A Hybrid Local Search Algorithm for the Cumulative Capacitated Vehicle Routing Problem with Prioritized Customers

Chunyang Zhang, Defeng Zhu, and Yan-e Hou

***Abstract*—This paper investigates a cumulative capacitated vehicle routing problem with prioritized customers (CCVRP-Pr), which aims to provide preferential service based on the severity of disaster areas while also reducing the losses of affected people. First, a mathematical model is established with the objective of minimizing the time to reach disaster areas and the penalty time for priority violations. Then, a hybrid local search (HLS) algorithm is proposed to solve the problem. The algorithm employs a greedy insertion method to rapidly construct an initial solution, which also allows for the generation of infeasible initial solutions violating the constraint of a specified number of vehicles. The initial solutions are iterated improved by the procedure of variable neighborhood descent (VND). Meanwhile, that infeasible solutions are changed to be feasible by the repair method. To prevent the algorithm from trapping into a local optimum early, two perturbation operators based on ruin-and-recreate and ejection chain are randomly selected to perturb the local best solution. For those inferior solutions found by local search, a rule based on simulated annealing acceptance is used to ensure the solutions diversity. Finally, several experiments were conducted on 82 benchmark instances, and the results demonstrate the effectiveness of the proposed algorithm.**

***Index Terms*—vehicle routing problem, cumulative objective, prioritized customers, hybrid local search, variable neighborhood descent**

I. Introduction

IN the context of disaster relief, the problem of planning the optimal route for rescue vehicles is a typical application of the vehicle routing problem (VRP), which is a classical NP-hard combinatorial optimization problem. The traditional approach to VRP is to minimize operating costs, but the main goal in this context is to minimize casualties and victims' suffering. It is essential to consider the speed and fairness of material distribution from the perspective of the affected areas in order to ensure that relief supplies reach each disaster area promptly. In order to achieve this goal, Ngueveu et al. [1] first introduced the cumulative capacity vehicle routing problem (CCVRP), which aims to minimize the total arrival time of all customers under vehicle capacity constraints, rather than the traditional path length. Its solution has been demonstrated to be more suitable for humanitarian supply chains due to the fact that route distribution costs are not a significant issue. Since the establishment of CCVRP, with the continuous improvement of the quality of the solving algorithm, it has attracted great attention from researchers. A substantial corpus of literature exists on CCVRP and its variants, and the recent literature review can be found in [2].

Due to the effect of geographical location, the degree of disaster in each region is different. If each disaster region is treated fairly, it can easily lead to the hard-hit areas not receiving timely assistance. Generally speaking, the areas closest to the disaster are more severely affected. Following an equal relief principle might mean that disaster areas don't get help in time, causing more deaths and suffering. Also, the company's goals of customer satisfaction and profits are important. When resources are limited, the company must make trade-offs to ensure the satisfaction of important customers. Thus, it is important to consider the priority of service regions in humanitarian supply chains.

In view of the foregoing, this paper studies a new variant of CCVRP, named CCVRP-Pr. For the CCVRP-Pr addressed in this paper, there are a few high-priority customers, and the majority of customers are low-priority. When a low-priority customer is served before a high-priority one, dissatisfaction penalty cost is incurred. We define a lateness penalty function to calculate the penalty time for scenarios in which customers jump queues. Then, we develop a hybrid local search (HLS) algorithm to minimize the total arrival time of all customers and the penalty time.

The main contributions of this paper are as follows:

- A problem model is established, whose goal is to minimize the arrival time at disaster sites and the penalty time for violation of priority.
- The different optimization operators are designed in local search by the concept of variable neighborhood descent (VND) to find better solutions, and a new initial solution generation algorithm is proposed to generate both feasible solutions and infeasible solutions.
- Two different perturbation operators, ruin-and-recreate and ejection chain, are designed to escape local optimum and explore larger solution spaces.
- A simulated annealing acceptance criterion is applied to ensure solution diversity, preventing premature convergence of the algorithm and enhancing solution quality.

The rest of this paper is organized as follows. The literature review of problem and solution approaches is given in Section II. In Section III, the model of CCVRP-Pr is defined and its mathematical model is described. To solve this

Manuscript received June 09, 2025; revised Aug 15, 2025. This work was supported by National Natural Science Foundation of China (42471459) and Key Scientific and Technological Project of Henan Province (242102210080).

Chunyang Zhang is an undergraduate of School of Computer and Information Engineering, Henan University, Henan, China (e-mail: cyzh@vip.henu.edu.cn).

Defeng Zhu is an undergraduate of School of Computer and Information Engineering, Henan University, Henan, China (e-mail: zhudefeng@henu.edu.cn).

Yan-e Hou is a professor of Henan Key Laboratory of Big Data Analysis and Processing, Henan University, Henan, China (Corresponding author, e-mail: houyane@henu.edu.cn)

problem, an HLS algorithm is developed in Section IV. The computational results, using a set of benchmark instances, are presented in Section V. Finally, we summarize the research conclusions and suggest future research directions in Section VI.

II. Literature Review

The cumulative vehicle routing problem is classified into the cumulative vehicle routing problem (Cum-VRP) and CCVRP [2] according to the cumulative objective. For Cum-VRP, its objective prefers to travel the most distant arcs as the vehicle gets lighter, which is related to the distance and the load of the vehicle. However, for CCVRP, the arrival of customers is the cumulative component, and the total arrive time is the optimization objective. CCVRP has received more attention due to several application in emergency relief operations and customer-centered logistics operations.

The CCVRP arises from the Capacitated Vehicle Routing Problem (CVRP), and most of the CCVRP literature focuses on general CCVRP [1], [3], [4], [5], [6], [7]. As research deepens, some CCVRP variants considering multiple depots [8], [9], [10] and time windows [11], [12], are also proposed to address more complex real-world requirements. This paper primarily focuses on general CCVRP, thus the related work about which will be described in the following.

Ngueveu et al. [1] was the first to study CCVRP and developed a memetic algorithm combing crossover operations and local search procedures to generate solutions, using the vrpnc instances with 50-199 nodes proposed in [13]. Lysgaard et al. [3] proposed a branch-and-cut-and-price algorithm (BCP) to solve CCVRP instances, and it was the first exact algorithm for CCVRP. The BCP algorithm is effective for small-scale problems but cannot solve larger or more complex problem types. Nucamendi-Guillén et al. [14] proposed two manageable integer formulations which can solve instances with up to 44 nodes, alongside two iterative greedy algorithms for larger instances. The experimental results indicate that these integer formulations can achieve optimal solutions within reasonable computation time, and the two metaheuristic algorithms can achieve outcomes comparable to the best-known solutions. Exact methods can obtain optimization solutions on small-scale problems, but they are impractical for complex scenarios due to the limitations of the solvers in handling medium to large-scale instances.

The heuristic and metaheuristic methods are the main methods to solve the CCVRP and its variants. Mattos Ribeiro et al. [4] employed the Adaptive Large Neighborhood Search algorithm (ALNS), which utilizes adaptive probability models to apply destruction and repair operators to generate new solutions. These solutions were then compared with those generated by [1]. Ke and Feng [5] proposed a two-phase metaheuristic algorithm whose algorithm integrates perturbation and local search operators. In each iteration, two interdependent phases use different perturbation and local search operators to improve the solutions. Sze et al. [6] developed a two-phase adaptive large neighborhood search algorithm, using large neighborhood search as a diversification strategy for solutions. The comparative analysis shows that the optimal solutions of classical CVRP can lead to suboptimal solutions of CCVRP, which is consistent with the findings of [15]. Ke et al. [7] proposed the Brainstorm Optimization Algorithm (BSO), designing new convergence and divergence operations. The convergence operation selects and perturbs the optimal solution so far, decomposing the CCVRP into multiple sub-problems to be solved, and the divergence operation selects one of three operators to generate new solutions to the sub-problems, and then assembles these solutions into a solution to the original problem. This algorithm has effectively solved large instances with up to 1200 nodes. Liu and Jiang [11] developed an effective algorithm based on the large neighborhood search and genetic algorithms to address the CCVRP with time window constraints. This algorithm involved a constraint relaxation scheme to expand the search space, to enable iterative exploration of feasible and infeasible neighboring solutions. Smiti et al. [16] proposed the Skewed General Variable Neighborhood Search algorithm, which yielded superior solutions compared to memetic algorithms [1] and ALNS [4] in the set of instances of [13] and [17]. Recently, Kyriakakis et al. [18] utilized two algorithms, named Ant Colony System Variable Neighborhood Descent algorithm (ACO-VND) and Max-Min Ant System Variable Neighborhood Descent algorithm (MMAS-VND), and it reached best-known solutions in 92 out of the 112 instances tested, offering superior results and reduced computation time. Later, Kyriakakis et al. [12] also presented a Hybrid Tabu Search - Variable Neighborhood Descent algorithm. This algorithm can solve both general CCVRP and its varient with time windows, and it can reach 84 of them on the 92 CCVRP instances tested.

These successful experiences in solving general CCVRP have promote the related research in emergency vehicle routing problems. However, considering the difference in the degree of damage in different regions in actual disasters scenario, the optimization objective of the CCVRP need include total arrival times and the degree of satisfaction of all customers. Nucamendi-Guillén et al. [19] proposed the bi-objective CCVRP with customer prioritization, consisting of the weighted sum of arrival time and penalty time due to prioritization, and developed two memetic algorithms to solve it. The experimental results show that their methods are effective for small instances, but the solving quality of methods are not sufficient for large instances.

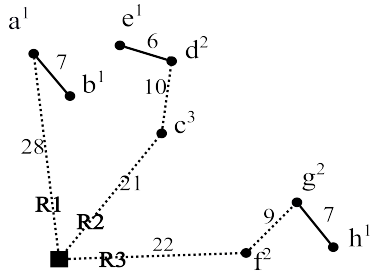
In this paper, we study the post-disaster vehicle routing problem with customer prioritization that is a CCVRP variant, which is still a NP-hard problem. The effectiveness of heuristic and metaheuristic encourage us to develop a more efficient algorithm to solve the addressed problem. Local search methods, a kind of simple and effective heuristic, can find better solutions in optimization problems [9], [10]. Therefore, we try to propose a hybrid local search algorithm integrated local search with variable neighborhood descent procedure to solve this problem efficiently.

III. Problem Formulation

This section will give the description and definition of the CCVRP-Pr problem, along with the method for calculating priorities and the corresponding mathematical formulation.

A. Problem Description

The CCVRP-Pr is defined as an undirected graph $G = (V, E)$. $V = \{0, 1, 2, \ldots, n, n+1\}$ represents the node set,



$$\sum I_{a,d} = \left|t_a^1 - t_d^2\right| = 31 - 28 = 3$$
$$\sum I_{a,g} = \left|t_a^1 - t_g^2\right| = 31 - 28 = 3$$
$$\sum I = I_{a,g} + I_{a,d} = 6$$

Fig. 1. The calculation of penalty time

both node 0 and $n+1$ are the same depots. The customer set is denoted as $V' = \{1, 2, \ldots, n\}$, and E is the set of all edges. Each edge $(i, j) \in E$ has a travel time w_{ij} and each node $j \in V'$ has a demand q_j. The solution consists of R homogeneous vehicles ($R > 0$), every of which has the same maximum capacity Q. The variable t_i^k donates as the time when vehicle k arrives at node i from the depot. At the same time, an $n \times n$ priority matrix with binary variables is defined to consider priority, in which $p_{ij} = 1$ means customer i is served before customer j, and $p_{ij} = 0$ means customer i can be served after customer j.

If a lower-priority customer is served before a higher-priority customer in the same route or different routes, dissatisfaction may emerge. This dissatisfaction arose based on the order of service, so the difference in their arrival time can measure that dissatisfaction. Accordingly, the penalty time is the difference in their arrival time, written as $I_{ij} = |t_i - t_j|$. The total penalty time of the system can be shown in formula (1)($j \neq i$):

$$\sum_{i \in\ V'} \sum_{\substack{j \in V' \\ j \neq i}} I_{ij} = p_{ij} \left| \sum_{k \in R} t_i^k - \sum_{k \in R} t_j^k \right|, \forall i \in V', j \in V' \quad (1)$$

Since the minimum of I_{ij} is 0 and the maximum represents the inversion order of customer service, also serves lower-priority customers first and higher-priority customers last, we can optimize each route's customers to get a maximum lateness time I_{max}. This means the system's total penalty time ranges from $[0, I_{max}]$.

Figure 1 graphically illustrates an instance with 9 nodes which has a solution with 3 routes. For each number symbol at each point on the graph, the form a^1 indicates the customer a has the priority level of 1 (the higher the number, the higher the priority). The depot is marked specially. The weight value on each edge represents the path length. According to the above calculation method, customers d and g show dissatisfaction with customer a. Thus, the corresponding penalty times are the difference in their service time, resulting in a total penalty time of 6.

B. Mathematical Formulation

The objective of CCVRP-Pr is to minimize the total arrival time of all customers plus penalty time under the constraints of the maximum load capacity Q. The mathematical formulation of the CCVRP-Pr can be stated in the following.

Minimize:

$$F = \sum_{k=1}^{R} \sum_{i \in V'} t_i^k + \sum_{i \in\ V'} \sum_{\substack{j \in V' \\ j \neq i}} I_{ij} \quad (2)$$

subject to:

$$\sum_{j \in V} x_{ji}^k = \sum_{j \in V} x_{ij}^k,\ \forall i \in V',\ \forall k \in R \quad (3)$$

$$\sum_{k=1}^{R} \sum_{j \in V} x_{ij}^k = 1,\ \forall i \in V' \quad (4)$$

$$\sum_{j \in V} x_{0j}^k = 1,\ \forall k \in R \quad (5)$$

$$\sum_{j \in V} x_{j,n+1}^k = 1,\ \forall k \in R \quad (6)$$

$$\sum_{i \in V'} \sum_{j \in V} x_{ij}^k q_i \leq Q,\ \forall k \in R \quad (7)$$

$$t_i^k + w_{ij} - \left(1 - x_{ij}^k\right) G \leq t_j^k, \forall i \in V, \forall j \in V', \forall k \in R \quad (8)$$

$$t_i^k \geq 0,\ \forall i \in V, \forall k \in R \quad (9)$$

$$x_{ij}^k \in \{0, 1\},\ \forall i \in V, \forall j \in V, i \neq j, \forall k \in R \quad (10)$$

The objective function of CCVRP-Pr is shown in equation (2). Constraint (3) ensures that the vehicle serves customer i and must leave it. Constraint 4 ensures that each customer is served by exactly one vehicle. Constraints (5) and (6) indicate that per vehicle's route starts and ends at the depot. Constraint (7) limits every vehicle's total load to its maximum capacity. Constraint (8) prevents the creation of sub-tours using a large constant G. Constraint (9) require that t_i^k must be non-negative. Constraint (10) gives the value of the decision variable x_{ij}^k. If the vehicle k traverses edge (i, j), it is equal to 1; otherwise, it's equal to 0.

IV. Proposed Method

The proposed algorithm combines the iterated local search and variable neighborhood descent [20], [21], which are effective heuristic that has been widely applied in vehicle routing problem. The hybridization of components of two algorithms can take advantage of their performances to improve the quality of the proposed algorithm. Furthermore, we also introduce the effective perturbation methods and nondeterministic acceptance rule to keep the balance between exploration and exploitation of the proposed algorithm.

A. Overall Framework of HLS

The proposed algorithm, namely HLS, is developed in the framework of iterated local search, which has VND local search procedure, improved perturbation operators, and nondeterministic acceptance criterion. The overall structure of the HLS algorithm is shown in Algorithm 1.

Step (1) reads the node information from instances. Steps (2) to (4) generate the initial solution using the algorithm 2. The initial solution may be infeasible due to capacity constraints, which is subsequently corrected by the repair

Algorithm 1: HLS

Input : $file$: vrp file; n: number of customers; k: number of vehicles; T: initial temperture; a: colding rate; $epoch$: max termination nummber ; p: perturbation factor
Output: sol_b: best solution

```
node = ReadVRP(file)
sol_b = ø
sol_0 = InitialSolution(node,k,n)
sol_b = RepairLocalSearch(sol_0) // attempt to repair infeasible solution
while epoch > 0 do
    p = Rand(0,1)
    sol_v = PMSearch(sol_b,p) // execute perturbation operator
    sol_v = LS − VND(sol_v)
    if sol_v < sol_b then
        sol_b = sol_v
        epoch = 20
    else
        r = Rand(0,1)
        if r < T then
            sol_b = sol_v
            epoch = 20
        end
    end
    epoch = epoch − 1
    T = T × a
end
return sol_b
```

Algorithm 2: InitialSolution

Input : $node$: customers; n: number of customers; R: number of vehicles
Output: sol_0: initial solution

```
Sort(node)// Descending sort by priority and demand
sol_0 = ø with R routes
for i : node do
    Insert i into the head route of R routes in sol_0
    Ascending sort the sequence of R routes in sol_0 by its load
end
return sol_0
```

method defined in Section IV-E. Steps (5) to (21) are the main produce of the algorithm which is divided into three stages. Step (7) is the perturbation stage, during which a perturbation operator is randomly selected between two perturbation operators with a given intensity p to perturb the solution. Step (8) executes a VND search. The final stage applies the simulated annealing criterion to accept new solutions. Besides, the HLS algorithm records the best result per iteration.

Given that the HLS algorithm can accept infeasible solutions, the accept rule is the combination of the feasible and infeasible objective function value, which is defined as follows:

$$f(x) = \begin{cases} F(s), & s\ is\ feasible \\ Q(s), & s\ is\ infeasible \end{cases} \tag{11}$$

where $F(s)$ is the objective function to be optimized and $Q(s)$ is the objective function when constraints are violated. The algorithm accepts solutions with a lower objective function value and always prefers to accept feasible solutions. The exact acceptance rule is detailed in Section IV-E.

B. Initial Solution

The initial solution is generated by a greedy insertion approach. Initially, R empty paths are created, then customers are sorted by priority and demand and inserted into the path with the largest remaining capacity to balance customer priority and demand. The initial solution generated by the algorithm may be infeasible and will be repaired during the later procedure. Algorithm 2 shows how the initial solution is obtained.

C. Neighborhood Structures

The neighborhood structure generates neighboring solutions through small perturbations of the current feasible solution to construct a set of neighboring solutions. The procedure of selecting a neighboring solution to update the current solution is called a neighborhood move. The neighborhood is designed to consist of several solutions closest to the current solution in the HLS algorithm and the solution is updated by moving nodes among inter-routes or intra-routes. The HLS algorithm utilizes six neighborhood search operators, some of which can repair infeasible solutions. The operations are detailed in the following.

(1) One-Point Move: This operator moves a node to the before or after of another node which is also called a relocation move. It can happen on the same or different routes. The details are shown in Figure 2. Because this operator can change the route's capacity when executed between different routes it can repair infeasible solutions.

(2) Two-Point Swap: This operator swaps two nodes of the same or different routes. This operator can change route capacities when performed between different routes so it can repair solutions like one-point move. Figure 3 illustrates the procedure in detail.

(3) 2-Opt: This is a classical 2-opt from the Traveling Salesman Problem which deletes two non-continuous edges and reconnects them back-to-front. This operator doesn't change the route's capacity so it is unable to repair infeasible solutions. Besides, we make it only occur on routes with at least three customers. Figure 4 demonstrates this procedure by deleting edges $e1$ and $e2$, reversing the connection from $e1$ to $e2$, and then adding two new edges $e3$ and $e4$.

(4) Arc-Node Swap: This operator selects nodes and arcs (two consecutive customers) on the same or different routes and swaps them, also called the three-point move. Figure 5 shows the procedure. This operator can occur on the same or different routes but can't repair infeasible solutions.

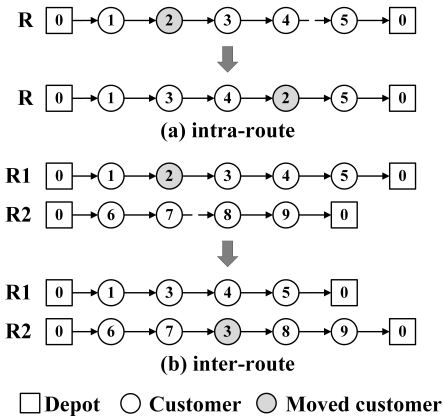


Fig. 2. Example of the One-Point Move

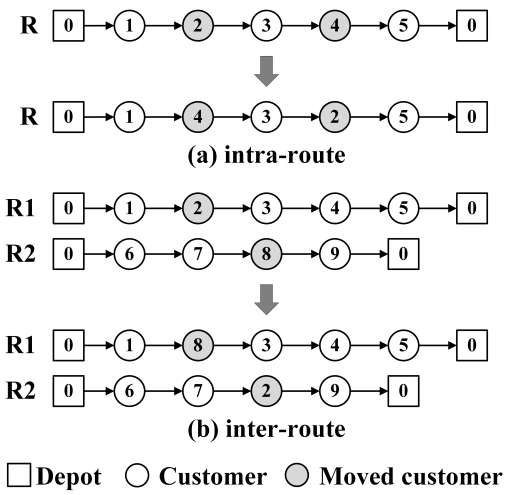


Fig. 3. Example of the Two-Point Swap

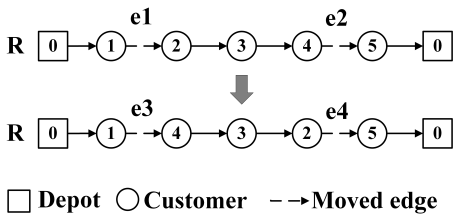


Fig. 4. Example of the 2-opt

(5) Or-Opt Move: It's an extended one-point move operator that moves a series of consecutive nodes (2 to 4) to the before or after of another node. Figure 6 illustrates the OrOpt2 process that moves two nodes. This operator can apply to the same or different routes and repair infeasible solutions when the route's capacities are changed.
(6) Arc Swap: This operator swaps the positions of two pairs of consecutive nodes. It can be applied on the same or different routes, as shown in Figure 7. This operator can repair infeasible solutions.

D. Variable Neighborhood Descent Search

The HLS algorithm uses VND as the local search process due to its ability to explore the large space. The detailed process is shown in Algorithm 3. The algorithm is initialized

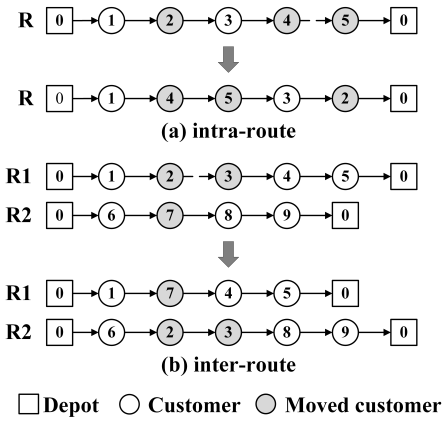


Fig. 5. Example of the arc-node swap

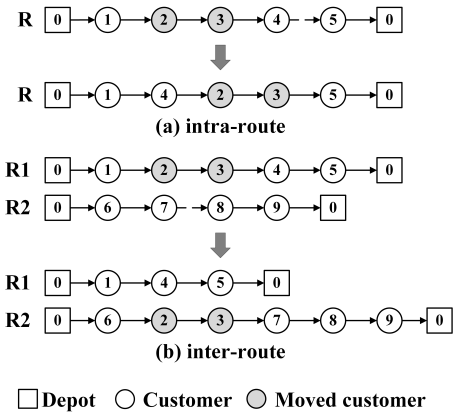


Fig. 6. Example of or-opt

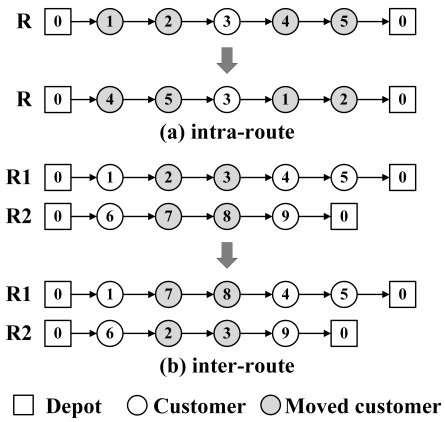


Fig. 7. Example of the arc swap

in Steps (1) and (2). The VND process is described in Steps (4) to (19), where we do a 2-opt after seven operators update the current solution. Last, a maximum number of executions is used to limit the execution time per operator.

E. Repair Method

In this section, we will describe the process of how each operator repairs solutions. The generation algorithm of the initial solution strictly limits the number of vehicles and this may result in some vehicles being overloaded. The repair process adjusts each route's capacity to satisfy capacity constraints. Through analyzing the instances, we reveal that

Algorithm 3: LS-VND

Input : sol_0: input solution; n: number of customers; k: number of vehicles
Output: sol_v: output solution

sol_v = sol_0
max_iter = $10 \times n$
foreach $ng \in N$ **do** // N is the set of seven operators
  $i = 0$
  $flag = true$
  while $flag = true$ **do**
    sol_v = Improve(sol_v,ng_i)
    if sol_v *improved* **then**
      sol_v = 2-Opt(sol_v)
    else
      $flag = false$
    end
    $i = i + 1$
    if $i > max\_iter$ **then**
      $flag = false$
    end
  end
end
return sol_v

Algorithm 4: Repair

Input : sol_i: infeasible solution; Q: max vehicle capacity
Output: sol_f: feasible solution

sol_f = sol_i
foreach $Operator\ in\ LocalSearch$ **do**
  $r1, r2$ = GetRouter(sol_f)
  $r1', r2'$ = Operator($r1$, $r2$)
  q_{r1} = GetRouterLoad($r1$)
  q_{r2} = GetRouterLoad($r2$)
  if $q_{r1} > Q \ || \ q_{r2} > Q$ **then** // infeasible
    ${q'}_{r1}$ = GetRouterLoad($r1'$)
    ${q'}_{r2}$ = GetRouterLoad($r2'$)
    if ${q_{r1}}^2 + {q_{r2}}^2 > {{q'}_{r1}}^2 + {{q'}_{r2}}^2$ **then**
      ApplySolUpdate(sol_f, $r1'$,$r2'$)
    end
  else // feasible
    if $f(sol_i, r1', r2') < f(sol_i, r1, r2)$ **then**
      ApplySolUpdate(sol_f, $r1'$,$r2'$)
    end
  end
end
return sol_f

its total demand *C* is a fixed value for one instance. Thus, repairing infeasible solutions can be abstracted into the following optimization problem:

$$min\ Q(s) = \sum_{k=1}^{R} r_k^2 \tag{12}$$

subject to:

$$\sum_{k=1}^{R} r_k = C \tag{13}$$

$$q_k \geq 0, \forall k \in \{1, \ldots, R\} \tag{14}$$

where r_k is the sum of demand in route k. And we can find that the minimum value of $Q(s)$ is acquired when $r_1 = r_2 = r_3 = ... = r_R = \lfloor C/R \rfloor$. Due to the capacity of a route is a discrete variable, the minimum value of $Q(s)$ is acquired when each r is close to the value of $\lfloor C/R \rfloor$. Further research shows that each operator involves changes in only two routes at most. The total load of the two routes remains unchanged before and after the change. Therefore, when comparing the $Q(s)$ values of infeasible solutions, it is only necessary to calculate the values of the two routes.

Algorithm 4 demonstrates the overall process of repairing infeasible solutions for each operator. The repair or optimization process of the operator is carried out in Steps (7) to (18) after moving nodes. If there is an overloaded route, the operator executes the criteria in Steps (8) to (12) to repair solution. It will accept the smaller value of $Q(s)$ and the $Q(s)$ is simplified calculated and compared in Step (10).

F. Perturbation Method

The HLS algorithm can get trapped in local optimum like other local search-based algorithms. The perturbation operators allow the HLS algorithm to restart the search from different solution spaces to find global optimum, so an effective heuristic method gets the best solution by the combination of local search and perturbations. We design two different kinds of perturbations: ruin-recreate and ejection chain to explore a larger neighborhood solution space.

The ruin-recreate perturbation originated from [22]. The pseudocode of the ruin-recreate is provided in Algorithm 5. The variables and best solution are initialized in Steps (1) to (3). The random customers are selected to be removed and then we obtain both the solution with customers removed and removed customers in Steps (4) and (5). The main loop is from Steps (6) to (16) and it consists of two phases: the repair phase and the update one. In the repair phase, the removed customers will be shuffled randomly, after which they will be reinserted to rebuild the solution. If the new solution rebuilt is better than the current solution, the algorithm will terminate early with the new solution returned in the update phase. It also records the best solution found in Steps (13) to (15). The above procedures are repeated until the termination condition is met.

The ejection chain perturbation is a variable-depth search including k insertion processes where k is the depth of the ejection chain. A node from path 1 is moved to path 2, a point from path 2 is moved to path 3, and so on. Finally, a node from path k is moved to path 1. The pseudocode of ejection chain perturbation is shown in Algorithm 6. The variables and sequence of *k* paths are initialized in Step (1) to (3). In step (4), the head route of the solution is copied to the tail, $routes = \{1, 2, 3, ..., k, 1\}$. In steps (5) to (9), k insertion procedures are executed to exchange a randomly chosen node between adjacent paths in routes.

G. Acceptance Criterion

It's important to maintain the diversity of solutions to prevent the HLS algorithm from getting trapped in a local

Algorithm 5: RuinRecreate

Input : sol: solution; p: the destruction factor; $epoch$: the max number of iteration
Output: sol_p: the best solution

sol_p = ∅
sol_b = ∅
t = 0
$nodes$ = GetRemovedNodes(sol, p)
sol_0 = RuinSolution(sol, $nodes$)
while $t < epoch$ **do**
  RandShuffle($nodes$)
  sol_n=RepairSolutionn(sol_0, $nodes$)
  if $sol_n < sol$ **then**
    $sol_p = sol_n$
    return sol_p
  end
  if $sol_n < sol_b$ **then**
    $sol_b = sol_n$
  end
  $t = t + 1$
end
$sol_p = sol_n$
return sol_p

Algorithm 6: EjectionChain

Input : sol: solution; k: the number of perturbation routes; num: the epoch of max perturbation
Output: sol_p: the perturbed solution

i = 0
for $i > num$ **do**
  $routes$ = RandSelect(sol, k)
  $routes$ = $routes \cup routes[0]$
  for $r1$, $r2$ *:* $routes$ **do**
    $node1$ = RandSelect($r1$)
    $node2$ = RandSelect($r2$)
    Swap($node1$, $node2$)
  end
  i = i + 1
end
$sol_p = sol$
return sol_p

optimum. When a better neighborhood solution is acquired from the local search it will accept this solution. This acceptance rule is called the improvement acceptance. However, if the algorithm always accepts the better solution, it may get trapped in the local optimum due to diversity lacking. We use a probable acceptance rule based on simulated annealing to accept a worse solution. It is defined in the following:

$$S = \begin{cases} S_n, & f(S_n) < f(S_c) \\ S_n, & f(S_n) > f(S_c),\ \Delta t > p,\ p \in (0,1) \\ S_c, & otherwise \end{cases} \quad (15)$$

where S_c and S_n are the current and new neighborhood solutions, $f(S_n)$ is the objective value of S_n, Δt is the current temperature and p is a random float between 0 and 1. S_n will be accepted when it is better than S_c. However, if S_n is worse than S_c, it is accepted with a probability of p.

V. Computational Experiments

In this section, we describe parameter settings and conduct experiments to evaluate the performance of the HLS algorithm. First of all, the rules with priority instances are presented, and then the parameter settings for the algorithm are presented. Second, several comparison experiments are performed by HLS and two effective metaheuristics on CCVRP-Pr and basic CCVRP respectively. Finally, we analyze the performance of hybridization and the influence of different perturbation strengths. The algorithm is implemented in C++ using Visual Studio Code, which is available at https://github.com/chyoungerz/HLS. All experiments are conducted on Windows 10 64-bit with an Intel i7-10700 CPU @ 2.90 GHz.

A. Benchmark Instances

Since CCVRP-Pr is a relatively new problem and no international benchmarks with priorities are available, we generated 82 instances A*, B*, E*, and P*, which based on existing benchmark instances A, B, E, and P [13], with sizes ranging from 16 to 101. These adapted instances are publicly available at https://github.com/chyoungerz/PrK_Instances. Each case name consists of four parts separated by '-', such as "A-n32-k5-Pr," where "A" represents the instance set, "n" is the number of nodes including the depot and customers, "k" is the maximum number of vehicles used, and the last part indicates that each customer has a corresponding priority. When generating the instances, customers were clustered using k-means, with the k-value determined by the elbow method, ranging from 1 to 10. The priority of customers within each cluster was assigned based on the square of the distance from the cluster center, decreasing from highest ($\lceil \ln n \rceil$) to lowest (1), simulating real-world disaster scenarios. The priority assignment formula is:

$$P_i = \frac{1}{\left(\frac{1}{\sqrt{n}} + \frac{d_{ci}}{r}\right)^2} \quad (16)$$

where P_i is the priority of customer i, n is the total number of customers, r is the radius of the cluster that is the distance from the farthest point in the cluster to the cluster center, and d_{ci} is the distance from customer i to the cluster center.

B. Parameters Settings

The HLS algorithm has some parameters that need to be set. The size of the neighborhood is set to half of the customer number. The maximum iteration of perturbation in the ruin-recreate operator is set to 10 and the destruction range is set to (0, 0.5). The k in the ejection chain operator is [2, r), where r is the number of total routes, and the range of perturbation iteration is [1, 10). The perturbation intensity of two methods is randomly selected at the range of (0.2, 1.0). For the acceptance rule based on simulated annealing, the parameters include an initial temperature of 1 and a cooling rate of 0.94. Besides, the algorithm will be terminated if the solution does not improve for 20 consecutive iterations.

TABLE I
COMPREHENSIVE RESULTS OF DIFFERENT ALGORITHMS ON A*, B*, E*, AND P* INSTANCES

Instances	ILS		VNS		HLS	
	#BKS	T/s	#BKS	T/s	#BKS	T/s
A*(27)	10	1.41	7	1.33	23	2.02
B*(22)	9	1.40	6	1.12	17	2.15
E*(11)	4	7.03	2	6.62	10	9.32
P*(22)	16	4.12	5	4.44	15	3.92
Total(82)	39	13.96	20	13.51	65	17.41

TABLE II
RESULTS FOUND BY HLS AND COMPARISON ALGORITHMS ON SET A*

Instance	BKS	ILS		VNS		HLS	
		best	avg	best	avg	best	avg
A-n32-k5-Pr	**2653**	**2653**	2655.00	**2653**	2775.10	**2653**	2664.90
A-n33-k5-Pr	**1894**	**1894**	1902.50	**1894**	1926.90	**1894**	1915.70
A-n33-k6-Pr	**2095**	**2095**	2121.00	2111	2136.80	**2095**	2118.00
A-n34-k5-Pr	**2533**	**2533**	2552.30	2540	2575.50	**2533**	2539.60
A-n36-k5-Pr	**3165**	**3165**	3188.20	**3165**	3219.50	**3165**	3187.60
A-n37-k5-Pr	**2415**	**2415**	2458.10	2454	2471.00	2454	2467.60
A-n37-k6-Pr	**2884**	2918	2977.20	2934	3045.50	**2884**	2983.90
A-n38-k5-Pr	**2500**	**2500**	2598.30	2641	2913.60	**2500**	2569.50
A-n39-k5-Pr	**3170**	3210	3237.10	3221	3503.40	**3170**	3244.60
A-n39-k6-Pr	**2664**	**2664**	2667.30	2687	2687.20	**2664**	2670.80
A-n44-k6-Pr	**3475**	3500	3570.70	3478	3635.30	**3475**	3536.20
A-n45-k6-Pr	**3954**	4195	4455.50	4078	4367.00	**3954**	4169.10
A-n45-k7-Pr	**3516**	**3516**	3520.10	**3516**	3565.90	**3516**	3519.00
A-n46-k7-Pr	**3061**	**3061**	3073.50	3069	3069.00	**3061**	3072.90
A-n48-k7-Pr	**3920**	3921	3977.10	**3920**	4014.90	3921	3947.50
A-n53-k7-Pr	**3859**	3918	4073.10	4018	4256.00	**3859**	4005.00
A-n54-k7-Pr	**4374**	4502	4602.30	4525	4651.10	**4374**	4496.30
A-n55-k9-Pr	**3304**	3314	3396.80	3430	3469.60	**3304**	3365.10
A-n60-k9-Pr	**4601**	4605	4711.50	4605	4728.30	**4601**	4617.50
A-n61-k9-Pr	**3740**	4397	4516.00	3822	4116.90	**3740**	4099.10
A-n62-k8-Pr	**5047**	5098	5139.40	5122	5185.70	**5047**	5104.20
A-n63-k9-Pr	**6423**	6640	6760.30	**6423**	6598.50	6426	6524.10
A-n63-k10-Pr	**4183**	4223	4351.20	4241	4379.70	**4183**	4305.70
A-n64-k9-Pr	**5304**	5316	5503.00	**5304**	5413.80	5313	5433.70
A-n65-k9-Pr	**4963**	5433	5647.50	5181	5420.40	**4963**	5189.20
A-n69-k9-Pr	**4666**	4790	4946.80	4685	4816.50	**4666**	4787.00
A-n80-k10-Pr	**7694**	7744	7946.50	7709	7786.30	**7694**	7814.50
Avg	3779.89	3860.00	3946.23	3830.59	3952.94	3781.81	3864.75
gap/%		2.12		1.34		0.05	

C. Computational Results on CCVRP-Pr

To verify the performance of the HLS algorithm, we conduct a series of comparative experiments with two effective local-search based algorithms, which are VNS and ILS. The reasons why we chose the two algorithms are that this problem is a relatively new one with no public algorithms to compare and the design of the HLS algorithm is inspired by these two algorithms. Furthermore, ILS and VNS are classical algorithms for solving CCVRP problems, and they have been proven to be highly effective for this class of problems[14], [12]. This can facilitate performance verification. The VNS algorithm uses four shake operators and the ILS algorithm uses the simulated annealing acceptance. The parameter settings of the two are the same as those of the HLS algorithm and the operators used by the three algorithms are also the same.

The three algorithms were tested on the A*, B*, P*, and E* instance sets and each instance ran 10 times. The number of optimal solutions found and the average elapsed time for each algorithm are summarized in Table I. In Table I, the first column shows the instance sets, and columns #BKS and T/s show the number of best-known solutions found and the average time to find the best solution.

According to the results of Table I, we reveal that the HLS algorithm outperforms the VNS and ILS algorithms. The HLS algorithm found 65 best solutions out of the 82 instances and its success rate is 79.27%. The VNS and

TABLE III
RESULTS FOUND BY HLS AND COMPARISON ALGORITHMS ON SET B*

Instance	BKS	ILS		VNS		HLS	
		best	avg	best	avg	best	avg
B-n31-k5-Pr	**2128**	**2128**	2132.00	2131	2141.20	2131	2131.10
B-n34-k5-Pr	**2871**	**2871**	2895.40	**2871**	2927.20	**2871**	2878.60
B-n35-k5-Pr	**3912**	**3912**	3920.20	**3912**	3963.70	**3912**	3912.00
B-n38-k6-Pr	**2307**	**2307**	2310.40	2318	2320.80	**2307**	2310.50
B-n39-k5-Pr	**2707**	**2707**	2707.50	2708	2708.00	**2707**	2708.00
B-n41-k6-Pr	**2868**	2880	2896.40	2911	3039.40	**2868**	2905.90
B-n43-k6-Pr	**2970**	**2970**	2985.10	**2970**	3020.30	**2970**	2986.10
B-n44-k7-Pr	**3660**	3666	3674.40	3664	3682.60	**3660**	3676.00
B-n45-k5-Pr	**3344**	**3344**	3471.60	3371	3552.10	3369	3440.60
B-n45-k6-Pr	**2333**	2404	2546.70	2566	2676.50	**2333**	2486.50
B-n50-k7-Pr	**2708**	**2708**	2722.30	**2708**	2719.60	**2708**	2710.50
B-n50-k8-Pr	**3644**	3657	3670.00	3650	3678.40	**3644**	3661.50
B-n51-k7-Pr	**3853**	4263	4438.40	4369	4544.60	**3853**	4041.90
B-n52-k7-Pr	**3096**	3108	3123.90	3155	3160.60	**3096**	3114.80
B-n56-k7-Pr	**3455**	3473	3542.90	**3455**	3629.50	3459	3524.90
B-n57-k7-Pr	**5552**	6025	6205.10	6002	6416.10	**5552**	6013.60
B-n57-k9-Pr	**5833**	5848	5899.00	5956	5988.20	**5833**	5871.20
B-n63-k10-Pr	**7120**	7166	7255.10	**7120**	7217.60	7182	7227.70
B-n64-k9-Pr	**3714**	3813	3947.20	3776	3985.90	**3714**	3771.20
B-n66-k9-Pr	**7074**	7179	7212.50	7123	7172.80	**7074**	7150.90
B-n67-k10-Pr	**4039**	4049	4172.80	4070	4164.20	**4039**	4118.80
B-n68-k9-Pr	**5651**	**5651**	5869.00	5740	5843.00	5738	5814.20
Avg	3856.32	3914.95	3981.72	3933.91	4025.10	3864.55	3929.84
gap/%		1.52		2.01		0.21	

TABLE IV
RESULTS FOUND BY HLS AND COMPARISON ALGORITHMS ON SET E*

Instance	BKS	ILS		VNS		HLS	
		best	avg	best	avg	best	avg
E-n22-k4-Pr	**896**	**896**	904.20	970	975.00	**896**	896.00
E-n23-k3-Pr	**2178**	**2178**	2178.00	2324	2324.00	**2178**	2195.10
E-n30-k3-Pr	**2261**	**2261**	2266.10	2276	2304.80	**2261**	2265.20
E-n33-k4-Pr	**3380**	**3380**	3391.80	**3380**	3399.40	**3380**	3402.80
E-n51-k5-Pr	**2527**	2655	2773.60	2747	2938.80	**2527**	2731.00
E-n76-k7-Pr	**4211**	4217	4293.20	4222	4394.50	**4211**	4281.40
E-n76-k8-Pr	**3935**	3970	4114.30	3950	4051.20	**3935**	4040.40
E-n76-k10-Pr	**3557**	3729	3915.90	3577	3748.50	**3557**	3696.60
E-n76-k14-Pr	**2542**	2692	2794.10	2683	2772.30	**2542**	2629.70
E-n101-k8-Pr	**5644**	5802	5937.60	5676	5841.90	**5644**	5811.30
E-n101-k14-Pr	**3573**	3694	3763.40	**3573**	3692.70	3617	3693.50
Avg	3154.91	3224.91	3302.93	3216.18	3313.01	3158.91	3240.27
gap/%		2.22		1.94		0.13	

ILS algorithms only found 20 and 39 best solutions, whose success rate is 24.40% and 47.56%. In the A*, B*, and E* instances, HLS found more optimal solutions than VNS and ILS, but in the P* instance, HLS is equals to ILS. The computational time of the three algorithms is almost the same. Overall, all three algorithms could find solutions within a limited time. The HLS shows superior performance in obtaining best solutions.

The optimal and average solutions for each instance set are illustrated in Table II to Table V. Columns best and avg denote the best solution and average solution found by the algorithm. The column BKS represent the best solution among all the algorithms. The best solutions are shown in bold in these tables. The gap value is defined as:

$$gap = \frac{best\ -\ BKS}{BKS} \times 100\% \tag{17}$$

As reported in Table II to Table V, the HLS outperforms the VNS and ILS and has the best average solution. In detail, the gap of the HLS is 0.05% on Set A* and much smaller than 1.34% and 2.12% of the VNS and ILS. Meanwhile, on Set B*, E* and P*, the gap of the HLS are 0.21%, 0.13%, and 0.31%, which are also smaller than those of the VNS and

TABLE V
RESULTS FOUND BY HLS AND COMPARISON ALGORITHMS ON SET P*

Instance	BKS	ILS		VNS		HLS	
		best	avg	best	avg	best	avg
P-n16-k8-Pr	**438**	**438**	438.00	**438**	439.20	**438**	438.00
P-n19-k2-Pr	**974**	**974**	990.30	1074	1070.00	**974**	985.20
P-n20-k2-Pr	**1086**	**1086**	1105.60	**1086**	1112.10	**1086**	1106.40
P-n21-k2-Pr	**1072**	**1072**	1076.50	**1072**	1164.70	**1072**	1083.60
P-n22-k8-Pr	**709**	**709**	709.00	739	739.00	**709**	709.40
P-n23-k8-Pr	**787**	**787**	807.10	863	863.00	792	805.10
P-n40-k5-Pr	**1773**	**1773**	1778.30	**1773**	1775.80	**1773**	1776.70
P-n45-k5-Pr	**2293**	**2293**	2334.60	2485	2542.40	2312	2357.80
P-n50-k7-Pr	**1892**	**1892**	1926.40	1922	1976.40	1909	1939.30
P-n50-k8-Pr	**1733**	1961	2050.40	1953	2130.50	**1733**	1891.30
P-n50-k10-Pr	**1396**	**1396**	1442.40	1482	1529.60	**1396**	1432.50
P-n51-k10-Pr	**1543**	1598	1734.80	1655	1736.00	**1543**	1617.10
P-n55-k7-Pr	**1854**	**1854**	1874.70	1875	1891.90	**1854**	1877.40
P-n55-k8-Pr	**1889**	**1889**	1913.60	1918	1934.50	**1889**	1911.50
P-n55-k10-Pr	**1507**	**1507**	1524.30	**1507**	1540.50	1509	1522.40
P-n60-k10-Pr	**2249**	2266	2350.90	2316	2355.40	**2249**	2298.00
P-n60-k15-Pr	**1872**	1879	1928.60	1909	1959.80	**1872**	1903.40
P-n65-k10-Pr	**2582**	2607	2680.80	2682	2715.50	**2582**	2636.20
P-n70-k10-Pr	**2896**	3046	3222.00	2953	3156.50	**2896**	3006.10
P-n76-k4-Pr	**7108**	**7108**	7345.30	7237	7513.70	7141	7281.10
P-n76-k5-Pr	**5665**	**5665**	6090.50	5870	6113.30	5690	5846.00
P-n101-k4-Pr	**9651**	**9651**	9960.20	10369	10668.60	9716	9970.20
Avg	2407.68	2429.59	2512.92	2507.91	2587.65	2415.23	2472.49
gap/%		0.91		4.16		0.31	

ILS. Overall, the average best solution of all 82 instances was 3348.40. The HLS achieves 3353.80, the VNS is 3421.02, and the ILS is 3405.78. To conclude, the HLS is superior to the VNS and ILS in overall performance.

D. Computational Results on CCVRP

The above experiments can conclude that HLS algorithm has the best solution performance among these three algorithms, and in order to prove the performance of the HLS algorithm even further, we removed the code that calculates the priority in the algorithm in order to make the algorithm ignore the priority of the nodes, so that the optimization objective of the algorithm becomes the basic CCVRP. Meanwhile, the termination condition was modified to 50 consecutive iterations without improvement, the cooling rate was adjusted from 0.94 to 0.99 to align with the new termination condition but the other parameters remain the same. We used the A, B, E&M, and P instances to test our algorithm. Each instance was run 10 times and the best solution was compared with the internationally published algorithms for solving the CCVRP, such as IG-PRB and IG-CE[14], ACS-VND and MMAS-VND[18], and HTS-VND[12]. All the results are shown in Tables VI to IX. Columns best and avg denote the best solution and the average solution found by HLS. Columns T/s represents the average CPU time per running. Column BKS represents the best solution so far and the BKS with underline is not proven to be optimal.

From the Tables VI to IX, by comparing the gap values, we can find that the HLS outperformed the IG-CE algorithm on all sets of instances. In particular, the HLS outperformed the IG-PRB and MMAS-VND in the B by a slight margin. While the HLS algorithm did not demonstrate the highest performance among the six algorithms when comparing the gap values, it is crucial to acknowledge that the HLS algorithm was designed to solve the problem with customer priority. In this regard, we have taken into account the order of priority in each operator, particularly in the perturbation and initial solution generation. In this study, we have solely eliminated the priority computation without much modifications to the algorithm. The solution quality of the HLS algorithm is within 0.3% of the results of the best algorithm (HTS-VND), which demonstrates the HLS is still advantageous in solving the basic CCVRP.

E. Stability Analysis

In this section, the computational experimental results of the proposed HLS algorithm on all instances are analyzed to verify its stability. Figure 8 is the comparison of the average standard deviation(STD) of the HLS algorithm, ILS algorithm and VNS algorithm on all instances with running 10 times independently. After analyzing, it can be found that the average STD of the HLS algorithm is relatively small compared with the result of VNS and ILS algorithms, among which the average STD of the VNS algorithm is the largest. The results show that the HLS algorithm has a high stability in solving the CCVRP-Pr problem compared with the VNS and ILS algorithms.

TABLE VI
RESULTS FOUND BY COMPARISON ALGORITHMS ON SET A

Instance	BKS	IG-PRB	IG-CE	ACO-VND	MMAS-VND	HTS-VND	HLS		
							Best	Avg	T/s
A-n32-k5	**2192**	**2192**	**2192**	**2192**	**2192**	**2192**	**2192**	2192.00	0.45
A-n33-k5	**1725**	**1725**	**1725**	**1725**	**1725**	**1725**	**1725**	1725.00	0.39
A-n33-k6	**1612**	**1612**	**1612**	**1612**	**1612**	**1612**	**1612**	1612.00	0.44
A-n34-k5	**2104**	**2104**	**2104**	**2104**	**2104**	**2104**	**2104**	2105.20	0.49
A-n36-k5	**2279**	**2279**	**2279**	**2279**	**2279**	**2279**	**2279**	2279.00	0.55
A-n37-k5	**1970**	**1970**	**1970**	**1970**	**1970**	**1970**	**1970**	1970.00	0.47
A-n37-k6	**2241**	**2241**	**2241**	**2241**	**2241**	**2241**	**2241**	2242.20	0.57
A-n38-k5	**2084**	**2084**	**2084**	**2084**	**2084**	**2084**	**2084**	2085.00	0.69
A-n39-k5	**2312**	**2312**	**2312**	**2312**	**2312**	**2312**	**2312**	2318.50	0.76
A-n39-k6	**2216**	**2216**	**2216**	**2216**	**2216**	**2216**	**2216**	2216.00	0.66
A-n44-k6	**2563**	**2563**	**2563**	**2563**	**2563**	**2563**	**2563**	2565.90	1.01
A-n45-k6	**2848**	**2848**	**2848**	**2848**	**2848**	**2848**	**2848**	2918.10	1.26
A-n45-k7	**2831**	**2831**	**2831**	**2831**	**2831**	**2831**	**2831**	2832.50	0.88
A-n46-k7	**2373**	**2373**	**2373**	**2373**	**2373**	**2373**	**2373**	2373.00	0.84
A-n48-k7	**3101**	**3101**	**3101**	**3101**	**3101**	**3101**	**3101**	3101.00	1.02
A-n53-k7	**3115**	**3115**	3122	**3115**	**3115**	**3115**	**3115**	3123.80	1.55
A-n54-k7	**3357**	**3357**	3382	**3357**	**3357**	**3357**	3363	3384.10	2.14
A-n55-k9	**2588**	**2588**	**2588**	**2588**	**2588**	**2588**	**2588**	2591.00	1.54
A-n60-k9	**3446**	**3446**	**3446**	**3446**	**3446**	**3446**	**3446**	3450.00	2.22
A-n61-k9	**2868**	**2868**	2869	**2868**	2879	**2868**	2887	2954.40	2.91
A-n62-k8	**3925**	**3925**	**3925**	**3925**	**3925**	**3925**	**3925**	3928.00	2.24
A-n63-k9	**4630**	**4630**	4681	**4630**	**4630**	**4630**	**4630**	4654.90	3.16
A-n63-k10	**3256**	**3256**	**3256**	**3256**	**3256**	**3256**	**3256**	3272.10	2.52
A-n64-k9	**4135**	**4135**	4137	**4135**	**4135**	**4135**	**4135**	4145.50	3.40
A-n65-k9	**3487**	**3487**	3549	**3487**	**3487**	**3487**	**3487**	3519.10	2.98
A-n69-k9	**3528**	**3528**	**3528**	**3528**	**3528**	**3528**	**3528**	3536.20	3.50
A-n80-k10	**5929**	**5929**	5944	**5929**	**5929**	**5929**	**5929**	5948.10	5.39
Avg	**2915.37**	**2915.37**	2921.41	**2915.37**	2915.78	**2915.37**	2916.30	2927.50	1.63
gap/%		0.00	0.21	0.00	0.01	0.00	0.03		

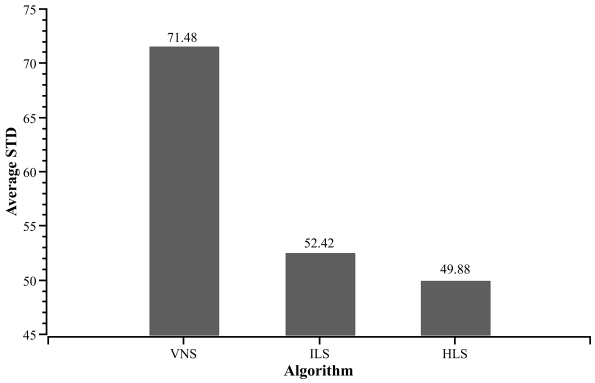


Fig. 8. Average STD of HLS and other algorithms

F. Performance Analysis of Hybridization

In this section, we will validate the effectiveness of the improved perturbation operator method (PM), VND, and acceptance rule based on simulated annealing (SA) on algorithm enhancement through experiments. We designed seven new algorithms for this purpose. The first is the basic ILS algorithm, which randomly uses one of the common perturbation operators (2-opt, Or-opt, arc-node swap, and arc-swap), and it executes LS operators in sequence, also its acceptance rule is only the improvement acceptance rule. The second algorithm uses the improved perturbation operator method on the basic ILS (denoted as ILS+PM), the third uses VND to replace common LS on the basic ILS (denoted as ILS+VND), and the fourth utilizes the acceptance rule based on simulated annealing on the basic ILS (denoted as ILS+SA). The remaining three algorithms are combinations of PM, VND, and SA on the basic ILS, denoted as ILS+PM+VND, ILS+PM+SA, and ILS+VND+SA. Each of these algorithms was executed 10 times independently, and the results are calculated.

Figure 9 illustrates the comparison results of using a single improvement method and multiple improvement methods. From the point of view of average best solutions values, all improvement measures enhance the algorithm's performance, with ILS+PM showing the most significant improvement. The value of ILS+PM is 3385, outperforming ILS+VND and ILS+SA, and it found the best of best solutions. For the comparison results of using multiple improvement methods, where HLS performed the best among the five algorithms, with a result of 3354 and the highest number of best solutions. The comparison of two figures indicates that the performance of combined methods exceeds that of individual methods. Further analysis shows that the improved perturbation operator has a greater enhancement effect on algorithm performance than VND and simulated annealing. The algorithms using single improvement method and multiple improvement methods have the similar solution time in total.

TABLE VII
RESULTS FOUND BY COMPARISON ALGORITHMS ON SET B

Instance	BKS	IG-PRB	IG-CE	ACO-VND	MMAS-VND	HTS-VND	HLS		
							Best	Avg	T/s
B-n31-k5	**1830**	**1830**	**1830**	**1830**	**1830**	**1830**	**1830**	1830.00	0.34
B-n34-k5	**2271**	**2271**	**2271**	**2271**	**2271**	**2271**	**2271**	2271.00	0.54
B-n35-k5	**2846**	**2846**	**2846**	**2846**	**2846**	**2846**	**2846**	2846.00	0.41
B-n38-k6	**2103**	2164	2164	**2103**	**2103**	**2103**	**2103**	2103.00	0.50
B-n39-k5	**1960**	**1960**	**1960**	**1960**	**1960**	**1960**	**1960**	1960.00	0.59
B-n41-k6	**2329**	**2329**	**2329**	**2329**	**2329**	**2329**	**2329**	2329.20	0.78
B-n43-k6	**2123**	**2123**	**2123**	**2123**	**2123**	**2123**	**2123**	2123.00	0.83
B-n44-k7	**2295**	**2295**	**2295**	**2295**	**2295**	**2295**	**2295**	2296.10	0.99
B-n45-k5	**2386**	**2386**	**2386**	**2386**	**2386**	**2386**	**2386**	2388.30	1.17
B-n45-k6	**2057**	**2057**	2110	**2057**	**2057**	**2057**	2058	2077.50	1.22
B-n50-k7	**2261**	**2261**	**2261**	**2261**	**2261**	**2261**	**2261**	2261.00	1.26
B-n50-k8	**2953**	**2953**	**2953**	**2953**	**2953**	**2953**	**2953**	2953.90	1.35
B-n51-k7	**3133**	**3133**	**3133**	**3133**	**3133**	**3133**	**3133**	3143.00	1.72
B-n52-k7	**2573**	**2573**	**2573**	**2573**	**2573**	**2573**	**2573**	2573.00	1.42
B-n56-k7	**2358**	**2358**	**2358**	**2358**	**2358**	**2358**	**2358**	2358.00	1.61
B-n57-k7	**3883**	3885	4340	**3883**	3918	**3883**	3886	4018.50	2.57
B-n57-k9	**4500**	**4500**	**4500**	**4500**	**4500**	**4500**	**4500**	4500.10	1.71
B-n63-k10	**4379**	**4379**	**4379**	**4379**	**4379**	**4379**	**4379**	4392.00	2.32
B-n64-k9	**2608**	**2608**	2621	**2608**	2632	**2608**	**2608**	2633.20	3.41
B-n66-k9	**4120**	4132	4175	4132	4132	**4120**	4134	4161.70	3.13
B-n67-k10	**2868**	**2868**	**2868**	**2868**	**2868**	**2868**	**2868**	2870.00	2.44
B-n68-k9	**4058**	**4058**	**4058**	**4058**	4059	**4058**	**4058**	4060.20	3.36
Avg	**2813.36**	2816.77	2842.41	2813.91	2816.64	**2813.36**	2814.18	2824.94	1.53
gap/%		0.12	1.03	0.02	0.12	0.00	0.03		

TABLE VIII
RESULTS FOUND BY COMPARISON ALGORITHMS ON SET E&M

Instance	BKS	IG-PRB	IG-CE	ACO-VND	MMAS-VND	HTS-VND	HLS		
							Best	Avg	T/s
E-n22-k4	**845**	**845**	**845**	**845**	**845**	**845**	**845**	845.00	0.16
E-n23-k3	**1908**	**1908**	**1908**	**1908**	**1908**	**1908**	**1908**	1908.00	0.22
E-n30-k3	**1984**	**1984**	**1984**	1987	1987	1987	1987	1988.30	0.38
E-n33-k4	**2852**	**2852**	**2852**	**2852**	**2852**	**2852**	**2852**	2852.00	0.54
E-n51-k5	**2213**	**2213**	**2213**	**2213**	**2213**	**2213**	**2213**	2240.30	1.83
E-n76-k7	**2920**	**2920**	**2920**	**2920**	**2920**	2926	**2920**	2922.90	4.05
E-n76-k8	**2686**	**2686**	2703	**2686**	**2686**	2687	**2686**	2695.60	4.66
E-n76-k10	**2366**	**2366**	2557	**2366**	**2366**	**2366**	**2366**	2425.70	5.28
E-n76-k14	**2080**	2093	2093	**2080**	2081	**2080**	2086	2124.30	5.19
E-n101-k8	**3954**	**3954**	3968	**3954**	**3954**	**3954**	**3954**	3984.00	10.10
E-n101-k14	**2955**	**2955**	2976	**2955**	**2955**	2958	**2955**	2980.10	8.64
M-n101-k10	**3565**	**3565**	3574	**3565**	**3565**	**3565**	**3565**	3573.00	9.09
M-n121-k7	**7223**	7230	7359	7230	7234	**7223**	7236	7324.20	24.28
M-n151-k12	**4917**	**4917**	4980	**4917**	**4917**	**4917**	**4917**	4961.40	30.10
Avg	**3033.43**	3034.86	3066.57	3034.14	3034.50	3034.36	3035.00	3058.91	7.46
gap/%		0.05	1.10	0.02	0.04	0.03	0.05		

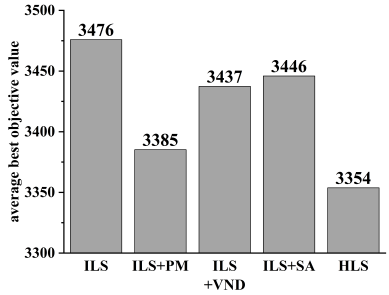
All of these algorithms can obtain satisfactory solutions in a relatively short time and our HLS algorithm performs the best of all.

G. Influence Analysis of Different Perturbation Strengths

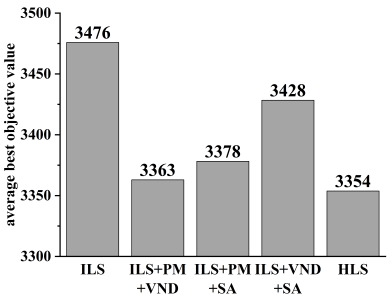
From the comparative experiments in aboved section, we can conclude that the perturbation operator significantly enhances overall solution performance. When designing the perturbation operator, we selected a perturbation intensity range of (0.2, 1.0), which chooses randomly within that range each time. To further validate the effectiveness of this perturbation intensity strategy, we designed two different perturbation intensity selection strategies, both related to the temperature of simulated annealing. The first strategy increases perturbation intensity as temperature drops, starting with a minimum of 0.2 and gradually increasing to 1 as the temperature drops. The second intensity is the opposite, decreasing perturbation intensity as temperature drops. These two algorithms are denoted as HLS(1-T) and HLS(T). Each

TABLE IX
RESULTS FOUND BY COMPARISON ALGORITHMS ON SET P

Instance	BKS	IG-PRB	IG-CE	ACO-VND	MMAS-VND	HTS-VND	HLS		
							Best	Avg	T/s
P-n16-k8	**396**	**396**	**396**	**396**	**396**	**396**	**396**	396.00	0.02
P-n19-k2	**849**	**849**	**849**	**849**	**849**	**849**	**849**	849.00	0.14
P-n20-k2	**924**	**924**	**924**	**924**	**924**	**924**	**924**	924.00	0.17
P-n21-k2	**928**	**928**	**928**	**928**	**928**	**928**	**928**	928.00	0.16
P-n22-k8	**681**	**681**	**681**	**681**	**681**	**681**	**681**	681.00	0.09
P-n23-k8	**616**	**616**	**616**	**616**	**616**	**616**	**616**	618.60	0.15
P-n40-k5	**1541**	**1541**	**1541**	**1541**	**1541**	**1541**	**1541**	1541.00	0.68
P-n45-k5	**1894**	**1894**	**1894**	**1894**	**1894**	**1894**	**1894**	1897.50	0.93
P-n50-k7	**1554**	**1554**	**1554**	**1554**	**1554**	**1554**	**1554**	1554.40	1.28
P-n50-k8	**1533**	**1533**	1682	**1533**	1540	**1533**	1541	1588.10	1.92
P-n50-k10	**1347**	**1347**	**1347**	**1347**	**1347**	**1347**	**1347**	1349.50	1.15
P-n51-k10	**1487**	1489	1489	**1487**	**1487**	**1487**	1489	1509.40	1.73
P-n55-k7	**1764**	**1764**	**1764**	**1764**	**1764**	**1764**	**1764**	1764.00	1.33
P-n55-k8	**1620**	**1620**	**1620**	**1620**	**1620**	**1620**	**1620**	1620.00	1.32
P-n55-k10	**1463**	**1463**	**1463**	**1463**	**1463**	**1463**	**1463**	1467.00	1.34
P-n60-k10	**1704**	**1704**	1705	**1704**	**1704**	**1704**	**1704**	1709.40	1.82
P-n60-k15	**1509**	**1509**	1517	**1509**	**1509**	**1509**	**1509**	1512.50	1.77
P-n65-k10	**1948**	**1948**	1949	**1948**	**1948**	**1948**	**1948**	1950.80	2.38
P-n70-k10	**2121**	**2121**	2264	**2121**	**2121**	**2121**	**2121**	2163.30	4.15
P-n76-k4	**4610**	**4610**	4677	**4610**	**4610**	**4610**	**4610**	4682.60	6.13
P-n76-k5	**3795**	**3795**	3796	**3795**	**3795**	**3795**	**3795**	3870.00	4.85
P-n101-k4	**6943**	**6943**	**6943**	**6943**	**6943**	6946	**6943**	6979.00	12.15
Avg	**1873.95**	1874.05	1890.86	**1873.95**	1874.27	1874.09	1874.41	1888.87	2.08
gap/%		0.0049	0.90	0.00	0.02	0.0073	0.02		



(a) using single method



(b) using multiple methods

Fig. 9. comparison of using single or multiple methods

of these was executed 10 times independently and the results are shown in Table X, where column BN indicates the number of best solutions among these three algorithm.

As shown in Table X, we find that the HLS algorithm with random perturbation intensity strategy performed best of all. It founds 54 best solutions among three algorithm, and outperforms HLS(1-T) and HLS(T). For the analysis of gap, HLS also has the lowest gap value of 0.29%, while HLS(T) has the worst solution quality with a gap value of 1.41%. For the average time, the three algorithms almost have the same elapsed time. In conclusion, HLS has a better performance than HLS(1-T) and HLS(T).

VI. CONCLUSION

This paper study the capacitated vehicle routing problem with priorities (CCVRP-Pr), focusing on minimizing the total time to reach disaster points while considering the urgency of different disaster points in order to prioritize service to severely affected areas. To address this issue, we develop a hybrid local search algorithm based on iterated local search, which incorporates improvements to the perturbation, local search, and acceptance criteria components. Our experimental findings demonstrate that these enhancements led to enhance algorithmic solution quality, Experimental results also reveal that our proposed algorithm can find better solutions in reasonable computation time compared with the local-search based metaheuristics.

From the experimental results, we have some findings. First, all the improvement strategies, including the improved perturbation operator method, variable neighborhood descent procedure, and simulated annealing-based acceptance rule, designed in our algorithm can enhance the algorithm's solution quality. Second, the combination of improvement strategies outperforms the single improvement strategy used in the proposed algorithm. Meanwhile, our algorithm is still advantageous in solving the basic CCVRP. Finally, our algorithm with random perturbation intensity has proven

TABLE X
COMPARISON RESULTS OF DIFFERENT PERTURBATION STRENGTH

	HLS(T)		HLS(1-T)		HLS	
	BN	T/s	BN	T/s	BN	T/s
A*(27)	8	0.97	12	2.50	19	2.02
B*(22)	6	1.09	10	2.56	13	2.15
E*(11)	5	4.42	5	11.94	7	9.32
P*(22)	8	2.10	14	4.97	15	3.92
Total(82)	27	8.58	41	21.97	54	17.42
avg	3391.16		3368.76		3353.80	
gap	1.41		0.74		0.29	

to be more effective than those increasing or decreasing perturbation intensity as the temperature drops.

In future, the CCVRP-Pr problem with additional attributes, such as heterogeneous vehicles, multiple depots, time windows and so on, need to be considered. Additionally, the more effective algorithms could be explored to solve the CCVRP and variants because of its hard complexity.

REFERENCES

[1] S. U. Ngueveu, C. Prins, and R. Wolfler Calvo, "An effective memetic algorithm for the cumulative capacitated vehicle routing problem," *Computers & Operations Research*, vol. 37, no. 11, pp. 1877–1885, 2010.

[2] K. Corona-Gutiérrez, S. Nucamendi-Guillén, and E. Lalla-Ruiz, "Vehicle routing with cumulative objectives: A state of the art and analysis," *Computers & Industrial Engineering*, vol. 169, p. 108054, 2022.

[3] J. Lysgaard and S. Wøhlk, "A branch-and-cut-and-price algorithm for the cumulative capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 236, no. 3, pp. 800–810, 2014.

[4] G. Mattos Ribeiro and G. Laporte, "An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem," *Computers & Operations Research*, vol. 39, no. 3, pp. 728–735, 2012.

[5] L. Ke and Z. Feng, "A two-phase metaheuristic for the cumulative capacitated vehicle routing problem," *Computers & Operations Research*, vol. 40, no. 2, pp. 633–638, 2013.

[6] J. F. Sze, S. Salhi, and N. Wassan, "The cumulative capacitated vehicle routing problem with min-sum and min-max objectives: An effective hybridisation of adaptive variable neighbourhood search and large neighbourhood search," *Transportation Research Part B: Methodological*, vol. 101, pp. 162–184, 2017.

[7] L. Ke, "A brain storm optimization approach for the cumulative capacitated vehicle routing problem," *Memetic Computing*, vol. 10, no. 4, pp. 411–421, 2018.

[8] E. Lalla-Ruiz and S. Voß, "A POPMUSIC approach for the multi-depot cumulative capacitated vehicle routing problem," *Optimization Letters*, vol. 14, no. 3, pp. 671–691, 2020.

[9] X. Wang, T.-M. Choi, Z. Li, and S. Shao, "An effective local search algorithm for the multidepot cumulative capacitated vehicle routing problem," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 12, pp. 4948–4958, 2020.

[10] A. Osorio-Mora, J. W. Escobar, and P. Toth, "An iterated local search algorithm for latency vehicle routing problems with multiple depots," *Computers & Operations Research*, vol. 158, p. 106293, 2023.

[11] R. Liu and Z. Jiang, "A hybrid large-neighborhood search algorithm for the cumulative capacitated vehicle routing problem with time-window constraints," *Applied Soft Computing*, vol. 80, pp. 18–30, 2019.

[12] N. A. Kyriakakis, I. Sevastopoulos, M. Marinaki, and Y. Marinakis, "A hybrid tabu search – variable neighborhood descent algorithm for the cumulative capacitated vehicle routing problem with time windows in humanitarian applications," *Computers & Industrial Engineering*, vol. 164, p. 107868, 2022.

[13] N. Christofides, A. Mingozzi, and P. Toth, *The vehicle routing problem*. Chichester: John Wiley & Sons Ltd, 1979, pp. 315–338.

[14] S. Nucamendi-Guillén, F. Angel-Bello, I. Martínez-Salazar, and A. E. Cordero-Franco, "The cumulative capacitated vehicle routing problem: New formulations and iterated greedy algorithms," *Expert Systems with Applications*, vol. 113, pp. 315–327, 2018.

[15] A. M. Campbell, D. Vandenbussche, and W. Hermann, "Routing for relief efforts," *Transportation Science*, vol. 42, no. 2, pp. 127–145, 2008.

[16] N. Smiti, M. M. Dhiaf, B. Jarboui, and S. Hanafi, "Skewed general variable neighborhood search for the cumulative capacitated vehicle routing problem," *International Transactions in Operational Research*, vol. 27, no. 1, pp. 651–664, 2020.

[17] B. L. Golden, E. A. Wasil, J. P. Kelly, and I.-M. Chao, *The Impact of Metaheuristics on Solving the Vehicle Routing Problem: Algorithms, Problem Sets, and Computational Results*. Boston, MA: Springer US, 1998, pp. 33–56.

[18] N. A. Kyriakakis, M. Marinaki, and Y. Marinakis, "A hybrid ant colony optimization-variable neighborhood descent approach for the cumulative capacitated vehicle routing problem," *Computers & Operations Research*, vol. 134, p. 105397, 2021.

[19] S. Nucamendi-Guillén, D. Flores-Díaz, E. Olivares-Benitez, and A. Mendoza, "A memetic algorithm for the cumulative capacitated vehicle routing problem including priority indexes," *Applied Sciences*, vol. 10, no. 11, 2020.

[20] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[21] A. X. Martins, C. Duhamel, P. Mahey, R. R. Saldanha, and M. C. de Souza, "Variable neighborhood descent with iterated local search for routing and wavelength assignment," *Computers & Operations Research*, vol. 39, no. 9, pp. 2133–2141, 2012.

[22] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck, "Record breaking optimization results using the ruin and recreate principle," *Journal of Computational Physics*, vol. 159, no. 2, pp. 139–171, 2000.