Calculate the probability Pro(sol_i) for each firework or spark sol_i ∈ ℱ by using the Equation 18;
10:     sumfitness := sumfitness + Pro(sol_i);
11:   end for
12:   for i := 1 to N−1 do //N−1 individuals are selected using the roulette wheel principle.
13:     pick :=random(0,1); //Randomly generate a number in the range (0,1).
14:     sum := 0;
15:     j := 1;
16:     while sum < = pick do
17:       sum := sum + Pro(sol_j)/sumfitness;
18:       j := j + 1;
19:     end while
20:     ℱ̃ := ℱ̃ ⋃ {sol_{j−1}}
21:   end for
22:   return ℱ̃;
23: end function
```

Algorithm 9. Pseudo-code to select next-generation fireworks.

4 Computational analysis

This section reports extensive computational experiments undertaken to assess the performance of the DFWA proposed to solve OCLRPs. For reasons outlined above, we use the data of benchmark CLRP instances without any adjustment to test our

TABLE 5 Parameter combinations.

No.	No.of fireworks	$\hat{M}$	No. Of mut. sparks	$\hat{A}$
1	$1.5(m+n)$	$1.5N$	$m+n$	$1.5N$
2	$1.5(m+n)$	$1.5N$	$m+n$	$2N$
3	$1.5(m+n)$	$1.5N$	$1.5(m+n)$	$1.5N$
4	$1.5(m+n)$	$1.5N$	$1.5(m+n)$	$2N$
5	$1.5(m+n)$	$2N$	$m+n$	$1.5N$
6	$1.5(m+n)$	$2N$	$m+n$	$2N$
7	$1.5(m+n)$	$2N$	$1.5(m+n)$	$1.5N$
8	$1.5(m+n)$	$2N$	$1.5(m+n)$	$2N$
9	$1.5(m+n)$	$2.5N$	$m+n$	$1.5N$
10	$1.5(m+n)$	$2.5N$	$m+n$	$2N$
11	$1.5(m+n)$	$2.5N$	$1.5(m+n)$	$1.5N$
12	$1.5(m+n)$	$2.5N$	$1.5(m+n)$	$2N$
13	$2(m+n)$	$1.5N$	$m+n$	$1.5N$
14	$2(m+n)$	$1.5N$	$m+n$	$2N$
15	$2(m+n)$	$1.5N$	$1.5(m+n)$	$1.5N$
16	$2(m+n)$	$1.5N$	$1.5(m+n)$	$2N$
17	$2(m+n)$	$2N$	$m+n$	$1.5N$
18	$2(m+n)$	$2N$	$m+n$	$2N$
19	$2(m+n)$	$2N$	$1.5(m+n)$	$1.5N$
20	$2(m+n)$	$2N$	$1.5(m+n)$	$2N$
21	$2(m+n)$	$2.5N$	$m+n$	$1.5N$
22	$2(m+n)$	$2.5N$	$m+n$	$2N$
23	$2(m+n)$	$2.5N$	$1.5(m+n)$	$1.5N$
24	$2(m+n)$	$2.5N$	$1.5(m+n)$	$2N$
25	$2.5(m+n)$	$1.5N$	$m+n$	$1.5N$
26	$2.5(m+n)$	$1.5N$	$m+n$	$2N$
27	$2.5(m+n)$	$1.5N$	$1.5(m+n)$	$1.5N$
28	$2.5(m+n)$	$1.5N$	$1.5(m+n)$	$2N$
29	$2.5(m+n)$	$2N$	$m+n$	$1.5N$
30	$2.5(m+n)$	$2N$	$m+n$	$2N$
31	$2.5(m+n)$	$2N$	$1.5(m+n)$	$1.5N$
32	$2.5(m+n)$	$2N$	$1.5(m+n)$	$2N$
33	$2.5(m+n)$	$2.5N$	$m+n$	$1.5N$
34	$2.5(m+n)$	$2.5N$	$m+n$	$2N$
35	$2.5(m+n)$	$2.5N$	$1.5(m+n)$	$1.5N$
36	$2.5(m+n)$	$2.5N$	$1.5(m+n)$	$2N$

10.3389/fieng.2025.1686126

TABLE 6 Results for parameter combinations.

No.	Gap_{ave}		Average computing time (s)		Number of runs that generated the best-known solution	
	Gaskell67-21×5	Gaskell67-29×5	Gaskell67-21×5	Gaskell67-29×5	Gaskell67-21×5	Gaskell67-29×5
1	0.13	0.2	9.5	30.56	15	12
2	0.24	0.23	14.7	37.98	13	11
3	0.23	0.24	13.9	35.66	13	10
4	0.22	0.24	21.5	42.32	13	10
5	0.17	0.22	17.9	38.73	14	11
6	0.25	0.25	20.8	47.98	12	9
7	0.26	0.21	22.3	44.35	12	12
8	0.26	0.22	26.72	50.63	12	11
9	0.17	0.21	15.07	34.98	14	12
10	0.24	0.24	20.32	48.55	13	10
11	0.25	0.25	19.84	50.65	12	12
12	0.27	0.25	27.52	54.52	10	10
13	0.28	0.25	34.26	45.63	10	9
14	0.25	0.24	37.28	59.53	12	10
15	0.28	0.25	35.43	50.46	10	8
16	0.27	0.25	39.82	62.73	11	10
17	0.17	0.24	33.36	47.66	14	10
18	0.14	0.2	38.58	53.75	15	12
19	0.23	0.21	39.18	48.23	13	11
20	**0.17**	**0.21**	**42.37**	**55.36**	**14**	**11**
21	0.26	0.23	41.56	57.64	12	11
22	0.28	0.26	48.29	66.27	10	10
23	0.27	0.27	43.38	53.81	11	10
24	0.13	0.22	47.53	64.78	15	11
25	0.23	0.21	40.27	47.96	13	12
26	0.13	0.19	44.32	56.38	15	12
27	0.25	0.22	48.45	66.43	12	11
28	0.27	0.25	53.75	72.48	11	9
29	0.18	0.24	45.23	52.34	14	10
30	0.24	0.23	54.92	63.66	13	10
31	0.27	0.26	50.69	67.23	10	9
32	0.28	0.27	59.85	75.86	10	10
33	0.27	0.27	63.83	68.57	11	10
34	0.29	0.32	61.18	67.33	9	9
35	0.29	0.3	66.29	71.92	10	10
36	0.31	0.31	73.67	79.64	9	9

The best results are indicated in bold in the tables.

TABLE 7 Comparison among DFWA, FWA1, FWA2, FWA3, and FWA4.

Instance	Metric	DFWA	FWA1	FWA2	FWA3	FWA4
Gaskell67-21 × 5	Mean obj	321.85	330.84	355.82	325.86	325.35
	SD	2.90	8.49	19.52	5.70	4.79
	Avg. time (s)	9.51	8.48	8.05	8.15	8.96
	Best runs	15.00	6.00	1.00	8.00	7.00
	p-value (vs. DFWA)		0.00	0.00	0.02	0.00
Gaskell67-22 × 5	Mean obj	460.77	475.44	505.33	470.71	471.13
	SD	4.08	9.85	15.75	7.00	7.04
	Avg. time (s)	17.28	16.37	15.48	16.32	16.83
	Best runs	14.00	3.00	0.00	4.00	4.00
	p-value (vs. DFWA)		0.00	0.00	0.00	0.00
Gaskell67-29 × 5	Mean obj	389.46	402.93	418.75	392.70	393.35
	SD	5.43	10.48	10.88	6.15	6.08
	Avg. time (s)	30.58	28.45	28.03	28.25	29.17
	Best runs	14.00	4.00	0.00	9.00	8.00
	p-value (vs. DFWA)		0.00	0.00	0.06	0.03
50-5-1	Mean obj	64421.67	65432.32	65470.69	64880.50	64949.55
	SD	158.03	579.78	635.39	386.65	431.31
	Avg. time (s)	87.41	83.21	81.72	84.66	85.82
	Best runs	5.00	2.00	1.00	3.00	3.00
	p-value (vs. DFWA)		0.00	0.00	0.00	0.00
50-5-3b	Mean obj	47615.66	48610.24	49230.19	48849.98	48600.44
	SD	708.87	1,005.38	1,604.08	795.62	1,163.28
	Avg. time (s)	93.68	86.24	88.58	89.07	90.85
	Best runs	2.00	0.00	0.00	0.00	1.00
	p-value (vs. DFWA)		0.01	0.00	0.00	0.01
100-5-1	Mean obj	224980.44	230378.59	239404.28	231091.49	229701.55
	SD	2,411.08	6,196.99	6,119.07	4,825.30	4,706.74
	Avg. time (s)	185.94	162.27	173.81	176.68	178.47
	Best runs	4.00	0.00	0.00	0.00	2.00
	p-value (vs. DFWA)		0.00	0.00	0.00	0.00
100-10-1	Mean obj	277251.15	279615.43	281385.79	279598.79	279483.10
	SD	444.62	1806.06	2,467.45	1,635.20	1,470.96
	Avg. time (s)	308.19	272.53	285.67	290.94	297.63
	Best runs	0.00	0.00	0.00	0.00	0.00
	p-value (vs. DFWA)		0.00	0.00	0.00	0.00
100-10-3b	Mean obj	188834.81	191321.05	193503.69	190572.50	190050.84
	SD	465.63	1,509.92	1775.70	1,186.42	1,034.21
	Avg. time (s)	234.48	200.86	213.92	220.54	228.84

(Continued on following page)

TABLE 7 (*Continued*) Comparison among DFWA, FWA1, FWA2, FWA3, and FWA4.

Instance	Metric	DFWA	FWA1	FWA2	FWA3	FWA4
	Best runs	5.00	0.00	0.00	1.00	2.00
	p-value (vs. DFWA)		0.00	0.00	0.00	0.00

algorithm. Implementation is based on Matlab R2016a, and experiments were conducted on a laptop equipped with an A10-7400P CPU at 3.40 GHz and 4 GB of RAM under Windows 10.

4.1 Parameter setting and sensitivity analysis

It is well-known that the proper tuning of parameters may considerably affect the performance of most meta-heuristic algorithms (Zhang et al., 2022a). In our DFWA case, we carried out experiments with two randomly selected benchmark instances—Gaskell67-21×5 and Gaskell67-29×5 (Yu and Lin, 2015) —averaged over 20 independent runs over each instance, using the average gap with respect to the best known solution, the average computing time, and the average number of runs to reach the best solution depending on the following key parameters: number of fireworks, parameter controlling the number of explosion sparks, maximum explosion amplitude, and number of mutation sparks. Only one parameter was modified at a time.

4.1.1 Number of fireworks

To properly set the number of fireworks, we investigated six different values (0.5, 1, 1.5, 2, 2.5, 3) as multipliers of the number $n+m$ of customers and depots in the instance (i.e., $0.5(n+m)$, $n+m$, ...). Table 1 presents, for each number of fireworks, average results over the 20 runs for both instances for the percent deviation from the best-known solution ($\text{Gap}_{\text{ave}} = \frac{\text{BOV}_{\text{ave}}-\text{BKOV}}{\text{BKOV}} \times 100\%$), where BOV_{ave} is the average objective value over 20 runs and BKOV is the best-known objective value reported in the literature, the average computing time in seconds, and the number of runs until generating the best solution.

According to this table, the number of fireworks plays a key role in the performance of our DFWA. When a small number is employed (like $0.5(n+m)$), the algorithm tends to end up in a local optimum, with a worse average objective value and a smaller number of runs. On the other hand, for large numbers of fireworks (like $3(n+m)$), better average results are attained, albeit at much longer computational times. Based on the solution quality and computing time, we would suggest using $N = 1.5(n+m)$, $2(n+m)$, or $2.5(n+m)$ fireworks in our framework.

4.1.2 Parameter $\hat{M}$

The number of explosion sparks is also a crucial parameter in searching for good solutions. A small number of explosion sparks might not strengthen exploration in a promising search region, decreasing the possibility of obtaining the optimal solution. However, a larger number of explosion sparks typically ends up being very similar to an exhaustive method, greatly increasing computing time. Table 2 presents average results over 20 runs for both instances using different values of $\hat{M}$ which controls the number of sparks generated by a firework as a multiple of the number N of fireworks, from $0.5N$ to $3N$. This table suggests that better values for $\hat{M}$ in our experiments are $1.5N$, $2N$, and $2.5N$, based on the number of runs and the values of Gap_{ave}.

4.1.3 Number of mutation sparks

For the proposed DFWA, properly balancing exploration and exploitation critically depends on choosing a suitable number of mutation sparks. A small number hinders the procurement of the optimal solution as the entailed lower population diversity interferes with escaping from a local optimum toward novel unexplored areas. However, a larger number of mutation sparks resembles a naive random multi-start approach. To select a suitable number of mutation sparks, six different multiples of $(n+m)$ were considered. Table 3 presents average results over 20 runs for both test instances, depending on the number of mutation sparks. Comparatively better number of runs and optimality gaps were globally obtained with $n+m$ and $1.5(n+m)$.

4.1.4 Parameter $\hat{A}$

In our algorithm, this parameter denotes the maximum swap moves for generating one explosion spark. When $\hat{A}$ is small, the sparks generated using swap moves cannot be sufficiently improved; however, when $\hat{A}$ is too large, it is also possible to become trapped in meaningless iterations. To properly set $\hat{A}$, six values are considered (as multiples of N). Table 4 presents average results over 20 runs for our test instances. Comparatively better values of $\hat{A}$ are obtained in our experiments with $1.5N$ and $2N$ based on the relatively better average number of runs and value of Gap_{ave}.

4.1.5 Parameter combinations

We undertook additional experiments to identify good parameter combinations. Table 5 lists the 36 combinations tested, based on those found above, for which ten runs over our two selected Gaskell instances took place.

Table 6 presents the average results for each parameter combination.

A closer look at Table 6 reveals that combination 26 shows marginal advantages in solution quality and success rate—its Gap_{ave} for Gaskell67-21×5 and Gaskell67-29×5 is 0.13% and 0.19% (slightly lower than combination 20s 0.17% and 0.21%), and the number of runs generating the best-known solution is 15 and 12 (same as combination 20s 14 and 11). However, combination 20, associated with parameter choices $2(m+n)$, $2N$, $1.5(m+n)$, and $2N$, was ultimately selected as the optimal parameter set, the rationale lying in the balance between computational efficiency and solution quality, especially considering practical application scenarios. For the two test instances, combination 20 exhibits slightly shorter computation time than combination 26: 42.37 s vs. 44.32 s for Gaskell67-21×5 (≈2-s difference) and 55.36 s vs. 56.38 s for

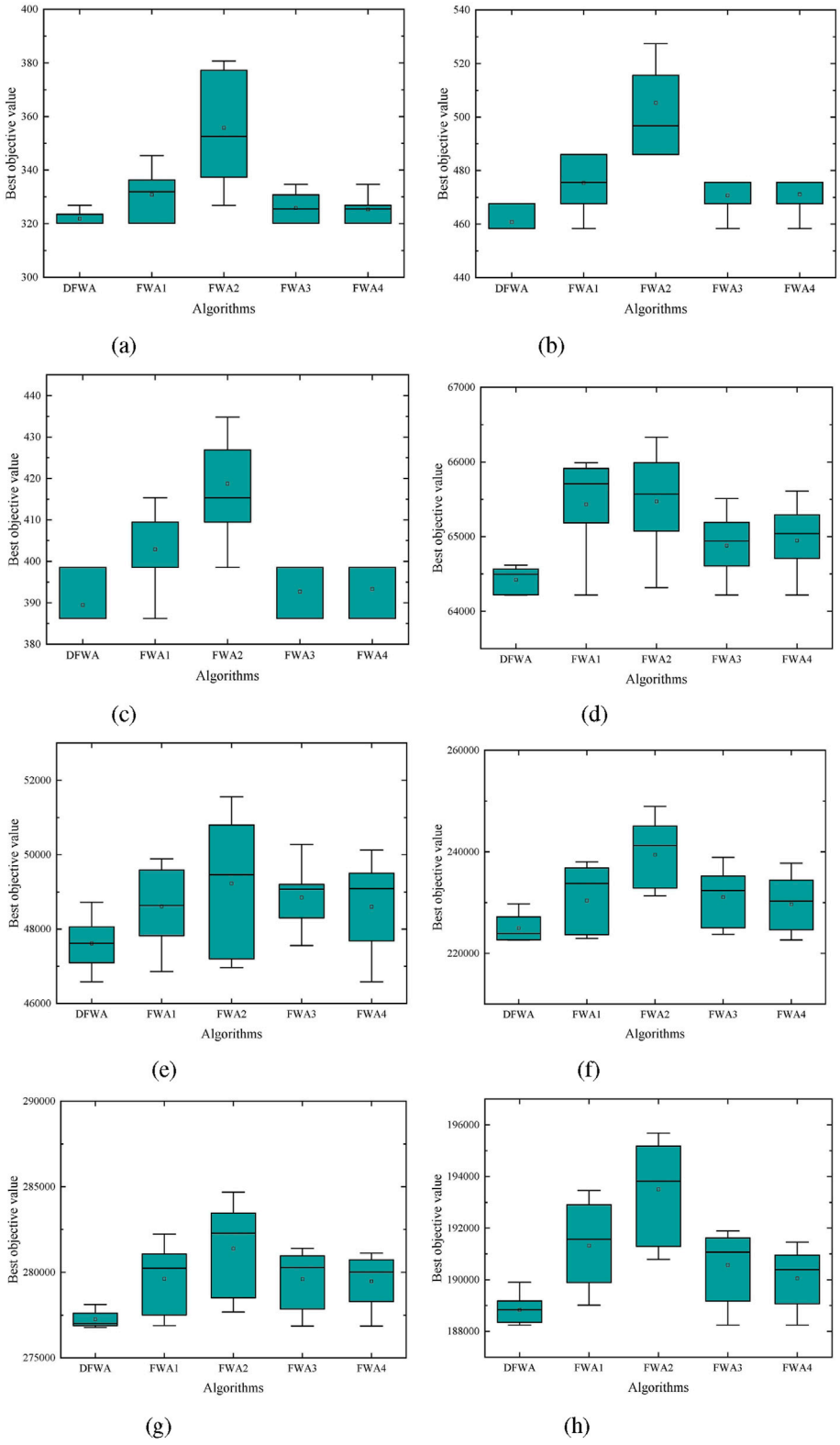

FIGURE 10
Box plots of DFWA, FWA1, FWA2, FWA3, and FWA4 on four small instances. **(a)** Gaskell67-21 × 5. **(b)** Gaskell67-22 × 5. **(c)** Gaskell67-29 × 5. **(d)** 50-5-1. **(e)** 50-5-3b. **(f)** 100-5-1. **(g)** 100-10-1. **(h)** 100-10-3b.

TABLE 8 Comparison of results with the OCLRP solved with CPLEX and best-known results of corresponding CLRP.

Instance	*n*	*m*	*Q*	CLRP	CPLEX (Yu and Lin, 2015)		
				Obj	Obj	CPU (s)	Gap_{CLRP} (%)
Srivastava86-8 × 2	8	2	200	-	**377.09**	0.05	-
Perl83-12 × 2	12	2	140	204.00	**176.34**	0.08	−13.56
Gaskell67-21 × 5	21	5	6,000	424.90	**320.17**	0.64	−24.65
Gaskell67-22 × 5	22	5	4,500	585.10	**458.33**	0.92	−21.67
Gaskell67-29 × 5	29	5	4,500	512.10	**386.26**	39.13	−24.57
Gaskell67-32 × 5_1	32	5	8,000	562.22	**395.57**	116.11	−29.64
Gaskell67-32 × 5_2	32	5	11000	504.30	**374.70**	11.25	−25.70
Min92-27 × 5	27	5	2,500	3,062.00	**2,110.03**	19.19	−31.09
Daskin95-88 × 8	88	8	9000000	355.80	288.54	14400.00	−18.90
20-5-1	20	5	70	54793.00	**43849.00**	33.27	−19.97
20-5-1b	20	5	150	39104.00	**33564.00**	2.36	−14.17
20-5-2	20	5	70	48908.00	**41125.00**	11.58	−15.91
20-5-2b	20	5	150	37542.00	**32520.00**	1.53	−13.38
50-5-1	50	5	70	90111.00	64358.00	14400.00	−28.58
50-5-1b	50	5	150	63242.00	**49114.00**	3,400.59	−22.34
50-5-2	50	5	70	88298.00	**68121.00**	11171.33	−22.85
50-5-2b	50	5	150	67308.00	57815.00	14400.00	−14.10
50-5-2bBIS	50	5	70	51822.00	**41193.00**	1,608.70	−20.51
50-5-2BIS	50	5	150	84055.00	60052.00	14400.00	−28.56
50-5-3	50	5	70	86203.00	62581.00	14400.00	−27.40
50-5-3b	50	5	150	61830.00	**46584.00**	11238.06	−24.66
100-5-1	100	5	70	274814.00	228085.00	14400.00	−17.00
100-5-1b	100	5	150	213615.00	190417.00	14400.00	−10.86
100-5-2	100	5	70	193671.00	167991.00	14400.00	−13.26
100-5-2b	100	5	150	157095.00	146668.00	14400.00	−6.64
100-5-3	100	5	70	200079.00	252087.00	14400.00	25.99
100-5-3b	100	5	150	152441.00	138364.00	14400.00	−9.23
100-10-1	100	10	70	287983.00	308507.00	14400.00	7.13
100-10-1b	100	10	150	231763.00	273445.00	14400.00	17.98
100-10-2	100	10	70	243590.00	295374.00	14400.00	21.26
100-10-2b	100	10	150	203988.00	260437.00	14400.00	27.67
100-10-3	100	10	70	250882.00	265158.00	14400.00	5.69
100-10-3b	100	10	150	204317.00	197206.00	14400.00	−3.48
Average	59.12	5.82	273923.64	106052.01	100894.00	8,256.21	−12.41

The best results are indicated in bold in the tables.

TABLE 9 Comparison between simulated annealing and the proposed DFWA.

Instance	SA				DFWA			
	Obj	CPU (s)	Gap_{CLRP} (%)	Gap_{CPLEX} (%)	Obj	CPU (s)	Gap_{CLRP} (%)	Gap_{CPLEX} (%)
Srivastava86-8 × 2	**377.09**	5.74	-	0.00	**377.09**	0.13	-	0.00
Perl83-12 × 2	**176.34**	1.33	−13.56	0.00	**176.34**	0.37	−13.56	0.00
Gaskell67-21 × 5	**320.17**	38.31	−24.65	0.00	**320.17**	9.50	−24.65	0.00
Gaskell67-22 × 5	**458.33**	1.51	−21.67	0.00	**458.33**	17.30	−21.67	0.00
Gaskell67-29 × 5	**386.26**	47.03	−24.57	0.00	**386.26**	30.58	−24.57	0.00
Gaskell67-32 × 5_1	**395.57**	23.06	−29.64	0.00	**395.57**	57.66	−29.64	0.00
Gaskell67-32 × 5_2	**374.70**	50.44	−25.70	0.00	**374.70**	32.31	−25.70	0.00
Min92-27 × 5	**2,110.03**	8.11	−31.09	0.00	**2,110.03**	30.62	−31.09	0.00
Daskin95-88 × 8	**285.91**	763.58	−19.64	−0.91	**285.91**	101.50	−19.64	−0.91
20-5-1	**43849.00**	4.18	−19.97	0.00	**43849.00**	31.59	−19.97	0.00
20-5-1b	**33564.00**	58.88	−14.17	0.00	**33564.00**	31.45	−14.17	0.00
20-5-2	**41125.00**	10.02	−15.91	0.00	**41125.00**	34.98	−15.91	0.00
20-5-2b	**32520.00**	39.51	−13.38	0.00	**32520.00**	30.65	−13.38	0.00
50-5-1	**64217.00**	342.11	−28.74	−0.22	**64217.00**	87.41	−28.74	−0.22
50-5-1b	**49114.00**	333.68	−22.34	0.00	**49114.00**	86.59	−22.34	0.00
50-5-2	**68121.00**	609.51	−22.85	0.00	**68121.00**	90.13	−22.85	0.00
50-5-2b	**57355.00**	582.93	−14.79	−0.80	**57355.00**	88.47	−14.79	−0.80
50-5-2bBIS	**41193.00**	866.25	−20.51	0.00	**41193.00**	91.92	−20.51	0.00
50-5-2BIS	60045.00	576.05	−28.56	−0.01	**60038.00**	86.55	−28.57	−0.02
50-5-3	**62581.00**	240.85	−27.40	0.00	**62581.00**	82.78	−27.40	0.00
50-5-3b	46756.00	725.89	−24.38	0.37	**46584.00**	93.68	−24.66	0.00
100-5-1	223694.00	1,451.64	−18.60	−1.93	**222634.00**	185.94	−18.99	−2.39
100-5-1b	**189208.00**	86.38	−11.43	−0.63	**189208.00**	97.42	−11.43	−0.63
100-5-2	166445.00	190.98	−14.06	−0.92	**166328.00**	69.97	−14.12	−0.99
100-5-2b	144777.00	1840.07	−7.84	−1.29	**144689.00**	205.83	−7.90	−1.35
100-5-3	162805.00	2,351.64	−18.63	−35.42	**162746.00**	291.57	−18.66	−35.44
100-5-3b	**134632.00**	472.71	−11.68	−2.70	**134632.00**	97.85	−11.68	−2.70
100-10-1	277415.00	1,424.05	−3.67	−10.08	**276859.00**	308.19	−3.86	−10.26
100-10-1b	250198.00	2,433.87	7.95	−8.50	**220134.00**	317.06	−5.02	−19.50
100-10-2	**213627.00**	2,652.10	−12.30	−27.68	**213627.00**	323.55	−12.30	−27.68
100-10-2b	190891.00	1895.08	−6.42	−26.70	**189818.00**	311.62	−6.95	−27.12
100-10-3	215526.00	2077.16	−14.09	−18.72	**214056.00**	319.96	−14.68	−19.27
100-10-3b	**188243.00**	1,461.79	−7.87	−4.54	**188243.00**	234.48	−7.87	−4.54
Average	89781.38	717.16	−17.57	−4.26	88730.89	117.56	−18.04	−4.66

The best results are indicated in bold in the tables.

Gaskell67-29×5 (≈1-s difference). While this time gap seems minimal in a single run, it accumulates significantly when the algorithm is executed repeatedly—for example, in large-scale benchmark testing (e.g., 10,000 runs for 33 OCLRP instances in Section 4.3) or real-world dynamic logistics scenarios (e.g., daily route optimization for a company with 100+ delivery points), the

TABLE 10 Current distribution routes and delivery costs.

Depots	Distribution routes
1	1–18–17-6-1
	1-9-13-8-1
	1–16–12-7-1
2	2–14–15–19–11–2
	2–20–10-5-2
Monthly delivery cost	93615 Yuan

TABLE 11 Capacities and rental costs of four potential depots.

Depot	Capacity (Tons)	Monthly rental cost (Yuan)
1	400	31000
2	300	35000
3	300	32000
4	300	32000

total time saved by combination 20 would be approximately 10,000×(2 + 1)/2 = 15,000 s (≈4.2 h) for the two instances alone. For larger instances (e.g., 100-5-1 with 100 customers), this cumulative advantage would be even more pronounced, as computation time scales with problem size.

In all these preliminary experiments, the best solution obtained by the proposed DFWA could not be further improved after 400 iterations where we had included a maximum number $G = 500$ of iterations and a termination condition for the search process if the best solution thus far found could not be improved after 100 iterations.

4.2 Comparative performance

As described, our DFWA was initially based on a greedy initialization and iterated swap, insertion, and inverse moves respectively employed to generate good initial solutions, strengthen local search, and enhance population diversity. To further investigate the effectiveness of the proposed search strategies, we compared it with four alternative DFWAs by solving eight representative OCLRP instances (small-scall: Gaskell67-21×5, Gaskell67-22×5, Gaskell67-29 × 5. medium-scale: 50-5-1, 50-5-3b. large-scale: 100-5-1, 100-10-1, 100-10-3b) systematically selected from 33 benchmark instances (Tables 7 and 8). These instances cover different scales and customer/depot ratios to ensure generalizability. The alternative algorithms were as follows

FWA1: initial solution is randomly generated rather than through greedy search. This serves to evaluate whether the proposed greedy approach enhances the overall performance of the DFWA.

FWA2: proposed DFWA except for the iterated swap move imitates the explosion operator, which is removed. This serves to assess the effectiveness of such a move.

FWA3: proposed DFWA except for the insertion move, which is dropped. This serves to assess the effectiveness of the insertion move.

FWA4: proposed DFWA without inverse move, which is therefore assessed.

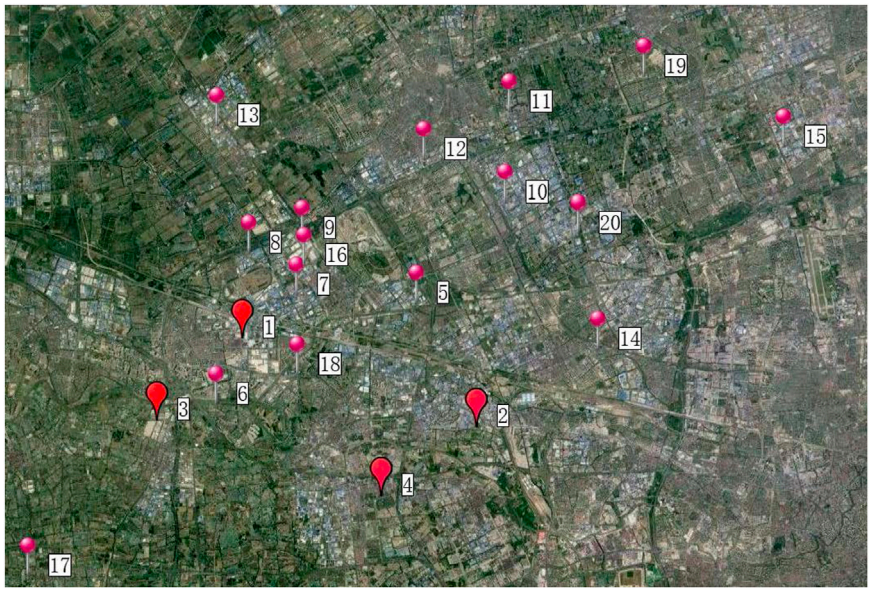

FIGURE 11
Map with four potential depots (1–4) and 16 delivery points (5–20) (supermarkets).

TABLE 12 Demand at each delivery point.

Index	Demand (Tons)	Index	Demand (Tons)
5	3.60	13	3.00
6	5.20	14	4.00
7	4.90	15	3.00
8	3.20	16	4.50
9	4.80	17	2.00
10	2.00	18	3.00
11	1.80	19	3.00
12	2.50	20	4.80

TABLE 13 Transportation distance in km between pairs of points.

Index	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	8.4	3.4	3.2	4.1	5.5	14.0	16.6	12.1	10.1	16.8	27.2	4.6	15.2	3.1	22.8	16.7
2	4.2	13.8	9.7	12.5	10.2	6.4	10.6	9.0	16.4	4.8	16.2	9.8	25.3	9.7	14.1	6.1
3	13.4	2.9	8.8	9.0	11.1	19.4	22.2	17.8	13.9	21.2	32.5	10.1	9.6	7.0	28.3	21.8
4	7.4	5.6	6.8	9.1	9.0	13.8	17.5	13.9	15.6	13.3	25.5	8.2	16.0	3.2	22.6	15.1
5	0.0	10.6	5.6	8.3	6.0	6.4	10.1	6.8	12.7	9.1	18.9	5.6	22.6	6.7	15.3	8.5
6		0.0	6.5	7.3	8.9	16.7	19.7	15.2	14.4	18.3	29.6	7.8	12.1	4.3	25.5	19.0
7			0.0	2.9	2.7	10.8	13.3	8.7	8.8	14.6	24.2	1.5	23.1	3.7	19.4	13.7
8				0.0	2.6	12.3	14.0	9.4	6.1	17.1	25.9	2.6	18.6	6.2	20.5	15.6
9					0.0	9.7	11.4	6.8	6.7	15.0	23.3	1.3	20.8	6.5	17.9	13.1
10						0.0	4.3	4.4	14.2	8.3	13.6	10.0	28.8	12.7	8.9	3.7
11							0.0	4.6	13.9	12.0	13.2	12.1	31.8	16.1	6.6	6.5
12								0.0	9.9	12.2	17.1	7.6	27.4	11.8	11.2	8.0
13									0.0	21.0	27.1	7.8	23.2	12.4	20.4	17.9
14										0.0	13.1	14.5	29.3	14.3	13.2	5.8
15											0.0	23.5	41.4	25.5	7.4	10.6
16												0.0	19.8	5.2	18.5	13.1
17													0.0	16.1	37.8	30.8
18														0.0	21.7	15.0
19															0.0	8.3
20																0.0

To ensure a fair comparison, all four alternative algorithms (FWA1–FWA4) adopt the same parameter combination as the original DFWA. No separate parameter tuning was performed for FWA1–FWA4; this eliminates the interference of parameter differences on performance and ensures that any observed gaps in results are attributed to the absence of specific components (greedy initialization, swap/insertion/inverse moves) rather than parameter adjustments.

Table 7 reports the comprehensive performance metrics of the five algorithms over 20 independent runs per instance, including mean objective value, standard deviation (SD) of objective value, average computation time (s), number of runs that found the best-known

TABLE 14 Results obtained by implementing OCLRP and CLRP.

OCLRP		CLRP	
Depots	Distribution route	Depots	Distribution route
3	3-7-16–12	3	3-8-9-13–3
	3-6-18–8		3-7-18–17–3
	3–17		3-6-3
2	2–14–20	2	2–15–19–11–10–2
	2-5-9-13		2–12–16-5-2
	2–10–11–19–15		2–14–20–2
Monthly delivery cost	82965 Yuan		93030 Yuan

TABLE 15 Monthly routing costs in Yuan at different fixed vehicle costs for the OCLRP.

Fixed vehicle cost	1,000	1,200	1,500	1,800	2,000	2,500	2,600	2,700
Delivery cost	82965	84165	85965	87765	88965	91965	92565	93165

solution, and p-values (vs. DFWA from Wilcoxon signed-rank test). Except for the instance Gaskell67-29 × 5, in which the p-value for the comparison between FWA3 and DFWA is larger than 0.05, all other p-values were less than 0.05, indicating statistically significant differences among FWA1–FWA4 and DFWA. Table 7 clearly shows that DFWA outperforms the other four algorithms in all instances in terms of both statistics. FWA2 provides the worst mean and SD; additionally, pairwise differences between the original DFWA and the four modifications seem significant according to the boxplots in Figure 10. Table 7 also suggests that: (1) the combination of the insertion and inversion move not only effectively improves DFWA population diversity, allowing it to escape from a current local optimum and move to non-previously visited spaces, but also enhances the exploration capabilities of the algorithm; (2) besides effectively strengthening the local search of the DFWA, the iterated swap move also enhances the exploitation capabilities of the DFWA; (3) the greedy approach tends to generate a relatively good-quality initial population, playing a significant role in improving the DFWA performance. To sum up, DFWA achieves better performance than FWA1–FWA4.

Moreover, Table 7 also suggests that DFWA seems more stable than the four variants, as also reported through the box plots in Figure 10 portraying the results of the five FWAs on eight instances.

4.3 Benchmarking our DFWA

To assess the performance of our DFWA in solving OCLRPs, we obtained 33 instances from the corresponding CLRP benchmark instances, solved them, and compared their results with those in Yu and Lin (2015) and their best-known solution.

Table 8 presents the benchmark results obtained by solving the OCLRP with CPLEX and the best-known CLRP solution. Each row provides the number n of customers, the number m of depots, the capacity Q of each depot, the best known objective value of the CLRP instance (from Ting and Chen, 2013), the best objective value obtained with CPLEX (from Yu and Lin, 2015), the CPU seconds spent by CPLEX, and the percent deviation from the best-known CLRP solution ($Gap_{CLRP} = \frac{\text{Obj. obtained using CPLEX}-\text{Obj. of CLRP}}{\text{Obj. of CLRP}} \times 100\%$). The last row provides average values. Values in bold indicate that the optimal solution is obtained within 14,400 s for each instance. Yu and Lin (2015) used CPLEX 12.0 and a simulated annealing heuristic implemented in Microsoft Visual C++ 2008 to solve the OCLRP instances on a laptop equipped with A10-7400P CPU at 3.40 GHz and 4 GB of RAM under Windows 10. These are key parameters in the SA (Yu and Lin, 2015): the iteration number equals $5000L$, where L is the coding length of the solution representation; the maximum allowable number of consecutive temperature reductions without improvement in the solution value is 100; the initial temperature is 30, and the final temperature is 0.1; the Boltzmann constant used in calculating the probability of accepting a worse solution is 1/9; the coefficient of the cooling schedule is 0.99.

Some might question whether the objective function values between the CLRP and OCLRP are comparable. After all, the OCLRP vehicle costs should differ from those in the CLRP, since the vehicles in the OCLRP are leased and are thus not owned by the company. However, the objective values in Table 8 are only used to explain that the logistics costs of the OCLRP are lower than those of the CLRP when vehicle costs are the same. In reality, should the logistics costs of the TPL company be lower than the logistics cost of the company's own distribution, the company would outsource its logistics activities to the TPL provider. Otherwise, the company would not outsource such activities. This provides a decision-support argument in relation to outsourcing logistics.

As Table 8 shows, CPLEX finds feasible solutions for all 33 instances, with 16 of them optimally solved within 14,400 s.

However, the computing time grows considerably with instance size. For 26 out of 33 test instances, CPLEX provided lower objective values than the corresponding CLRP instances. Such value difference could suggest potential benefits of outsourcing logistics (Yu and Lin, 2015). In particular, we can observe that the percentage deviation from the best-known objective value of the CLRP varies from −13.38% to −31.09%, indicating that the OCLRP can save important costs by outsourcing logistics (about 19.33% for the above-mentioned 26 instances).

Table 9 compares the results attained over the 33 OCLRP instances with our DFWA and SA—a powerful state-of-the-art heuristic algorithm for solving the OCLRP (Yu and Lin, 2015). As mentioned, the OCLRP has been significantly less investigated in the literature. Thus, we only compare the proposed algorithm with a heuristic in our study, as it is difficult to find other heuristic algorithms used to solve the OCLRP. Besides the objective value (Obj.) and CPU time spent by both algorithms, the table also provides the percentage deviation from the best-known solution of the CLRP (Gap_{CLRP}) and the percentage deviation from the best solution obtained with CPLEX ($Gap_{CPLEX} = \frac{\text{Obj. obtained using SA (or DFWA)} - \text{Obj. obtained using CPLEX}}{\text{Obj. obtained using CPLEX}} \times 100\%$). Values in bold indicate the best result between both algorithms. Note that the best objective values obtained using SA are calculated using the distribution routes found in Yu and Lin (2015).

From Table 9, we appreciate that SA and DFWA obtained the same results in 23 instances, whereas DFWA produced better solutions (and lower percentage deviations) in the remaining ten. Moreover, the average percentage deviations (Gap_{CPLEX} and Gap_{CLRP}) produced by DFWA over the 33 instances are lower than the corresponding values generated by SA. Thus, the proposed DFWA slightly outperforms SA in terms of solution quality. Concerning CPU times, we observe that, except for instances Gaskell67-22 × 5, Gaskell67-32 × 5_1, Min92-27 × 5, 20-5-1, 20-5-2, and 100-5-1b, the DFWA spent far less computational time in solving the instances. Note that different hardware, compilers, and programming languages could have a significant impact on the comparison. For our proposed DFWA, the slower MATLAB code did not seem to result in very large computational times.

Moreover, the results presented in Tables 7 and 8 enable a comparison between our DFWA and the CPLEX implementation. DFWA provides the same or better solutions. On the other hand, CPLEX spent slightly less computational time than our DFWA in most instances, with the number of customers smaller than 32. However, for instances with 50 or more customers, CPLEX spent much more computational time, and our DFWA provides better solutions within less than 330 s. The average computational time spent by CPLEX is about 8,256.21 s—much longer than DFWA's average 117.56 s. Thus, compared to CPLEX and SA, our DFWA seems to provide good solutions in a reasonable time for large-size OCLRP instances.

We also performed non-parametric Friedman and Wilcoxon signed-rank tests (at a significance level 0.05) to assess the statistical significance of the observed differences between DFWA and SA based on the Gap_{CPLEX} in Table 9. The average ranks for the Friedman test are 1.35 and 1.65, respectively, for the DFWA and SA, and the corresponding Friedman χ_F^2 statistic and p-value are 10 (critical value is 3.84) and 0.002. This suggests a rejection of the null hypothesis and indicates a significant difference between DFWA and SA results. Similarly, the Wilcoxon signed-ranks non-parametric test was applied to detect differences between DFWA and SA at instance level (pairwise comparison). The p-value was 0.005, suggesting the rejection of the null hypothesis and, again, statistical differences between our DFWA and the compared SA. Hence, the proposed algorithm seems truly competitive when solving OCLRPs.

4.4 A real case

Given the successful benchmarking of our DFWA for OCLRP, we tested it in a real case for a company that delivers to fruit and vegetable supermarkets in Shanghai, China. This company has been using five 12-ton capacity vehicles for daily distribution of merchandise from two depots to 16 supermarkets in the city. The transportation and monthly fixed costs (including only monthly depreciation) are, respectively, 5 Yuan per kilometer and 1,000 Yuan. Table 10 presents the current five distribution routes and monthly delivery costs (covering routing, fixed vehicle, depot rental costs). Depots are numbered 1 to 4 (the current two, plus an additional two), with their features in Table 11, and supermarkets are numbered 5 to 20.

In order to strengthen its competitive ability and save distribution costs, this company considers optimizing its logistics system. Besides the two current depots, two additional potential depots numbered 3 and 4 are contemplated. Figure 11 illustrates the locations of the depots and the supermarkets. Table 12 shows the daily demand of these. Table 13 lists the distances between each pair of points. The company needs to determine which depots from the four candidates should be open to provide service to the 16 supermarkets and whether it should outsource its delivery activities to a TPL company.

We solve the corresponding OCLRP and CLRP with our DFWA with results in Table 14. Compared with Table 10, we observe the differences between the current distribution routes and those obtained using the OCLRP and CLRP formulations. These two not only suggest the same open depots (2 and 3) but also produce the same number of distribution vehicles (six). However, as the last row in Table 14 shows, the monthly delivery cost using OCLRP is 82965 Yuan, which is less than 93030 Yuan produced with the CLRP formulation. The savings with respect to the current distribution routes are 11.38% (OCLRP) and 0.62% (CLRP). These results indicate that large distribution costs can be saved if the company outsources its delivery activities to a TPL company.

The same transportation and fixed vehicle costs are used for OCLRP and CLRP in the above discussion. We now analyze whether the company would still need to cooperate with a TPL provider should the unit transportation and fixed vehicle costs of the TPL company be larger than those used by the company-owned logistics. For different fixed vehicle costs, we obtained the same distribution routes shown in Table 14. Table 15 presents the monthly routing costs at different fixed vehicle costs for the OCLRP. As shown there, so long as the fixed vehicle costs are less than 2,700 Yuan, cooperation with a TPL provider seems a good option because the monthly delivery costs generated using OCLRP are less than the 93030 Yuan required if in-company logistics are used. However, when such costs are larger than 2,700 Yuan, it is better for the

company to deliver the merchandise to the supermarkets through its own logistics.

5 Discussion

We have studied the OCLRP, a variant of the classical CLRP, in which vehicles do not return to distribution centers or depots after having served the customers, and we proposed an effective DFWA to solve the OCLRP. This algorithm includes repeated swap moves to strengthen local search and improve solution quality. Moreover, to guarantee fireworks diversity, we implemented not only insertion and inversion moves in the mutation process but also a fitness-and-Hamming-distance-based selection strategy to select the next-generation fireworks. To verify the performance of the proposed DFWA, an extensive numerical experiment was carried out with 33 benchmark CLRP instances. Numerical results suggest that the proposed algorithm is quite competitive compared with the best-published results both in terms of solution quality and computational time. In addition, the differences between the CLRP and OCLRP solutions are further illustrated through a real case study providing improved distribution routes as well as information to support logistics outsourcing decisions.

Several potential extensions for this work include (1) developing other variants of the OCLRP, such as OCLRP with time windows, multi-objective OCLRP, and OCLRP with stochastic or fuzzy demands; (2) designing other meta-heuristic algorithms to solve the OCLRP and its variants; and (3) adapting the DFWA to solve other variants of FLPs and VRPs.

Data availability statement

Publicly available datasets were analyzed in this study. The original contributions presented in the study are included in the article/supplementary material; further inquiries can be directed to the corresponding author.

Author contributions

HZ: Supervision, Project administration, Conceptualization, Methodology, Investigation, Data curation, Writing – original draft, Writing – review and editing, Formal Analysis. XZ: Methodology, Writing – review and editing, Formal Analysis, Writing – original draft, Data curation. DR: Investigation, Writing – review and editing, Writing – original draft, Methodology.

Funding

The author(s) declare that financial support was received for the research and/or publication of this article. The research was supported by the Humanities and Social Sciences Foundation of the Ministry of Education of China (Grant No. 21YJC630087) and the High-end Foreign Experts Recruitment Plan of China (Grant No. G2023013029). The work of David Rios Insua is supported by the Spanish Ministry of Science and Research program PID2021-124662OB-I00 and the AXA-ICMAT Chair in Adversarial Risk Analysis.

Acknowledgements

The authors would like to thank all individuals and organizations who contributed directly or indirectly to this study. We also appreciate the insightful comments and suggestions provided by colleagues during the research process.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative AI statement

The author(s) declare that no Generative AI was used in the creation of this manuscript.

Any alternative text (alt text) provided alongside figures in this article has been generated by Frontiers with the support of artificial intelligence and reasonable efforts have been made to ensure accuracy, including review by the authors wherever possible. If you identify any issues, please contact us.

Publisher's note

References

Arifuddin, A., Utamima, A., Mahananto, F., Vinarti, R. A., and Fernanda, N. (2024). Optimizing the capacitated vehicle routing problem at pqr company: a genetic algorithm and grey wolf optimizer approach. *Procedia Comput. Sci.* 234, 420–427. doi:10.1016/j.procs.2024.03.023

Bootaki, B., and Zhang, G. (2024). A location-production-routing problem for distributed manufacturing platforms: a neural genetic algorithm solution methodology. *Int. J. Prod. Econ.* 275, 109325. doi:10.1016/j.ijpe.2024.109325

Chen, Y., Li, L., Zhao, X., Xiao, J., Wu, Q., and Tan, Y. (2019). Simplified hybrid fireworks algorithm. *Knowledge-Based Syst.* 173, 128–139. doi:10.1016/j.knosys.2019.02.029

Dai, Z., Aqlan, F., Gao, K., and Zhou, Y. (2019). A two-phase method for multi-echelon location-routing problems in supply chains. *Expert Syst. Appl.* 115, 618–634. doi:10.1016/j.eswa.2018.06.050

Ding, R., Hu, S., Xing, Z., and Yan, T. (2025). Multi-type radar deployment for uav swarms defense coverage using firework algorithm with determinantal point processes under complex terrain. *Appl. Soft Comput.* 170, 112681. doi:10.1016/j.asoc.2024.112681

Erbao, C., and Mingyong, L. (2009). A hybrid differential evolution algorithm to vehicle routing problem with fuzzy demands. *J. Comput. Appl. Math.* 231 (1), 302–310. doi:10.1016/j.cam.2009.02.015

Farham, M. S., Süral, H., and Iyigun, C. (2018). A column generation approach for the location-routing problem with time windows. *Comput. & Operations Res.* 90, 249–263. doi:10.1016/j.cor.2017.09.010

Ferreira, K. M., and Alves de Queiroz, T. (2022). A simulated annealing based heuristic for a location-routing problem with two-dimensional loading constraints. *Appl. Soft Comput.* 118, 108443. doi:10.1016/j.asoc.2022.108443

Garcia-Diaz, A., and Smith, J. M. (2024). "Facility location models," in *Facilities planning and design* (Cham: Springer Nature Switzerland), 149–188.

Garside, A. K., Ahmad, R., and Muhtazaruddin, M. N. B. (2024). A recent review of solution approaches for green vehicle routing problem and its variants. *Operations Res. Perspect.* 12, 100303. doi:10.1016/j.orp.2024.100303

Kechmane, L., Nsiri, B., and Baalal, A. (2018). A hybrid particle swarm optimization algorithm for the capacitated location routing problem. *Int. J. Intelligent Comput. Cybern.* 11 (1), 106–120. doi:10.1108/ijicc-03-2017-0023

Kyriakakis, N. A., Sevastopoulos, I., Marinaki, M., and Marinakis, Y. (2022). A hybrid tabu search – variable neighborhood descent algorithm for the cumulative capacitated vehicle routing problem with time windows in humanitarian applications. *Comput. & Industrial Eng.* 164, 107868. doi:10.1016/j.cie.2021.107868

Marques, G., Sadykov, R., Deschamps, J.-C., and Dupas, R. (2020). An improved branch-cut-and-price algorithm for the two-echelon capacitated vehicle routing problem. *Comput. & Operations Res.* 114, 104833. doi:10.1016/j.cor.2019.104833

Meng, X., and Tan, Y. (2024). Multi-guiding spark fireworks algorithm: solving multimodal functions by multiple guiding Sparks in fireworks algorithm. *Swarm Evol. Comput.* 85, 101458. doi:10.1016/j.swevo.2023.101458

Mohamed, I. B., Klibi, W., Sadykov, R., Sen, H., and Vanderbeck, F. (2022). The two-echelon stochastic multi-period capacitated location-routing problem. *Eur. J. Operational Res.* doi:10.1016/j.ejor.2022.07.022

Niu, Y. Y., Shao, J., Xiao, J. H., Song, W., and Cao, Z. G. (2022). Multi-objective evolutionary algorithm based on rbf network for solving the stochastic vehicle routing problem. *Inf. Sci.* 609, 387–410. doi:10.1016/j.ins.2022.07.087

Nucamendi-Guillén, S., Gómez Padilla, A., Olivares-Benitez, E., and Moreno-Vega, J. M. (2021). The multi-depot open location routing problem with a heterogeneous fixed fleet. *Expert Syst. Appl.*, 165, 113846, doi:10.1016/j.eswa.2020.113846

Prins, C., Prodhon, C., and Calvo, R. W. (2006). Solving the capacitated location-routing problem by a grasp complemented by a learning process and a path relinking. 4(3), 221–238. doi:10.1007/s10288-006-0001-9

Shen, X., Xu, D., Song, L., and Zhang, Y. (2023). Heterogeneous multi-project multi-task allocation in mobile crowdsensing using an ensemble fireworks algorithm. *Appl. Soft Comput.* 145, 110571. doi:10.1016/j.asoc.2023.110571

Shi, Y., Zhou, Y., Boudouh, T., and Grunder, O. (2022). A memetic algorithm for a relocation-routing problem in green production of gas considering uncertainties. *Swarm Evol. Comput.* 74, 101129. doi:10.1016/j.swevo.2022.101129

Tan, Y. (2015). *Fireworks algorithm: a novel swarm intelligence optimization method*. Incorporated: Springer Publishing Company.

Tan, Y., and Zhu, Y. (2010). "Fireworks algorithm for optimization," in *Advances in swarm intelligence. ICSI 2010. Lecture notes in computer science*. Editors Y. Tan, Y. Shi, and K. C. Tan (Berlin, Heidelberg: Springer), 6145, 355–364. doi:10.1007/978-3-642-13495-1_44*Lect. Notes Comput. Sci.*

Tan, D., Liu, X., Zhou, R., Fu, X., and Li, Z. (2025). A novel multi-objective artificial bee colony algorithm for solving the two-echelon load-dependent location-routing problem with pick-up and delivery. *Eng. Appl. Artif. Intell.* 139, 109636. doi:10.1016/j.engappai.2024.109636

Ting, C.-J., and Chen, C.-H. (2013). A multiple ant colony optimization algorithm for the capacitated location routing problem. *Int. J. Prod. Econ.* 141 (1), 34–44. doi:10.1016/j.ijpe.2012.06.011

Toro, E., Franco, J. F., Echeverri, M. G., Guimarães, F. G., and Gallego Rendón, R. A. (2017). Green open location-routing problem considering economic and environmental costs. *Int. J. Industrial Eng. Comput.* 8 (2), 203–216. doi:10.5267/j.ijiec.2016.10.001

Wang, Y., Sun, Y., Guan, X., Fan, J., Xu, M., and Wang, H. (2021). Two-echelon multi-period location routing problem with shared transportation resource. *Knowledge-Based Syst.* 226, 107168. doi:10.1016/j.knosys.2021.107168

Yu, V. F., and Lin, S.-Y. (2015). A simulated annealing heuristic for the open location-routing problem. *Comput. & Operations Res.* 62, 184–196. doi:10.1016/j.cor.2014.10.009

Yu, V. F., and Lin, S. Y. (2016). Solving the location-routing problem with simultaneous pickup and delivery by simulated annealing. *Int. J. Prod. Res.* 54 (2), 526–549. doi:10.1080/00207543.2015.1085655

Zhang, H. Z., Zhang, Q., Ma, L., Zhang, Z., and Liu, Y. (2019). A hybrid ant colony optimization algorithm for a multi-objective vehicle routing problem with flexible time windows. *Inf. Sci.* 490, 166–190. doi:10.1016/j.ins.2019.03.070

Zhang, H. Z., Zhang, K., Zhou, Y. Y., Ma, L., and Zhang, Z. Y. (2022a). An immune algorithm for solving the optimization problem of locating the battery swapping stations. *Knowledge-Based Syst.* 248, 108883–108883. doi:10.1016/j.knosys.2022.108883

Zhang, S., Zhang, J., Zhao, Z., and Xin, C. (2022b). Robust optimization of municipal solid waste collection and transportation with uncertain waste output: a case study. *J. Syst. Sci. Syst. Eng.* 31, 204–225. doi:10.1007/s11518-021-5510-8