An effective hybrid evolutionary algorithm for the clustered orienteering problem

Qinghua Wu [a], Mu He [a], Jin-Kao Hao [b], Yongliang Lu [c,*]

[a] *School of Management, Huazhong University of Science and Technology, 430074 Wuhan, China*
[b] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*
[c] *School of Economics and Management, Fuzhou University, 350116 Fuzhou, China*

European Journal of Operational Research,
https://doi.org/10.1016/j.ejor.2023.08.006

Abstract

In this paper, we study a variant of the orienteering problem called the clustered orienteering problem. In this problem, customers are grouped into clusters. A profit is associated with each cluster and is collected if and only if all customers in the cluster are served. A single vehicle is available to visit the customers. The goal is to maximize the total profits collected within a maximum travel time limit. To address this NP-hard problem, we propose the first evolutionary algorithm that integrates a backbone-based crossover operator and a destroy-and-repair mutation operator for search diversification and a solution-based tabu search procedure reinforced by a reinforcement learning mechanism for search intensification. The experiment results indicate that our algorithm outperforms the state-of-the-art algorithms from the literature on a wide range of 924 well-known benchmark instances. In particular, the proposed algorithm obtains new records (new lower bounds) for 14 instances and finds the best-known solutions for the remaining instances. Furthermore, a new set of 72 large instances with 50 to 100 clusters and at least 400 vertices is generated to evaluate the scalability of the proposed algorithm. Results show that the proposed algorithm manages to outperform three state-of-the-art COP algorithms. We also adopt our algorithm to solve a dynamic version of the COP considering stochastic travel time.

Keywords: Heuristics; Clustered orienteering problem; Tabu search; Hybrid evolutionary algorithm; Reinforcement learning

* Corresponding author.
Email addresses: `qinghuawu1005@gmail.com` (Qinghua Wu), `muhe2020@hust.edu.cn` (Mu He), `jin-kao.hao@univ-angers.fr` (Jin-Kao Hao), `luyonglianglyl@gmail.com` (Yongliang Lu).

1. Introduction

The clustered orienteering problem (COP) (Angelelli et al., 2014) is an extension of the classic orienteering problem (OP) (Tsiligirides, 1984) that aims at finding a tour across a subset of customers (starting from the depot and finishing at the depot) so as to maximize a collected profit within a given travel time budget. Unlike the classic OP, the customers in COP are grouped into clusters. A profit is associated with each cluster rather than each customer, and this profit is collected if and only if all customers in the cluster are served. A single vehicle is available to visit the customers. The goal is to maximize the total profits collected within a maximum travel time limit. Figure 1 illustrates this problem using 15 vertices (representing customers) and 4 clusters represented by different colors. Vertices of the same color belong to the same cluster, and a vertex with two colors belongs to two clusters. The profit of a cluster can be obtained only when all vertices of the cluster are visited. In Figure 1, the profits of clusters C_1, C_3, and C_4 are collected.

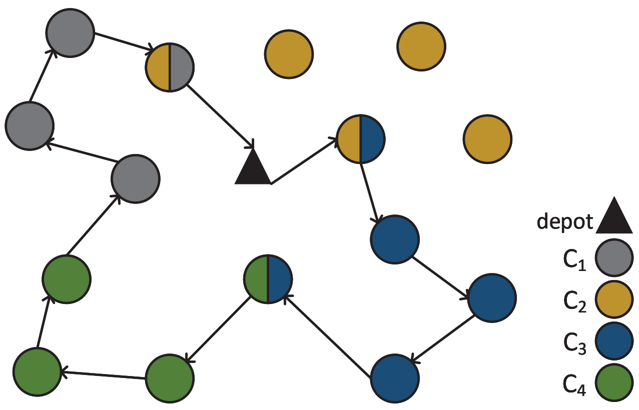


Fig. 1. Illustrative example of COP.

The COP has a number of applications in the modern logistics and transportation industry. As shown in Angelelli et al. (2014), a typical application is related to the distribution of mass products, wherein each supply chain (cluster) contains a set of retailers (customers), and if a carrier agrees to serve a supply chain, then it needs to serve all retailers in that chain. The carrier wants to acquire as much profits as possible within a given maximum travel time budget. Note that COP is equivalent to the classic NP-hard OP (Golden et al., 1987) when each cluster contains exactly one vertex. Therefore, COP is NP-hard and computationally challenging.

Despite the interest of the COP problem as a general model for a number of relevant applications, few efforts have been devoted to advancing efficient algorithms for COP in the literature compared to the classic OP (see the literature review in Section 2). Meanwhile, existing heuristic algorithms are based on local search (Angelelli et al., 2014; Yahiaoui et al., 2019). To enrich the solution approaches for solving COP, we devise the first population-based hybrid evolutionary algorithm (HEA) with several important features.

- First, the COP problem can be decomposed into two subproblems: a cluster selection subproblem at the cluster level and a routing subproblem at the customer level. Since the served customers in the lower-level routing subproblem are first determined by the selected clusters in the upper-level cluster selection subproblem, it is critical to select promising clusters effectively. For this purpose, we develop a solution-based tabu search (SBTS) procedure for cluster selection in the cluster level and apply the Lin-Kernighan heuristic (Lin and Kernighan, 1973) for the route planning in the customer level. To examine intensively the search space of the cluster selection subproblem, we design solution-based tabu search, which employs hashing technique to precisely record each visited solution and to accurately determine the tabu status of neighboring solutions. Additionally, given that the evaluation for each neighboring solution is computationally expensive due to the application of the Lin-Kernighan heuristic at the customer level, we devise a reinforcement learning strategy to guide the algorithm to focus on examining promising candidate solutions and omit non-promising solutions, thus significantly accelerating the search process.

- Second, an analysis on a sample of locally optimal COP solutions discloses that high-quality solutions always share many common clusters (see Section 6.8), which provides a strong motivation for the design of our backbone-based crossover operator. By inheriting the shared clusters from parent solutions to offspring solutions, this crossover operator contributes to the discover of very high quality solutions.

- Third, we assess the proposed algorithm on a set of 924 well-known COP benchmark instances from the literature. The experimental results demonstrate that HEA outperforms the existing state-of-the-art heuristics. In particular, it reports improved best-known solutions (new lower bounds) for 14 instances and finds the best-known solutions for all remaining instances. Besides, we further generated a new set of 72 large instances with 50 to 100 clusters and at least 400 vertices, and present our computational results on these instances, which can serve as reference values for new COP algorithms.

- Finally, to demonstrate the practical usefulness of our proposed algorithm, we adopt the proposed approach to address a real-life case (a dynamic version of COP with stochastic travel time) from an intra-city delivery platform in a large city in China. Moreover, the code of our HEA algorithm will be made available online, which can help researchers and practitioners to better solve various practical problems that can be formulated as COP.

The rest of this paper is organized as follows. Section 2 presents a review of the related literature. Section 3 provides a detailed description of the problem. Section 4 presents the proposed algorithm. Section 5 presents the computational results and comparisons with the state-of-the-art approaches. Section 6 investigates some key ingredients of the proposed algorithm. Section 7 presents a case example using real-life data to evaluate the proposed algorithm. Section 8 concludes the paper.

2. Literature review

This section provides a brief overview of the related OPs and node routing problems and a summary of the solution methods for COP.

The COP is an extension of the well-known OP that is originally proposed by Tsiligirides (1984). The OP derives from the outdoor sport game of orienteering (Chao et al., 1996a; Tsiligirides, 1984), in which each player tries to visit the checking points to collect points or scores within a given time frame. Due to its practical applications including logistics and tourism (Gunawan et al., 2016), several important variants of OP can also be found in the literature, such as the team OP (Chao et al., 1996b), the multi-profit OP (Kim et al., 2020; Kim and Kim, 2022), the probabilistic OP (Chou et al., 2021), the OP with time windows (Kantor and Rosenwein, 1992), the OP with hotel selection (Divsalar et al., 2014; Sohrabi et al., 2020), the OP with variable profits (Yu et al., 2019), the multi-visit team OP (Hanafi et al., 2020), and the stochastic OP (Campbell et al., 2011). Given the theoretical and practical significance, a large number of exact and heuristic solution algorithms have been presented for the OP and its variants over the past few decades. For more details on the recent variants and solution approaches of the OP, we refer the reader to the surveys by Vansteenwegen et al. (2011) and Gunawan et al. (2016).

The COP is a variant of the OP where the customers are grouped into clusters, a profit is associated with each cluster rather than each customer, and this profit is collected if and only if all customers in the cluster are visited. Among the OP variants, the set orienteering problem (SOP) (Archetti et al., 2018) and the clustered team orienteering problem (CTOP) (Yahiaoui et al., 2019) are two closely related problem. In the SOP, customers are grouped into clusters too, and the profit of a cluster is collected if at least one customer in that cluster is visited. To solve the SOP, a matheuristic algorithm (Archetti et al., 2018), a variable neighborhood search (Pěnička et al., 2019), a genetic algorithm (Carrabs, 2021), and an adaptive memory matheuristic algorithm (Dontas et al., 2023) have been proposed. The CTOP (Yahiaoui et al., 2019) is an extension of the COP where multiple vehicles are used to collect the profits of clusters within a limited travel time. To solve the CTOP, Yahiaoui et al. (2019) proposed an exact method based on a cutting plane approach and a hybrid heuristic that combines an adaptive large neighborhood search with an effective problem-specific split procedure.

In addition, several other node routing problems with customers divided into groups (clusters) can also be found in the literature and thus share some similarities with the COP studied in this work. For instance, in the generalized traveling salesman problem (Fischetti et al., 1997), the customers are partitioned into several subsets called clusters, and the salesman has to visit at least one customer for each cluster. In the clustered traveling salesman problem (CTSP) (Chisman, 1975), customers belonging to the same cluster must be visited contiguously. An extension of CTSP with multiple vehicles called the clustered vehicle routing problem has been introduced in Battarra et al. (2014), in which all customers of the same cluster must be served by the same vehicle. In the COP, the concept of cluster is referred to a subset of customers belonging to the same transport task, with a profit collected only when all customers in that task are served.

To the best of our knowledge, the COP has only been studied in Angelelli et al. (2014) and Yahiaoui et al. (2019). The COP was introduced by Angelelli et al. (2014) along with a set of benchmark instances, a branch-and-cut algorithm, and a tabu search (TS) algorithm. Recently, Yahiaoui et al. (2019) extended the COP by considering the use of multiple vehicles to collect the profits of several clusters and proposed an exact method and a hybrid heuristic algorithm to solve instances with several vehicles and a single vehicle.

Compared to the OP and some of its variants, few efforts have been devoted to developing efficient algorithms for the COP in the literature. To enrich the arsenal of solution methods for solving the problem, this work introduces an effective population-based hybrid evolutionary algorithm. As shown in Section 5, the proposed algorithm competes very favorably with the existing state-of-the-art COP heuristic algorithms on a wide range of 924 well-known COP benchmark instances from the literature.

3. Problem description

Let $G = (V, E)$ be a complete undirected graph, where V is a set of vertices and E represents a set of edges. In the set of vertices $V = \{v_0, v_1, \ldots, v_n\}$, v_0 is the depot (the starting and ending point of the tour), and the remaining vertices are customers. We define $C = \{C_1, C_2, \ldots, C_k\}$ such that $\cup_{i=1}^{k} C_i = V \backslash \{v_0\}$, each $C_i \in C$ is a non-empty subset of $V \backslash \{v_0\}$ called cluster and each vertex $i \in V$ belongs to at least one cluster. An integer profit p_i is associated with each cluster, and the profit is collected if and only if all customers in the cluster are served. Each edge $e \in E$ corresponds to a cost t_e, which represents the time required for visiting the edge. A single vehicle is available to visit the customers. The maximum duration of the vehicle route is limited to T_{max}. The vehicle does not need to visit all customers in a cluster successively, that is, the vehicle can first visit some customers in the cluster C_i, leave the cluster to visit other customers in another cluster C_j, and then return to C_i to continue its route. The goal is to maximize the total profits collected within the maximum time limit T_{max}.

To formally state the COP, we recall the mathematical model proposed in Angelelli et al. (2014), which is based on the following notations.
- $\delta(i)$: set of edges adjacent to the vertex v_i.
- $E(U)$: set of edges with both endpoints in U, and U is a subset of V.
- z_i: equals 1 if all customers in cluster $C_i \in C$ are served and equals 0 otherwise.
- y_j: equals 1 if vertex $v_j \in V$ is served and equals 0 otherwise.
- x_e: equals 1 if edge $e \in E$ is traversed and equals 0 otherwise.

Model

$$\max \sum_{C_i \in C} p_i z_i \tag{1}$$

s.t.

$$y_0 = 1 \tag{2}$$

$$\sum_{e \in \delta(j)} x_e = 2y_j, \qquad \forall v_j \in V \tag{3}$$

$$\sum_{e \in E} t_e x_e \leqslant T_{\max} \tag{4}$$

$$\sum_{e \in E(U)} x_e \leqslant \sum_{v_j \in U \setminus \{v_t\}} y_j, \qquad \forall U \subseteq V \setminus \{0\}, \forall v_t \in U \tag{5}$$

$$z_i \leqslant y_j, \qquad \forall C_i \in C, \forall v_j \in C_i \tag{6}$$

$$z_i \in \{0,1\}, \qquad \forall C_i \in C \tag{7}$$

$$x_e \in \{0,1\}, \qquad \forall e \in E \tag{8}$$

$$y_j \in \{0,1\}, \qquad \forall v_j \in V. \tag{9}$$

The objective function (1) aims at maximizing the total collected profit. Constraint (2) ensures that the depot is visited. Constraint (3) guarantees that the vehicle traverses two edges adjacent to each served vertex. Constraint (4) imposes that the total travel time does not exceed T_{max}, and constraint (5) represents the subtour elimination constraints. Constraint (6) ensures that the profit of a cluster is collected only if all vertices in the cluster are served. Constraints (7) to (9) define the nature of the decision variables.

4. Hybrid evolutionary algorithm for COP

4.1. *General procedure*

The proposed HEA algorithm follows the basic memetic framework (Moscato and Cotta, 2003; Hao, 2012; Neri and Cotta, 2012) combining the diversification power of genetic algorithm with the intensification strength of local search. The general scheme of HEA is summarized in Algorithm 1. HEA starts from an initial population with p feasible solutions (Section 4.2) improved with SBTS (Section 4.3). After initializing the required parameters, HEA enters the "while" loop to evolve the population iteratively. At each generation, HEA applies either a backbone-based crossover operator (Section 4.5) on two randomly selected parent solutions (Section 4.5) or a destroy-and-repair mutation operator (Section 4.6) on a single randomly selected parent to generate an offspring solution. To encourage the particular operator producing quality solutions, we employ an adaptive selection strategy inspired by Lu et al. (2022) which applies the crossover and mutation operators with probabilities $\frac{50+q_1}{100+q_1+q_2}$ and $\frac{50+q_2}{100+q_1+q_2}$ respectively, where q_1 and q_2 denote the number of times an offspring solution by the crossover and mutation operator has been introduced into the population. Subsequently, the generated offspring is improved by the solution base tabu search procedure (Section 4.3), followed by a probability updating procedure to update the learning matrix (Section 4.4), and a population updating strategy to update the population (Section 4.7). Throughout the search process, a population replacement strategy is triggered to produce a new population if the best-found solution S_{best} fails to be improved for 30 consecutive generations (Section 4.8) to avoid premature convergence. The whole HEA procedure terminates when a fixed maximum number of generations is reached. A more detailed description of the algorithmic components is provided in the following subsections.

Algorithm 1 HEA for the COP

```
Input: a COP instance $G$; population size $p$; probability matrix $P$
Output: the best found solution $S_{best}$
 1: $POP = \{S_1, S_2, \ldots, S_p\} \leftarrow Initial\_Population(G)$
 2: $P \leftarrow Initial\_Probability\_Matrix()$
 3: for $i = \{1, 2, \ldots, p\}$ do
 4:     $S_i \leftarrow$ Solution_based_tabu_search($S_i$, $P$)
 5: end for
 6: $S_{best} \leftarrow Best(POP)$ /* record the best solution found so far */
 7: $q_1 \leftarrow 0$ /* record the number of times an offspring solution created with the crossover operator has
    been introduced into the population $POP$ */
 8: $q_2 \leftarrow 0$ /* record the number of times an offspring solution created with the mutation operator has
    been introduced into the population $POP$ */
 9: while stopping condition is not met do
10:     Generate a random number $\sigma \in [0, 1]$
11:     Randomly select two parents $S_1$ and $S_2$ from $POP$
12:     if $\sigma \leq \frac{50+q_1}{100+q_1+q_2}$ and $f(S_1) \neq f(S_2)$ then
13:         $S_0 \leftarrow$ Crossover($S_1$, $S_2$)
14:         $flag = 1$
15:     else
16:         $S_0 \leftarrow$ Mutate($S_1$)
17:         $flag = 2$
18:     end if
19:     $S \leftarrow$ Solution_based_tabu_search($S_0$, $P$) /* improve the offspring solution from crossover or
    mutation */
20:     $P \leftarrow$ Probability_updating($S_0$, $S$, $P$)
21:     if $f(S) \geq f(S_{best})$ then /* update the best solution found so far */
22:         $S_{best} \leftarrow S$
23:     end if
24:     $POP \leftarrow$ Population_update($POP$, $S$)
25:     if population updating is successful then
26:         if $flag = 1$ then
27:             $q_1 = q_1 + 1$
28:         else
29:             $q_2 = q_2 + 1$
30:         end if
31:     end if
32:     if $S_{best}$ is not improved for 30 consecutive generations then
33:         Apply population replacement strategy
34:     end if
35: end while
36: return $S_{best}$
```

A distinguishing feature of our solution based tabu search is that it uses the learned information stored in a probability matrix to guide the local search toward promising regions. Specifically, we adopt a probability matrix P of size $k \times 2$ (k is the number of clusters), where the element p_{i1} and p_{i2} ($i \in \{1, \ldots, k\}$) respectively denote the probability of retaining and removing a cluster C_i. When a cluster C_i is retained in the current solution, all customers in C_i are preserved, while if C_i is removed from the solution, only customers belonging to C_i but not belonging to any other cluster in the solution are dropped from the solution. Initially, all elements in P are set to $\frac{1}{2}$, meaning that all clusters are either retained or dropped with an equal probability. During the search process, the probability matrix P is used to guide the selection of the added or removed clusters (Section 4.3.2), and is updated via its probability learning procedure (Section 4.4).

4.2. *Population initialization*

The initial solutions of the population are constructed in a randomized manner similar to that in Angelelli et al. (2014). Initially, the initial solution S_{init} contains only the depot vertex, and all clusters are assigned to a candidate set L. Afterward, a cluster is picked from L randomly, and then added to the current solution S_{init}. Each time a cluster is added to S_{init}, the new tour of S_{init} is generated on all vertices included in S_{init} through the popular Lin-Kernighan algorithm (Lin and Kernighan, 1973). This process is repeated until no new cluster can be added to S_{init} without exceeding the time limit T_{max}. We repeat the construction procedure p times to fill the population with p solutions, where p is a parameter denoting population size. To obtain a high-quality population, each newly generated solution is further improved by SBTS (Section 4.3). As observed in studies like Hao (2012) and Zhou et al. (2022), a high-quality initial population enables the hybrid evolutionary algorithm to converge more quickly towards good final solutions. In Section 6.4, we demonstrate the benefit of the high-quality initial population to the performance of the hybrid evolutionary algorithm.

4.3. *Local optimization using solution-based tabu search*

The tabu search (TS) metaheuristic (Glover and Laguna, 1998) has been applied to many combinatorial optimization problems. Typically, TS explores the search space by iteratively transitioning from the current solution to one of its neighboring solutions. At each iteration, a best neighboring solution is sought to replace the current solution even if it does not improve the current solution. To prevent the search from revisiting a previously encountered solution, a tabu list strategy is applied. In the so-called solution-based TS (SBTS) (Woodruff and Zemel, 1993), the tabu list is implemented with hash functions and their associated hashing vectors. Unlike the classical attribute-based TS, SBTS records the visited solutions (instead of attributes) to avoid search cycling, thus overcoming the difficulty of tuning the parameters for tabu list management. In this work, we adopt for the first time SBTS for solving the COP and devise a multiple neighborhood SBTS reinforced by a reinforcement learning mechanism.

4.3.1. *Main scheme of the solution-based tabu search procedure*

Algorithm 2 summarizes the general scheme of our SBTS. After recording the best solution S_b found during the search of SBTS and initializing the associated hash vectors, SBTS performs a series of iterations. At each iteration, SBTS jointly explores two probabilistic neighborhoods $N_1(S)$ and $N_2(S)$ (Section 4.3.2) and selects the best feasible admissible neighboring solution from the neighborhoods $N_1(S)$ and $N_2(S)$ to replace the current solution. Then, the new current solution is marked as visited by updating the hash

Algorithm 2 Solution-based tabu search

```
Input: a solution $S$; maximum allowed consecutive iterations $T$ of no improvements; probability matrix $M$
Output: the best solution $S_b$
1:  $S_b \leftarrow S$ /* record the best solution $S_b$ found during tabu search */
2:  $(H_1, H_2, H_3) \leftarrow$ Initialize hash vectors
3:  $Noimprove \leftarrow 0$
4:  while $Noimprove \leq T$ do
5:      Construct the probabilistic neighbourhoods $N_1(S)$ and $N_2(S)$ /* Section 4.3.2 */
6:      Choose the best feasible non-tabu neighboring solution $S'$ in $N_1(S) \bigcup N_2(S)$
7:      $S \leftarrow S'$
8:      if $f(S) > f(S_b)$ then /* update the best solution found so far */
9:          $S_b \leftarrow S$
10:         $Noimprove \leftarrow 0$
11:     else
12:         $Noimprove \leftarrow Noimprove + 1$
13:     end if
14:     /* Update the hash vectors with $S$ */
15:     $H_1[h_1(S)] \leftarrow 1$
16:     $H_2[h_2(S)] \leftarrow 1$
17:     $H_3[h_3(S)] \leftarrow 1$
18: end while
19: return $S_b$
```

vectors (Section 4.3.3). The best-found solution S_b is updated if an improved solution has been found. If no improvement with respect to S_b is achieved during T consecutive iterations, then the best-found solution S_b is returned as the final output of SBTS.

As shown in Section 4.3.2, the neighborhoods N_1 and N_2 are induced by the Add and Drop move operators respectively. The Add operator can improve the quality of the current solution, while the Drop operator always decreases the objective value. However, if all neighboring solutions in N_1 are forbidden to access, or if adding any cluster causes the time budget to run out (i.e., the neighboring solutions in N_1 are infeasible), we switch to examining the neighboring solutions in N_2. To sum, we perform the Add operator whenever it is possible, and then the Drop operator when the Add operator cannot be applied.

4.3.2. *Move operators and probabilistic neighborhoods*

Given a solution S, let $K(S)$ be the set of clusters in S and $\overline{K}(S)$ be the set of remaining clusters. The proposed SBTS explores two neighborhoods $N_1(S)$ and $N_2(S)$ induced by the following basic move operators:

- **Add**: add a cluster $C_i \in \overline{K}(S)$ in S if the corresponding solution is feasible.
- **Drop**: drop a cluster $C_i \in K(S)$ from S.

After adding a cluster into the current solution S, the corresponding tour of S is generated by running the Lin-Kernighan algorithm to solve the corresponding TSP including the depot and all vertices belonging to S. Dropping a cluster always leads to a feasible solution and does not require any TSP calculation. Note that, when dropping a cluster C_i from S, we remove vertices only belonging to C_i but not belonging to any other cluster in $K(S)$.

We can see that an Add move always improves the current solution, whereas a Drop move always deteriorates the current solution. However, Add moves may become unavailable during the search because the insertion of any cluster in $\overline{K}(S)$ may make the travel time budget constraint violated, in this case, the algorithm resorts to the Drop moves to remove some clusters in $K(S)$, after which the Add moves can become available again.

Given that the examination of each neighboring solution consists of an application of Lin-Kernighan algorithm to determine the corresponding TSP tour for the served customers in the selected clusters, the computational cost to evaluate the neighborhood could be highly expensive. To reduce the computing time required to examine all neighboring solutions and improve the computational efficiency of our search procedure, we devise a probabilistic neighborhood which uses a reinforcement learning strategy to exclude some unpromising neighboring solutions. As mentioned in Section 4.1, the learned information is stored in a probability matrix P of size $k \times 2$, where k is the number of clusters, the element p_{i1} $(i \in \{1, \ldots, k\})$ denotes the probability that cluster C_i is retained in the current solution, and p_{i2} indicates the probability that cluster C_i is dropped from the current solution. Specifically, the chance of adding or dropping a cluster is proportional to the probability p_{ij} $(j = 1,\ 2)$, that is, cluster C_i is added to the current solution with the probability of $\frac{p_{i1}}{p_{i1}+p_{i2}}$ and dropped with the probability of $\frac{p_{i2}}{p_{i1}+p_{i2}}$.

Formally, the probabilistic add neighborhood N_1 of the current solution S can be defined as

$$N_1(S) = \{S' : S \oplus Add(C_i), C_i \in \overline{K}(S), rand(0,1) < \frac{p_{i1}}{p_{i1} + p_{i2}}\} \tag{10}$$

where $rand(0,1)$ denotes a random real number in $(0,1)$ and $S \oplus Add(C_i)$ is the neighboring solution obtained by adding C_i to S.

Similarly, the probabilistic drop neighborhood N_2 can be defined as

$$N_2(S) = \{S' : S \oplus Drop(C_i), C_i \in K(S), rand(0,1) < \frac{p_{i2}}{p_{i1} + p_{i2}}\} \tag{11}$$

where $S \oplus Drop(C_i)$ is the neighboring solution obtained by dropping C_i from S.

4.3.3. *Tabu list management strategy using hash functions*

Our SBTS operates in the cluster level and relies on a solution based tabu strategy to examine carefully in the search space. The basic idea behind the solution based tabu strategy is that it accurately mark each visited solution by memory and determines whether a neighbor solution has been previously visited by comparing it with each visited solution. As the number of solutions visited in the search history is typically immense, it is thus impractical to maintain a pool of solutions visited in the search history and to compare each new solution with the solutions in the pool to check if it is a previously visited solution. To effectively record visited solutions and to quickly determine whether a solution has been visited, we use hashing technique coupled with a fast computation method to keep track of each previously visited solution by mapping a solution into an integer. However, in hashing technique, hashing conflicts can frequently occur as two different solutions may be mapped into a same integer. To counter this, we simultaneously utilize three coordinated hashing vectors to precisely record each visited solutions, which is shown to be very effective in reducing the hashing conflicts compared to the use of only a single hashing vector (Lai et al., 2018; Wang et al., 2017).

Precisely, given a solution $S = (z_1, \ldots, z_k)$ where $z_i = 1$ if cluster i is in the current solution, and $z_i = 0$ otherwise. Each cluster i is associated with three weights w_{ij} $(j = 1, 2, 3)$ precomputed by $w_{i1} = \lfloor i^{2.7} \rfloor$, $w_{i2} = \lfloor i^{2.8} \rfloor$, and $w_{i3} = \lfloor i^{2.9} \rfloor$. Then, the three hash functions $h_j(S)$ $(j = 1, 2, 3)$ are defined as

$$h_j(S) = \left(\sum_{i=1}^{k} w_{ij} z_i \right) \bmod L \tag{12}$$

where L denotes the length of the hashing vectors (each hash function is associated with a hashing vector, and in this study, L is set to 10^8).

Then given the current solution S and its hash value $h_j(x)$, the hash value of its neighboring solution S' induced by $Add(C_i)$ and $Drop(C_i)$ can be quickly calculated respectively by Equations 13 and 14.

$$h_j(S') = (h_j(S) + w_{ij} z_i) \bmod L \tag{13}$$

$$h_j(S') = (h_j(S) - w_{ij} z_i) \bmod L \tag{14}$$

In this way, we can quickly calculate the hash value of any candidate neighboring solution in $O(1)$.

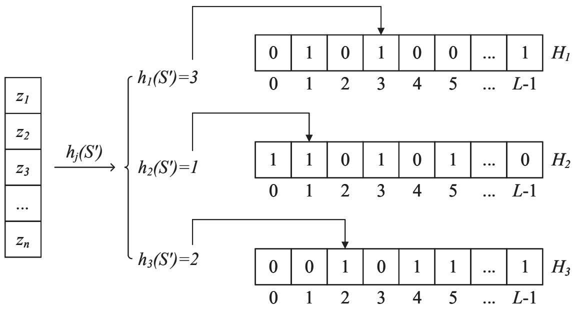


Fig. 2. Determining the tabu status of a solution using three hash functions and the associated hash vectors.

To determine if a solution has been previously visited, SBTS uses three hash vectors H_1, H_2, and H_3 which are initialized to 0, indicating that no solution is visited. At each iteration of SBTS, given the current solution S, for each candidate neighboring solution S' of S, S' is determined as a prohibited solution when the corresponding positions in H_1, H_2, and H_3 take the value of 1, i.e., $H_1[h_1(S')] = 1, H_2[h_2(S')] = 1, H_3[h_3(S')] = 1$. Otherwise, S' has not been visited previously and is qualified as an eligible neighboring solution. If S' is chosen to replace S, the corresponding positions in H_1, H_2, and H_3 are set to 1, i.e., $H_j[h_j(S')] \leftarrow 1, (j = 1, 2, 3)$. As long as collisions do not simultaneously occur in all three hash vectors, the tabu status is not misjudged, thereby significantly reducing the chance of collisions. An illustrative example is shown in Figure 2, where the neighboring solution S' is classified as tabu and thus cannot be used for solution transition.

Finally, we also mention that, contrary to the popular attribute-based tabu strategy which forbids the reverse move to be performed for a number of iterations called tabu tenure, the solution based tabu strategy is better suited for the tabu search in the context of solving the COP due to two advantages. First, it can avoid missing out any high-quality non-visited solution which may otherwise be mistakenly marked as tabu under the attribute-based tabu rule, thus enabling the SBTS make a more thorough examination in the neighborhood search. Second, the attribute-based tabu strategy can prevent the search from short-term cycling, but may be easily trapped in long-term cycling, visiting repeatedly previously examined solutions. Our solution based tabu strategy avoids such a situation by accurately keeping track of each previously visited solution, thus helping the algorithm not only escape from long-term cycling but also reduces significantly computational cost.

4.4. *Reinforcement learning*

As mentioned in Section 4.1, we adopt the probability matrix P to implement the reinforcement learning strategy. During SBTS, the probability of adding or dropping a cluster is determined by P. Recall that the chance of adding or dropping a cluster is proportional to the probability p_{ij}, with the add probability of $\frac{p_{i1}}{p_{i1}+p_{i2}}$ and drop probability of $\frac{p_{i2}}{p_{i1}+p_{i2}}$. This strategy compares the starting solution $S = (z_1, \ldots, z_k)$ and the improved solution $S' = (z'_1, \ldots, z'_k)$ with the cluster matching phase to check whether a cluster is kept or dropped from the improved solution. Afterward, the probability matrix P is updated according to the following rule.

Precisely, for each cluster C_i, if the state of C_i remains to be unchanged in the improved solution S', (i.e., $z'_i = z_i = u$, $u \in \{0, 1\}$), then we reward its original operation with C_i receiving state u and penalize the opposite operation with C_i receiving state $1 - u$ by adjusting the probabilities using the following equations:

$$p_{ij} = \begin{cases} \alpha + (1-\alpha)p_{ij} & j = u \\ (1-\alpha)p_{ij} & j = 1-u \end{cases} \tag{15}$$

where α is a reward factor and $0 < \alpha < 1$.

If the state of C_i is changed from $z_i = u$ in the initial solution S to $z_i = 1 - u$ in the improved solution S', then we penalize its former operation with C_i receiving state u and compensate for the new operation with C_i receiving state $1 - u$, then the probabilities are updated as

$$p_{ij} = \begin{cases} (1-\gamma)(1-\beta)p_{ij} & j = u \\ \gamma + (1-\gamma)\beta + (1-\gamma)(1-\beta)p_{ij} & j = 1-u \end{cases} \tag{16}$$

where β is the penalization factor, and γ is the compensation factor.

Note that in both formulas (15) and (16), $u = 1$ indicates the situation that C_i is selected to be included in the solution while $u = 0$ indicates that C_i is excluded from the solution.

With the help of learning schemes (15) and (16) applied iteratively, the probability for a cluster to receive a correct state is expected to increase gradually, thus guiding the search gradually towards promising areas. This probability updating scheme is inspired by the reinforcement learning mechanism proposed by Zhou et al. (2016, 2018); Sun et al. (2022). To prevent the search from being too greedy, we limit the probabilities in a suitable range. According to our experiments (Section 6.6), we choose the range $[0.2, 0.8]$.

4.5. *Crossover operator and parent selection*

Crossover operator is one of the crucial components of HEA. Generally, a successful crossover operator can inherit meaningful attributes from parent solutions and generate diversified offspring solutions (Hao, 2012). An analysis on a sample of locally optimal COP solutions discloses that high-quality solutions always share some common clusters (Section 6.8), thereby providing a strong motivation for preserving the common clusters from parent solutions to the offspring solution. Based on the observation above, we adopt a backbone-based crossover operator, where the "backbone" refers to the clusters shared in both parents.

The proposed backbone-based crossover constructs an offspring solution in two sequential stages.

Create a partial solution based on the backbone. First, the common clusters CM in two parent solutions S_1 and S_2 are identified, that is, $CM = \{C : C \in S_1, C \in S_2\}$. Second, the partial offspring S_o is created by solving the TSP defined by the depot and all vertices belonging to the common clusters CM using the Lin-Kernighan algorithm.

Complete the partial offspring in a random manner. If the duration of the tour of S_o does not exceed the maximum time limit T_{max}, then a random cluster that is not in S_o is selected and added to S_o from S_1 and S_2 in an alternating manner. Each time a cluster is added to S_o, the new tour of S_o is generated on all vertices included in S_o using the Lin-Kernighan algorithm. This process is repeated until no new cluster can be added to S_o (the duration of the new tour of S_o exceeds the time limit T_{max}).

In the literature, a crossover operator typically generates a single offspring solution (Zhou et al., 2020, 2022), two offspring solutions (Ayadi and Hao, 2014; Lu et al., 2020), or multiple offspring solutions (He and Hao, 2022). In this work, the backbone-based crossover operator is constrained to produce only a single offspring solution at each generation of the algorithm. We have also carried out experiments to test several other schemes where two or more offspring solutions are generated by the crossover. We have observed that the similarity between the offspring solutions is very high when two or more offspring solutions are generated. Therefore, we adopt in this work the crossover scheme to generate only a single offspring solution.

As for the selection of parents, we adopt the simple random selection instead of other mechanisms such as the roulette wheel selection (Lipowski and Lipowska, 2012) and the tournament selection (Coello and Montes, 2002). According to our experiments (Section 6.7), the simple random selection, the roulette wheel selection, and the tournament selection lead to similar performances of the HEA algorithm. Because of its simplicity, we use the random selection as the parent selection strategy in our algorithm.

4.6. *Mutation operator*

The mutation operator is a destroy-and-repair procedure. Given a solution S, the operator performs the following steps:

(1) Randomly remove $\lfloor \mu \times k \rfloor$ clusters from the current solution S, where μ is a parameter called mutation strength that determines the ratio of clusters to be removed, and k is the number of clusters in S.

(2) Randomly add a cluster into S, and then generate the new tour of S on all vertices included in S through the Lin-Kernighan algorithm. This process is repeated until no new cluster can be added to S without violating the time limit constraint.

4.7. *Population update strategy*

After an offspring solution S_o is generated by the crossover/mutation operator and then improved by SBTS, it replaces the worst solution S_w in the population if S_o is different from any existing solution in the population and $f(S_o) > f(S_w)$. Otherwise, S_o is abandoned.

4.8. *Population replacement strategy*

As shown in the previous section, the offspring solution S_o is introduced to the population regardless of its similarity to other individuals. For two given solutions S_1 and S_2 with their corresponding cluster sets C_1 and C_2, their similarity is defined by $Sim(S_1, S_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$. To avoid premature convergence and maintain a healthily diversified population throughout the search, a population replacement strategy is applied if the best-found solution S_{best} fails to be improved for 30 consecutive generations (Algorithm 1).

Specifically, we apply the construction procedure (Section 4.2) to generate a new solution S_{new} and then use SBTS (Section 4.3) to improve this solution. To ensure the population diversity, the improved solution S_{new} should be different enough to the best-found solution S_{best}. If the similarity is less or equal than 0.5 (i.e., $Sim(S_{new}, S_{best}) \leq 0.5$), then S_{new} replaces a randomly selected individual in the population (except the best solution of the population); otherwise, S_{new} is discarded. This process is repeated until 50% of the solutions in the population are replaced.

This population replacement strategy prevents the search from prematurely converging and being trapped into the deep local optimum.

4.9. *Discussion on the innovations of the proposed algorithm*

Compared with the existing OP algorithms in the literature, our proposed algorithm for solving COP includes mainly three original features.

First, the cluster selection subproblem is critical for our two-level approach. To avoid missing high-quality solutions in the cluster level, we adopt for the first time the solution-based tabu search approach (instead of the attribute-based tabu search approach), which uses the hashing technique to precisely record each visited solution and accurately determine the tabu status of neighboring solutions.

Second, as the evaluation cost for each neighboring solution is very computationally expensive due of the application of the Lin-Kernighan heuristic at the customer level, to improve the computational efficiency, we devise a reinforcement learning strategy which gathers useful historical information from visited local optima to guide the search to focus on these promising solutions and omit the examination of non-promising candidate solutions, thus significantly accelerating the neighborhood search.

Third, an analysis on a sample of locally optimal COP solutions discloses that high-quality solutions always share many common clusters (see Section 6.8), thereby providing a strong motivation for preserving the common clusters from parent solutions to the offspring solution. Based on the observation above, we devise a backbone-based crossover operator for offspring solution generation by inheriting the "backbone" refers to the clusters shared in both parents to guide the search towards promising search regions.

As shown in the next section, our proposed algorithm integrating these features performs very well on a number of COP benchmark instances commonly used in the literature. Finally, given that the ideas of the solution-based tabu search, the reinforcement learning strategy, and the backbone-based crossover are very general, they can be adapted to other related OP problems, such as the SOP (Archetti et al., 2018) and CTOP (Yahiaoui et al., 2019).

5. Computational and comparative results

In this section, we assess the performance of HEA over the benchmark instances and provide the comparative results with the state-of-the-art heuristics for the COP.

5.1. *Benchmark instances and experimental settings*

Our computational experiments are tested on two different problem set. The first set (Set A) contains 924 benchmark instances[1] which is generated from Angelelli et al. (2014) and used by Yahiaoui et al. (2019). The benchmark is derived from 57 instances of TSPLIB with 42 to 532 vertices. For each instance, the first vertex is set as the depot, and the remaining parameters are generated as follows:

(i) Clusters: the number of clusters takes the value of 10, 15, 20, or 25. Three additional values (50, 75, and 100) are considered for the largest instance (att532).

(ii) Profits: the profit of a cluster is calculated as the sum of the profits of its customers. The profit of a given customer is generated in two ways: in the first pattern, the profit of each customer is set to 1, whereas in the second pattern, the profit is generated by the formula $1 + (7141j + 73) mod(100)$, where j is the index of the vertex.

(iii) T_{max}: $T_{max} = \theta \times TSP^*$, where TSP^* is the optimal value of TSP over all vertices of a given instance, and θ takes the values of $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{3}{4}$.

For more details on generating instances, readers can refer to Angelelli et al. (2014).

To further assess the scalability of our proposed HEA algorithm, we additionally generated a new set (Set B) of 72 instances[2] with a larger number of clusters (50 to 100 clusters) following Angelelli et al. (2014). These instances are derived from 6 TSPLIB instances with 400 to 493 vertices. The results of HEA on this new data set can be useful for future work to evaluate the performance of new COP algorithms.

[1] The instances are available at http://or-brescia.unibs.it/.
[2] The new large instances are available at https://github.com/muhe2020/COP.

Table 1
Parameter settings.

Parameter	Section	Description	Considered values	Final value
p	4.2	Population size	{5, 10, 20}	5
T	4.3	Max allowed SBTS iterations without improvement	{5, 10, 20}	20
μ	4.6	Mutation strength	{0.2, 0.3, 0.5}	0.3
α	4.4	Reward factor	{0.10, 0.15, 0.20, 0.25, 0.30}	0.10
β	4.4	Penalization factor	{0.10, 0.15, 0.20, 0.25, 0.30}	0.20
γ	4.4	Compensation factor	{0.10, 0.15, 0.20, 0.25, 0.30}	0.30

The HEA algorithm is coded in C++ and executed on a computer with an Intel Core i5-8400 CPU with 2.80 GHz and 16 GB RAM. The parameter settings are given in Table 1. All parameters are tuned by the popular iterated racing method (F-race) (Birattari et al., 2010) to automatically determine the required parameters from a finite pool of parameter configurations. The tuning is performed on 15 randomly selected instances. The tuning budget was set to be 1000 executions of HEA. The setting of the parameters recommended by F-race is shown under the "Final value" column in Table 1 and used for our experiments.

5.2. *Computational and comparative results*

Given the stochastic nature of HEA, we perform 10 independent runs per instance, each run being limited to 50 generations (stopping condition). To assess the performance of HEA, we compare its results with those of the following state-of-the-art COP heuristics from the literature:

- Three variants of TS from Angelelli et al. (2014) called COP-TABU-Basic (CTB), COP-TABU-Multistart (CTM), and COP-TABU-Reactive (CTR), which adopt the traditional tabu strategy, tabu search with multistart strategy, and dynamic tabu tenure strategy, respectively.
- A hybrid heuristic (HH) (Yahiaoui et al., 2019) that combines an adaptive large neighborhood search with an effective problem-specific split procedure.

Table 2
Reference algorithms for COP.

Name	Reference	Experimental environment	Stopping condition
CTB	Angelelli et al. (2014)	Intel Xeon W3680 6-core CPU 3.33 GHz with 12 GB RAM	1000 iterations
CTM	Angelelli et al. (2014)	Intel Xeon W3680 6-core CPU 3.33 GHz with 12 GB RAM	999 iterations
CTR	Angelelli et al. (2014)	Intel Xeon W3680 6-core CPU 3.33 GHz with 12 GB RAM	1000 iterations
HH	Yahiaoui et al. (2019)	Intel Xeon E2-2670 16-core CPU 2.60 GHz with 128 GB RAM	$log(n \cdot k)$ iterations without improvement or n iterations
HEA	This work	Intel Core i5-8400 6-core CPU 2.80 GHz with 16 GB RAM	50 generations

Note that the reference algorithms were tested on different computing platforms. According to the SETI@home experiment website[3], our computing platform is the slowest with the lowest score in terms of the peak CPU speed (The scores of the computing platforms for running these comparative algorithms are respectively 6.00 (HEA), 12.00 (CTB), 12.00 (CTM), 12.00 (CTR) and 28.92 (HH)). These four COP heuristics, including their references, experimental environments, and stopping conditions, are listed in Table 2.

[3] Detailed CPU performance for all CPU model are available at https://setiathome.berkeley.edu/cpu_list.php.

5.2.1. *Computational results on Set A*

Tables 3 summarizes the results of the five algorithms on all 57 classes of 924 benchmark instances. Row '#Best' indicates the number of cases for which the best-known solution has been reached or improved, whereas row 'Dev.(%)' indicates the average percentage gap to the best-known solution. For each instance, the gap between the best-known result f_{bk} and the best result produced by the current algorithm f_{best} is calculated as $100 \times (f_{bk} - f_{best})/f_{bk}$. Row 'T(s)' provides the average total computation time of all instances in each class. Detailed computational results for each instance are available online at the link of footnote 2.

Table 3 shows that HEA outperforms all other heuristics on all instances. In particular,

Table 3
Performance of HEA on Set A.

Class	CTB			CTM			CTR			HH			HEA		
	#Best	Dev.(%)	T(s)	#Best	Dev.(%)	T(s)	#Best	Dev.(%)	T(s)	#Best	Dev.(%)	T(s)	#Best	Dev.(%)	T(s)
dantzig42	16	0.000	13.27	16	0.000	17.77	16	0.000	38.95	16	0.000	9.12	16	0.000	5.58
swiss42	13	0.670	15.38	14	0.275	23.09	15	0.013	31.93	16	0.000	6.70	16	0.000	4.39
att48	16	0.000	18.24	16	0.000	26.08	15	0.061	38.76	16	0.000	22.42	16	0.000	4.25
gr48	11	4.555	13.74	12	2.648	26.02	16	0.000	37.96	16	0.000	9.54	16	0.000	4.91
hk48	15	1.042	20.58	16	0.000	30.76	15	0.300	37.68	16	0.000	12.87	16	0.000	4.29
eil51	11	2.006	15.92	11	2.006	24.46	15	0.233	36.83	14	0.318	10.61	16	0.000	6.82
berlin52	15	0.504	38.88	15	0.118	53.39	15	0.118	60.41	16	0.000	29.50	16	0.000	6.18
brazil58	13	0.540	58.99	14	0.114	75.72	16	0.000	83.97	16	0.000	49.60	16	0.000	10.18
st70	11	1.225	23.18	11	0.973	38.95	12	0.622	48.01	16	0.000	18.00	16	0.000	10.33
eil76	9	5.190	24.50	10	3.429	33.74	15	0.123	45.84	16	0.000	12.18	16	0.000	12.29
pr76	11	0.959	21.40	13	0.105	30.88	15	0.009	54.76	16	0.000	29.94	16	0.000	20.42
gr96	12	0.572	44.07	13	0.115	51.35	14	0.024	68.19	16	0.000	31.46	16	0.000	18.88
rat99	12	1.399	32.99	12	0.126	52.03	15	0.033	63.65	15	0.147	34.02	16	0.000	19.94
kroA100	11	4.107	44.65	14	0.122	50.98	14	0.409	52.62	15	0.081	22.03	16	0.000	9.47
kroB100	15	0.641	47.96	16	0.000	58.94	16	0.000	62.20	16	0.000	21.02	16	0.000	12.55
kroC100	10	3.186	37.55	15	0.258	48.74	14	0.427	59.42	16	0.000	23.73	16	0.000	9.18
kroD100	10	1.762	36.85	11	1.182	56.70	13	0.495	69.57	16	0.000	32.41	16	0.000	11.22
kroE100	12	2.316	46.59	12	1.275	48.83	14	0.260	62.77	16	0.000	21.97	16	0.000	10.03
rd100	12	1.329	36.51	13	0.953	47.81	15	0.521	82.29	16	0.000	28.31	16	0.000	13.87
eil101	7	2.277	32.97	12	0.691	44.62	16	0.000	79.00	16	0.000	32.81	16	0.000	27.11
lin105	11	1.307	36.06	13	0.446	52.48	14	0.337	105.21	16	0.000	68.60	16	0.000	16.35
pr107	13	3.312	72.19	15	0.197	86.35	15	0.156	135.39	16	0.000	204.68	16	0.000	62.69
gr120	10	2.572	50.87	11	2.511	66.36	14	0.182	105.25	15	0.054	54.16	16	0.000	22.59
pr124	14	1.043	80.33	16	0.000	88.26	16	0.000	150.15	16	0.000	87.29	16	0.000	17.93
bier127	12	0.827	63.05	14	0.107	94.57	15	0.005	149.64	16	0.000	69.41	16	0.000	35.53
ch130	7	3.570	49.79	9	2.589	64.58	12	1.285	106.57	16	0.000	44.22	16	0.000	26.97
pr136	12	1.462	59.86	14	0.869	71.37	15	0.625	121.50	16	0.000	57.12	16	0.000	39.31
gr137	15	0.152	82.07	16	0.000	104.45	16	0.000	181.54	16	0.000	62.47	16	0.000	23.12
pr144	16	0.000	168.25	16	0.000	175.28	16	0.000	247.29	16	0.000	124.72	16	0.000	24.03
ch150	8	2.416	34.19	8	2.288	53.97	14	0.529	101.37	16	0.000	59.81	16	0.000	27.59
kroA150	9	0.934	36.84	13	0.225	50.60	14	0.073	102.11	15	0.046	54.30	16	0.000	21.88
kroB150	8	2.260	40.06	10	1.940	56.69	14	0.591	107.93	16	0.000	51.66	16	0.000	22.63
pr152	15	0.502	120.08	16	0.000	164.81	16	0.000	248.10	16	0.000	126.88	16	0.000	43.59
u159	6	3.066	113.36	9	2.198	125.51	8	1.385	184.68	15	0.269	162.81	16	0.000	47.15
si175	16	0.000	47.69	16	0.000	63.36	16	0.000	126.80	16	0.000	1022.91	16	0.000	76.34
brg180	12	0.608	54.29	13	0.531	72.18	15	0.089	127.74	16	0.000	605.49	16	0.000	52.62
rat195	12	0.511	68.18	10	0.208	78.52	14	0.378	172.00	16	0.000	162.70	16	0.000	96.53
d198	15	0.061	172.56	16	0.000	217.29	16	0.000	368.98	16	0.000	147.69	16	0.000	144.27
kroA200	11	0.999	55.09	12	0.964	76.17	14	0.951	139.43	16	0.000	100.08	16	0.000	43.94
kroB200	8	2.321	71.45	10	1.732	87.73	13	0.128	142.43	15	0.034	103.26	16	0.000	40.60
gr202	11	1.196	88.17	12	0.951	121.27	16	0.000	236.24	16	0.000	147.79	16	0.000	95.32
ts225	12	0.255	162.94	12	0.157	189.13	15	0.019	234.81	16	0.000	204.00	16	0.000	96.09
tsp225	9	1.403	87.78	9	0.481	102.99	11	0.141	180.26	16	0.000	191.03	16	0.000	106.12
pr226	12	0.784	244.84	12	0.713	268.33	15	0.042	331.10	16	0.000	314.10	16	0.000	58.57
gr229	15	0.023	109.99	15	0.023	121.07	15	0.023	170.85	16	0.000	96.69	16	0.000	128.97
gil262	6	6.649	57.09	6	3.915	84.20	10	2.256	135.48	14	0.050	183.41	16	0.000	85.80
pr264	11	3.171	151.96	10	3.183	208.51	14	0.309	304.70	16	0.000	256.35	16	0.000	131.34
a280	11	0.158	99.98	12	0.155	150.54	10	0.252	191.39	14	0.382	690.45	16	0.000	174.39
pr299	11	0.747	105.14	11	0.739	125.98	13	0.289	205.23	16	0.000	515.12	16	0.000	115.42
lin318	8	0.964	247.01	9	0.830	260.81	11	0.469	311.25	14	0.032	733.50	16	0.000	165.04
rd400	11	1.156	100.44	12	0.623	147.13	13	0.596	203.08	13	0.760	557.71	16	0.000	140.72
fl417	11	0.976	518.97	12	0.381	577.53	13	0.078	708.57	16	0.000	672.82	16	0.000	1022.07
gr431	12	0.740	236.75	15	0.009	252.35	16	0.000	280.53	16	0.000	429.84	16	0.000	391.80
pr439	11	0.628	180.16	13	0.074	221.23	14	0.058	324.17	15	0.002	531.14	16	0.000	279.34
pcb442	11	0.376	151.28	11	0.584	199.82	13	0.567	274.18	13	0.327	840.38	16	0.000	382.75
d493	7	1.102	418.35	9	1.139	419.66	12	0.996	515.84	16	0.000	949.26	16	0.000	549.07
att532	16	1.829	2180.37	18	1.522	2244.39	22	0.931	2332.69	19	0.454	3590.19	28	0.000	1941.70
Total	657	1.485	153.72	721	0.828	174.37	816	0.313	223.88	894	0.057	301.73	924	0.000	145.95

HEA improves the best-known results (new lower bounds) for 14 out of 924 instances and matches the best-known results for the remaining instances. In Table 4, we list the results for the 14 instances where our HEA improves the previous best-known results. The 'BKS' value of Table 4 show the best-known results compiled from the literature. Furthermore, HEA consistently finds the best-known solutions (BKS) in each run for most instances, thereby demonstrating its robustness. This algorithm also requires less computing time than the four reference algorithms on most instance sets, hence demonstrating its high competitiveness, in terms of both solution quality and computing time compared with state-of-the-art COP heuristics.

Table 4
New lower bounds for the 14 instances.

Instance	CTB	CTM	CTR	HH	BKS	HEA
kroA100s20g1q3	76	76	76	76	76	**77**
kroA150s25g2q3	5907	5907	5976	5997	5997	**6041**
lin318s15g2q2	7737	7760	7785	7787	7787	**7812**
lin318s25g2q3	13044	13083	13068	13191	13191	**13216**
rd400s20g2q3	13353	13353	13293	13374	13374	**13894**
rd400s25g2q3	12962	13010	12962	13546	13546	**13604**
pcb442s25g1q3	339	339	339	353	353	**355**
pcb442s25g2q3	17716	17103	17103	17103	17716	**17883**
att532s50g2q3	24195	24293	24353	24357	24357	**24362**
att532s75g1q3	500	501	518	518	518	**519**
att532s75g2q3	26223	26223	25897	26257	26257	**26362**
att532s100g2q2	18727	17649	16016	19996	19996	**20449**
att532s100g1q3	527	555	555	548	555	**562**
att532s100g2q3	25328	25651	27970	27905	27970	**28296**

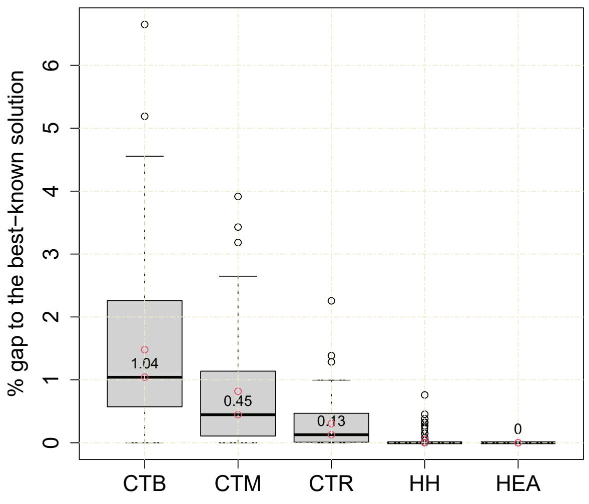


Fig. 3. Boxplots of the objective values on five algorithms.

The boxplot graphs in Figure 3 to compare the distribution and range of the average results obtained using the five algorithms. From this figure, we can observe that among compared heuristics, HEA obtains the smallest gap to the best-known solution, which equals 0 on all instances. This figure further highlights how HEA outperforms the recently proposed COP heuristics.

In sum, this comparative assessment confirms the effectiveness and robustness of HEA on different instance classes in terms of both solution quality and computational time.

5.2.2. *Computational results on Set B*

Table 5
Performance of HEA on Set B.

Instance	f_{best}	f_{avg}	$t_{avg}(s)$	Instance	f_{best}	f_{avg}	$t_{avg}(s)$
rd400s50g1q2	139	133.6	498.58	pr439s50g1q2	308	302.4	1100.03
rd400s50g1q3	300	297.9	1446.67	pr439s50g1q3	457	457.0	2366.27
rd400s50g2q2	6945	6875.0	424.81	pr439s50g2q2	15249	15092.9	918.79
rd400s50g2q3	15510	15097.3	1389.16	pr439s50g2q3	23155	23155.0	2302.19
rd400s75g1q2	152	148.7	914.89	pr439s75g1q2	352	341.6	2828.61
rd400s75g1q3	331	326.4	3150.78	pr439s75g1q3	496	496.0	8508.93
rd400s75g2q2	8319	8044.8	833.70	pr439s75g2q2	16616	16253.6	2571.02
rd400s75g2q3	16917	16514.3	2555.01	pr439s75g2q3	24984	22260.6	4860.22
rd400s100g1q2	180	175.0	1693.99	pr439s100g1q2	368	359.2	3601.64
rd400s100g1q3	372	369.6	6176.65	pr439s100g1q3	546	540.3	11482.45
rd400s100g2q2	8797	8607.8	1531.93	pr439s100g2q2	18900	18018.3	3157.90
rd400s100g2q3	18477	18206.9	3825.74	pr439s100g2q3	27185	26139.0	5879.60
fl417s50g1q2	219	219.0	3808.38	pcb442s50g1q2	262	248.0	1338.52
fl417s50g1q3	418	414.2	10428.72	pcb442s50g1q3	385	383.3	2865.50
fl417s50g2q2	10931	10931.0	3164.27	pcb442s50g2q2	12805	12044.3	1227.09
fl417s50g2q3	20923	20898.8	8631.26	pcb442s50g2q3	20200	19805.2	3074.04
fl417s75g1q2	262	262.0	3936.00	pcb442s75g1q2	280	277.6	2798.16
fl417s75g1q3	443	430.2	24117.45	pcb442s75g1q3	432	425.5	6491.69
fl417s75g2q2	13127	12773.8	3964.93	pcb442s75g2q2	13872	12526.0	1856.66
fl417s75g2q3	23064	22191.7	19858.29	pcb442s75g2q3	21238	20822.4	4627.13
fl417s100g1q2	284	279.4	5755.94	pcb442s100g1q2	304	304.0	6572.88
fl417s100g1q3	504	475.3	27421.32	pcb442s100g1q3	460	456.6	11366.28
fl417s100g2q2	14500	14482.9	8294.47	pcb442s100g2q2	15588	14870.2	4114.02
fl417s100g2q3	24644	23511.6	13599.34	pcb442s100g2q3	23527	22919.8	7694.41
gr431s50g1q2	395	387.4	2009.32	d493s50g1q2	344	344.0	3057.10
gr431s50g1q3	479	479.0	3745.20	d493s50g1q3	473	464.4	3794.96
gr431s50g2q2	19996	19886.5	2290.25	d493s50g2q2	17260	17206.6	2902.91
gr431s50g2q3	24226	24226.0	3793.28	d493s50g2q3	23231	22876.5	3577.71
gr431s75g1q2	433	430.0	5405.28	d493s75g1q2	332	316.8	3688.30
gr431s75g1q3	523	523.0	10880.92	d493s75g1q3	496	489.0	6647.96
gr431s75g2q2	22042	21923.2	4275.16	d493s75g2q2	18594	17950.5	4218.42
gr431s75g2q3	26382	26380.0	4053.22	d493s75g2q3	25992	25413.7	7347.46
gr431s100g1q2	477	470.4	10527.46	d493s100g1q2	399	399.0	12496.35
gr431s100g1q3	575	575.0	22569.80	d493s100g1q3	556	549.7	15024.25
gr431s100g2q2	23605	23192.1	8887.60	d493s100g2q2	19217	18451.9	6500.75
gr431s100g2q3	29094	28865.4	7742.63	d493s100g2q3	27947	27293.5	12347.89

We perform 10 independent runs on each instance of Set B, and the results for all 72 instances are presented in Table 5 in the following format: the instance name (instance), the best objective value (f_{best}), the average objective value (f_{avg}), and the average computation time in seconds ($t_{avg}(s)$). For detailed computational results for each instance of Set B, please refer to the online table at the link of footnote 2. The results of HEA on Set B can serve as references for future comparative studies of new COP methods.

To evaluate the performance of the proposed HEA algorithm on Set B, we compare the solutions obtained by the proposed HEA algorithm and the three TS variants of Angelelli et al. (2014) called COP-TABU-Basic (CTB), COP-TABU-Multistart (CTM), and COP-TABU-Reactive (CTR), which were specifically designed for the COP. Given that the source codes of the three TS algorithms are unavailable, we faithfully re-implemented them, and verified that our implementation was able to reproduce the results reported in Angelelli et al. (2014). Detailed computational results of the three re-implemented reference algorithms on Set B are also available online at the link of footnote 2.

Table 6
HEA vs. three TS variants of Angelelli et al. (2014) on the instances from Set B.

	Best results			Average results		
	Better	Equal	Worse	Better	Equal	Worse
HEA vs CTB	62	6	4	71	0	1
HEA vs CTM	67	3	2	71	0	1
HEA vs CTR	62	5	5	71	0	1

Table 6 indicates the number of times HEA reports a better, equal, or worse result, in terms of both the best and the average results, compared to each TS variant from Angelelli et al. (2014). We notice that HEA significantly dominates the three heuristic algorithms on Set B.

6. Analysis and discussion

This section analyzes the key components of HEA, namely, solution-based tabu strategy, reinforcement learning strategy, crossover and mutation operator, high-quality initial population, and population replacement strategy. We also reveal the rationale behind the proposed backbone-based crossover, analyze the range of the probability in reinforcement learning, the mechanism of parent selection, the sensitivity of the parameters, and assess the influence of the number of clusters on HEA's performance.

6.1. *Effect of the solution-based tabu strategy*

As shown in Section 4.3, SBTS records all the visited solutions and prevents them from being revisited during the following search process, thereby ensuring a stronger intensification search ability. To assess its benefit, we compare HEA with a variant called ABHEA, where SBTS is replaced by a traditional attributed-based TS. For ABHEA, each time a cluster is added in or dropped from the current solution, this cluster cannot be selected for the next tt (tabu tenure) iterations. We employ three different values of tt (tt = 5, 10, 15) to extensively evaluate the performance of ABHEA (denoted by ABHEA$_{tt}$ with tt = 5, 10, 15). We then evaluate HEA and ABHEA on 15 randomly selected large instances with at least 200 vertices from Set A under the same experimental conditions as before.

The experimental results are summarized in Table 7, where Columns 'f_{best}', 'f_{avg}', and '$t_{avg}(s)$' present the best objective value, the average objective value over 10 independent runs, and the average computation time in seconds across the 10 runs, respectively. Table 7 shows that HEA significantly outperforms ABHEA in terms of the best and average objective values. Furthermore, ABHEA requires much more time than HEA, and with a decreasing tt, the computational time of ABHEA gradually increases. These results confirm that the solution-based tabu strategy is suitable for determining the tabu status of solutions for COP.

Table 7
Comparison between HEA and ABHEA on 15 randomly selected instances with at least 200 vertices.

Instance	HEA			ABHEA_5			ABHEA_{10}			ABHEA_{15}		
	f_{best}	f_{avg}	$t_{avg}(s)$	f_{best}	f_{avg}	$t_{avg}(s)$	f_{best}	f_{avg}	$t_{avg}(s)$	f_{best}	f_{avg}	$t_{avg}(s)$
a280s25g2q2	7539	7386.4	179.94	7539	7176.1	704.61	7539	7149.0	661.65	7539	7274.4	651.25
att532s10g1q2	315	315.0	133.60	315	308.8	630.66	315	308.8	355.63	315	315.0	257.25
gil262s25g2q2	4473	4017.7	59.41	4473	4041.2	236.51	4473	4023.2	225.67	4473	3974.0	209.41
gr202s25g2q2	7525	7367.5	142.59	7525	7231.5	376.71	7230	7204.0	304.76	7525	7253.5	277.94
gr229s15g2q2	7858	7852.6	70.08	7858	7858.0	276.93	7858	7855.5	209.41	7858	7858.0	171.71
kroA200s20g2q3	6654	6553.2	83.25	6654	6231.4	324.93	6654	6354.1	319.51	6654	6286.8	271.05
kroA200s25g2q3	7170	7080.8	132.59	6885	6849.0	450.72	7170	6939.3	416.44	7090	6872.6	375.70
lin318s15g2q2	7812	7804.5	94.39	7812	7812.0	495.18	7812	7812.0	373.37	7812	7812.0	304.43
pcb442s20g2q2	11805	11131.5	299.81	11805	10802.1	1435.70	11805	10801.5	1151.02	11768	10798.4	1078.09
pr226s20g1q2	117	115.8	56.85	117	111.0	174.09	117	109.8	137.72	117	109.8	125.69
pr299s20g2q2	7704	7401.4	79.67	7704	7035.1	323.86	7704	7089.1	314.27	7704	7047.9	280.28
pr439s25g2q2	14028	14020.8	328.06	14028	13918.6	1279.08	14028	13919.0	1064.45	14028	14017.4	962.33
rd400s20g2q2	5675	5663.0	68.45	5675	5668.2	306.79	5675	5664.2	297.69	5675	5670.8	271.80
ts225s25g1q2	121	117.6	80.10	121	115.5	345.78	121	115.5	322.38	121	114.4	296.99
tsp225s20g2q2	5661	5639.1	77.55	5661	5631.4	319.72	5661	5646.2	289.09	5661	5638.8	274.65
Average	6297.13	6164.46	125.76	6278.13	6052.66	512.08	6277.47	6066.08	429.54	6289.33	6069.59	387.24

6.2. *Impact of the reinforcement learning strategy*

Table 8
Comparative results between HEA and HEA_0 on 15 different instances with at least 200 vertices.

Instance	HEA			HEA_0		
	f_{best}	f_{avg}	$t_{avg}(s)$	f_{best}	f_{avg}	$t_{avg}(s)$
a280s25g2q2	7539	7386.4	179.94	7539	7180.8	225.86
att532s10g1q2	315	315.0	133.60	315	315.0	146.47
gil262s25g2q2	4473	4017.7	59.41	4460	4013.9	76.63
gr202s25g2q2	7525	7367.5	142.59	7525	7308.5	159.24
gr229s15g2q2	7858	7852.6	70.08	7858	7852.6	87.97
kroA200s20g2q3	6654	6553.2	83.25	6634	6596.0	103.15
kroA200s25g2q3	7170	7080.8	132.59	7170	6922.8	166.06
lin318s15g2q2	7812	7804.5	94.39	7812	7812.0	113.01
pcb442s20g2q2	11805	11131.5	299.81	11768	11121.8	325.10
pr226s20g1q2	117	115.8	56.85	117	112.2	68.78
pr299s20g2q2	7704	7401.4	79.67	7704	7400.4	100.23
pr439s25g2q2	14028	14020.8	328.06	14028	13907.8	402.96
rd400s20g2q2	5675	5663.0	68.45	5675	5629.5	82.17
ts225s25g1q2	121	117.6	80.10	121	114.1	93.22
tsp225s20g2q2	5661	5639.1	77.55	5661	5631.6	88.48
Average	6297.13	6164.46	125.76	6292.47	6127.93	149.29
p-value				8.32e-02	1.26e-02	1.08e-04

To evaluate the impact of the reinforcement learning strategy in HEA, we compare this algorithm with its variant HEA_0, where the reinforcement learning strategy is removed. The experiment is carried out on the 15 large instances with at least 200 vertices used in Section 6.1. We conduct 10 independent runs for each version and record the best objective value (f_{best}), the average objective value (f_{avg}), and the average run time ($t_{avg}(s)$) in seconds. From the comparative results of HEA and HEA_0 of Table 8, we observe that HEA outperforms HEA_0 in terms of both the best objective values and the average objective values. The experiment demonstrates clearly the usefulness of the reinforcement learning strategy.

6.3. *Effect of the crossover/mutation operator*

As shown above, the solution-based tabu strategy and the reinforcement learning strategy make critical contributions to the performance of HEA. We now study the

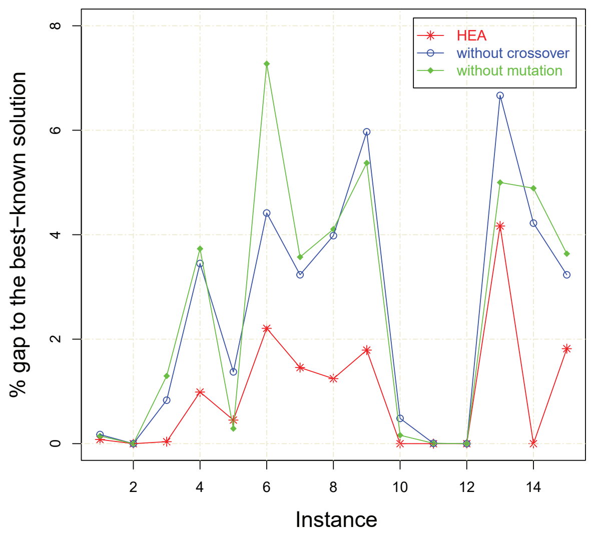


Fig. 4. Comparison between HEA and its two variants without the crossover and mutation operator.

impact of the crossover/mutation operator on our algorithm by comparing HEA with two variants. In the first variant, we disable the crossover operator (setting the random number (line 10 of Algorithm 1) to 1), while leaving the other components unchanged. Similarly, in the second variant, we disable the mutation operator (setting the random number (line 10 of Algorithm 1) to 0) and keeping the other components. We run HEA and its two variants 10 times independently on the 15 instances used above.

The average performance of these three versions is shown in Figure 4, where the y-axis represents the percentage gap between the updated best-known solution and the average result obtained by each algorithm. The gap of HEA is smaller than that of the other two algorithms, thereby highlighting the joint effect of the crossover/mutation operator.

6.4. *Benefit of the high-quality initial population*

As shown in Section 4.2, to obtain a high-quality initial population, each newly generated solution in the initial population is further improved by SBTS. To assess the benefit of the high-quality initial population on the performance of the proposed HEA algorithm, we compare HEA with an algorithmic variant (HEA_1) with a random population by disabling lines 3-5 in Algorithm 1. We run HEA and HEA_1 10 times independently on the 15 instances used above.

The average performance of two algorithms is shown in Figure 5, where the y-axis indicates the percentage gap between the best-known result and the average result obtained by each algorithm. The figure clearly shows that the performance of the HEA_1 variant with a random population is worse. This experiment thus confirms the usefulness of the high-quality initial population for the effectiveness of the HEA algorithm.

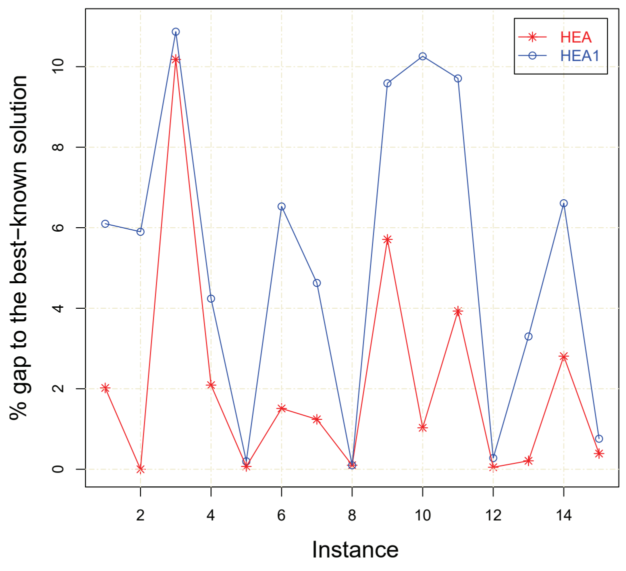


Fig. 5. Comparisons of HEA with its variant HEA$_1$ with a random population.

6.5. *Effectiveness of the population replacement strategy*

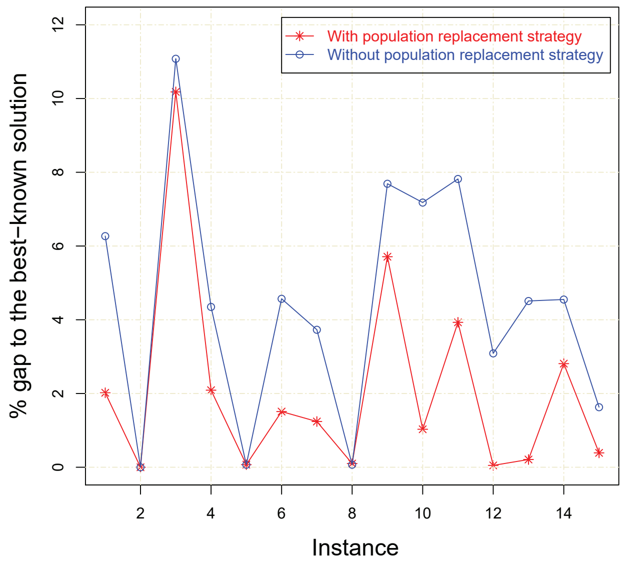


Fig. 6. Comparisons of two HEA versions with and without the population replacement strategy.

As shown in Section 4.1, if the best-found solution S_{best} fails to be improved for 30 consecutive generations, our HEA uses a population replacement strategy (Section 4.8) to produce a new population to prevent the search from prematurely converging and being trapped into deep local optima. To illustrate the effectiveness of the population replacement strategy to the HEA performance, we compare HEA with a weakened version of HEA that does not make use of the population replacement strategy by disabling lines 32-34 in Algorithm 1.

We run the two HEA versions with and without the population replacement strategy 10 times independently on the 15 selected instances used above. The average performance of the two versions is shown in Figure 6, where the y-axis indicates the percentage gap between the best-known result and the average result obtained by each algorithm. The figure clearly highlights the benefit of the population replacement strategy.

6.6. *Influences of the range of the probability in reinforcement learning*

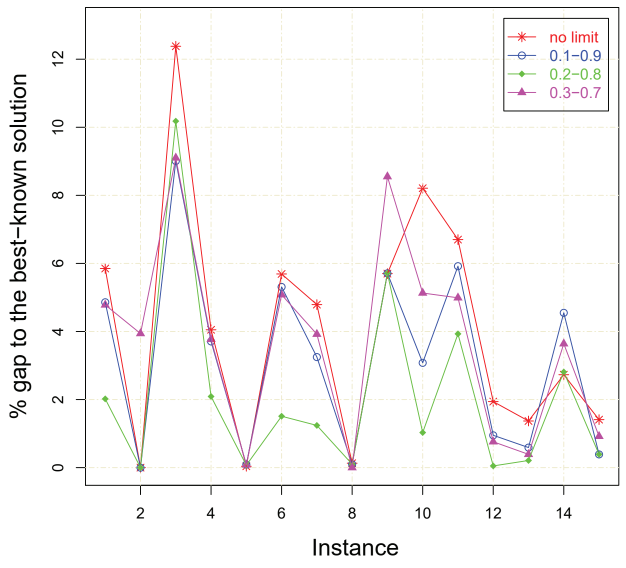


Fig. 7. Performance gaps of HEA with different ranges for the probability.

As shown in Section 4.4, we adopt the probability matrix to implement the reinforcement learning strategy, which is used to exclude some unpromising neighboring solutions. Since the probabilities are likely to continue updating to 0.0 or 1.0, which may result in a lack of diversity in the search, we limit the probabilities in a suitable range. To identify a suitable range for the probability, we tested three alternatives $[0.1, 0.9]$, $[0.2, 0.8]$, and $[0.3, 0.7]$. Besides, to check whether the range restriction on the probability will weaken the effect of the reinforcement learning strategy, we also test a HEA version where the range of the probability is not limited.

Figure 7 presents the average performance with the same information as before. From Figure 7, we observe that the probability in the range $[0.2, 0.8]$ produces the best performance, and when the probability has an unlimited range, it reports the worst performance. According to this experiment, we choose the range $[0.2, 0.8]$ as the default setting of the HEA algorithm.

6.7. *Impact of the selection of parents*

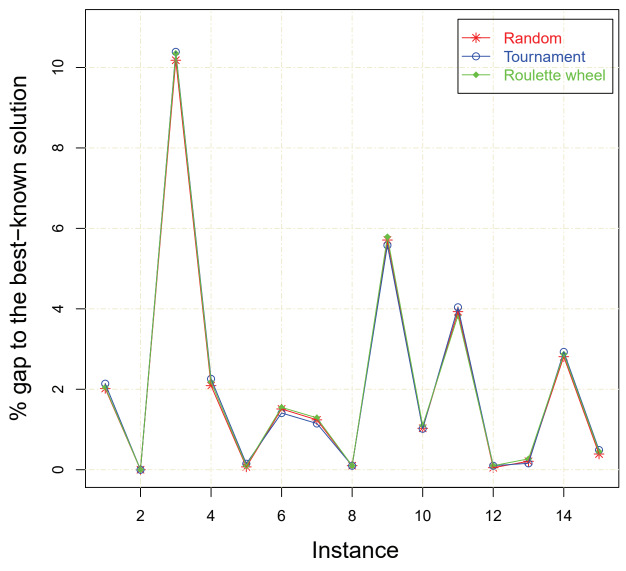


Fig. 8. Comparison of three parent selection mechanisms.

As for the selection of parents, we adopt the simple random selection. This is mainly due to two reasons. First, the local improvement procedure SBTS ensures a very high quality of the population solutions. Second, the mutation operator ensures a certain degree of population diversity. Also, the use of a very small population (5 in our case) tends to make different selection strategies behaving in a similar way. In order to verify if other mechanisms would be better, we compare HEA with two of its variants, which respectively uses the roulette wheel selection (Lipowski and Lipowska, 2012), and the tournament selection (Coello and Montes, 2002).

The average performance of the three algorithms are plotted in Figure 8 as before, where the y-axis indicates the average percentage gap from the updated best-known solution. This experiment indicates that the simple random selection, the roulette wheel selection, and the tournament selection do not significantly change HEA's performance. We adopt thus the simple random selection in our HEA algorithm.

6.8. *Rationale behind the backbone-based crossover*

To generate insights into the rationale behind the backbone-based crossover, we experimentally investigate the structural similarities between high-quality solutions. Recall that the similarity for two given solutions S_1 and S_2 is defined by $Sim(S_1, S_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$ in Section 4.8. Intuitively, a larger similarity between two solutions corresponds to more clusters they share, which is a favorable feature for the backbone-based crossover, where offspring solutions are allowed to inherit the common clusters that form the backbone of a high-quality solution.

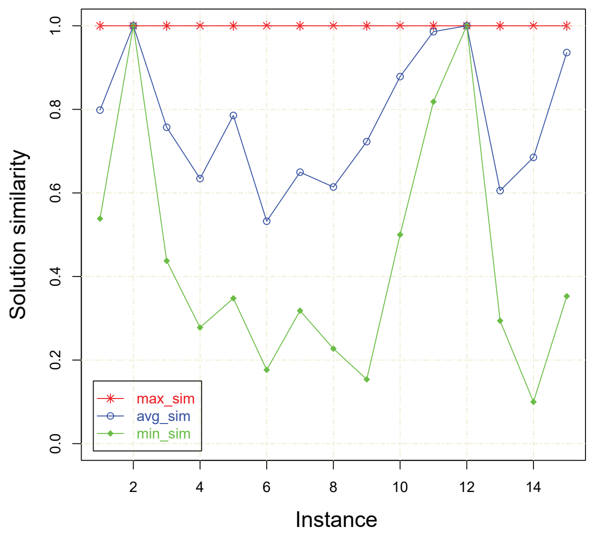


Fig. 9. Similarity between high-quality solutions.

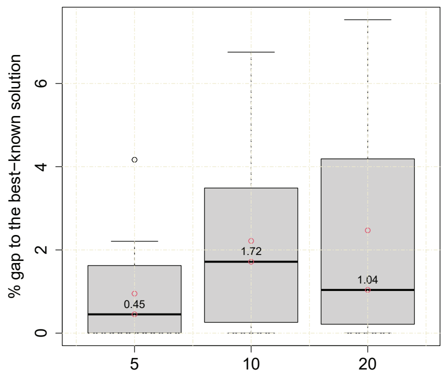
We run HEA 100 times on each of the 15 selected instances mentioned above and record the best solution found in each run. For each instance, we compute the maximum similarity (denoted by max_sim) between any two solutions by $max\_sim = \max_{1\leq i<j\leq 100} Sim(S_1, S_2)$, the average similarity (denoted by avg_sim) between any two solutions by $avg\_sim = \frac{1}{4950} \times \sum_{1\leq i<j\leq 100} Sim(S_1, S_2)$, and the minimum similarity (denoted by min_sim) between any two solutions by $min\_sim = \min_{1\leq i<j\leq 100} Sim(S_1, S_2)$. Figure 9 presents the solution similarities for the 15 selected instances.

Figure 9 shows a high degree of similarity between high-quality solutions. Specifically, for the 15 selected instances, the maximum similarity reaches 1.0 and the average similarity is greater than 0.5. These experimental results provide a solid foundation for the backbone-based crossover designed specifically for COP in our work.

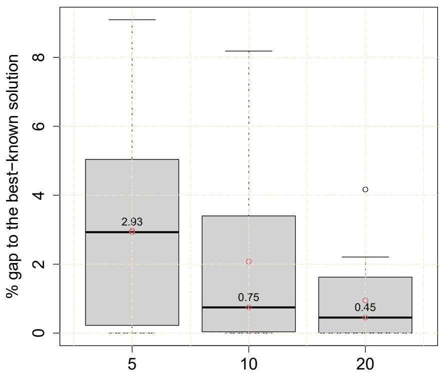
6.9. *Sensitivity analysis of parameters*

HEA requires 6 parameters as shown in Table 1, including population size (p), the maximum allowed SBTS iterations without improvement (T), the mutation strength (μ), the reward factor (α), the penalization factor (β), and the compensation factor (γ). To evaluate the sensitivity of each parameter, we test the considered values for each parameter from Table 1, while fixing the remaining parameters to their final values. For this analysis, we use the same selection of 15 instances as before and perform 10 independent runs for each considered value.

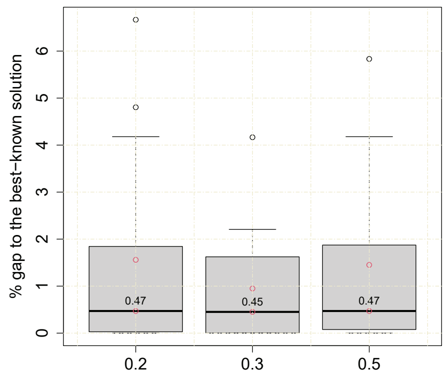
The distribution and ranges of the objective values are presented in the form of box-and-whisker plots (Figure 10). To determine whether different values of a given parameter show statistically significant differences in the samples, the Friedman rank sum test is performed. Results of the Friedman rank sum test indicate a significant difference in



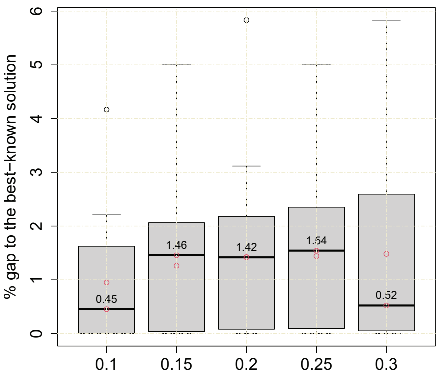
(a) p: Friedman p-value = 5.829e-05



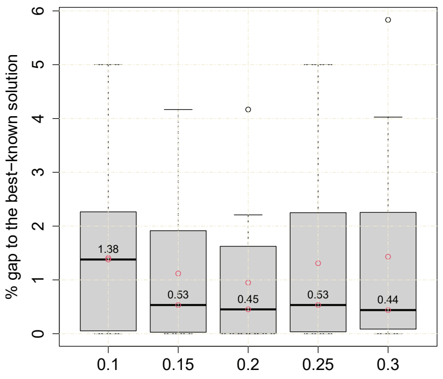
(b) T: Friedman p-value = 4.169e-06



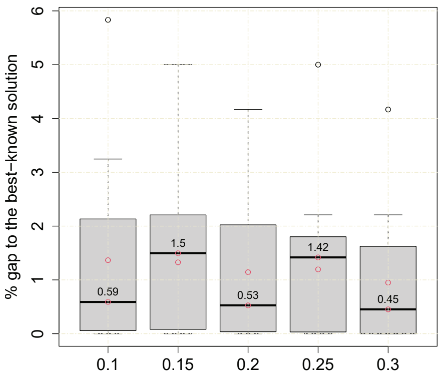
(c) μ: Friedman p-value = 1.135e-04



(d) α: Friedman p-value = 5.788e-06



(e) β: Friedman p-value = 2.743e-05



(f) γ: Friedman p-value = 5.334e-06

Fig. 10. Boxplots of the normalized average objective values for considered values of each parameter.

performance for p (p-value = 5.829e-05), T (p-value = 4.169e-06), μ (p-value = 1.135e-04), α (p-value = 5.788e-06), β (p-value = 2.743e-05), and γ (p-value = 5.334e-06). Figure 10 shows that the recommended parameter values from this calibration experiment are the same as those recommended by F-race.

6.10. *Influence of the cluster number k on the performance of HEA*

To assess the influence of the number of clusters on the performance of HEA, we further classified the instance classes according to the number of clusters. In Table 9, each entry shows the average computational time of each class respectively with $k = 10, 15, 20$, and 25.

Table 9
Computational time required by instance classes with different cluster size.

Class	$k = 10$	$k = 15$	$k = 20$	$k = 25$
dantzig42	1.75	3.65	6.77	10.13
swiss42	1.71	2.93	4.82	8.09
att48	1.56	3.27	4.90	7.27
gr48	1.73	3.03	5.55	9.32
hk48	1.52	2.95	4.83	7.85
eil51	2.57	4.56	7.99	12.15
berlin52	1.88	3.91	7.42	11.53
brazil58	2.65	5.92	10.83	21.34
st70	3.57	7.12	11.49	19.13
eil76	4.46	8.00	14.33	22.38
pr76	6.48	11.99	24.68	38.54
gr96	7.81	12.47	22.05	33.21
rat99	7.71	14.57	22.71	34.78
kroA100	3.39	6.11	12.66	15.73
kroB100	4.73	7.84	14.27	23.38
kroC100	3.14	6.50	10.46	16.60
kroD100	3.80	7.13	12.85	21.11
kroE100	3.78	7.21	11.22	17.92
rd100	4.84	9.12	16.54	25.00
eil101	9.61	16.43	32.98	49.43
lin105	6.27	12.05	18.97	28.12
pr107	21.38	41.61	75.45	112.31
gr120	7.39	13.91	25.07	43.97
pr124	5.65	12.52	20.20	33.33
bier127	12.83	24.07	42.84	62.39
ch130	10.01	18.98	31.93	46.96
pr136	15.92	27.01	43.55	70.77
gr137	9.34	15.88	26.22	41.01
pr144	9.76	19.42	27.62	39.32
ch150	10.53	17.94	32.65	49.25
kroA150	8.98	15.68	26.69	36.15
kroB150	9.99	16.27	24.39	39.88
pr152	17.21	31.76	49.83	75.57
u159	14.94	34.97	56.31	82.40
si175	27.17	51.60	90.45	136.13
brg180	14.87	26.99	71.03	97.60
rat195	46.07	79.27	106.25	154.53
d198	69.77	92.77	165.36	249.20
kroA200	18.28	30.54	48.38	78.56
kroB200	18.35	26.27	48.26	69.54
gr202	29.00	66.15	105.57	180.56
ts225	36.81	75.54	110.45	161.56
tsp225	42.70	73.67	127.03	181.10
pr226	24.99	46.39	69.95	92.96
gr229	42.20	86.70	141.71	245.29
gil262	37.96	64.23	105.46	135.57
pr264	47.16	94.83	151.35	232.01
a280	70.67	128.44	185.87	312.58
pr299	44.95	85.80	132.13	198.82
lin318	57.69	138.10	203.83	260.55
rd400	57.39	95.74	157.98	251.76
fl417	369.79	705.76	1066.41	1946.35
gr431	160.64	268.08	465.94	672.54
pr439	107.35	195.41	323.24	491.34
pcb442	183.64	288.22	460.72	598.41
d493	221.60	423.69	634.53	916.48
att532	151.77	263.12	407.89	581.97

As shown in Table 9, the computational time required by HEA has a positive relationship with the number of clusters over all classes, and an instance with a larger number of clusters generally requires more computational time. This is because a larger number of clusters will lead to more neighboring solutions in the neighborhood induced by the add/drop move operators consisting in adding/dropping a cluster, thereby increasing the evaluation time to examine the neighborhood. Figure 11 summarizes the computational time in seconds required by HEA on several instance classes with different cluster sizes, which also demonstrates that the computational time tends to increase when the cluster size becomes larger.

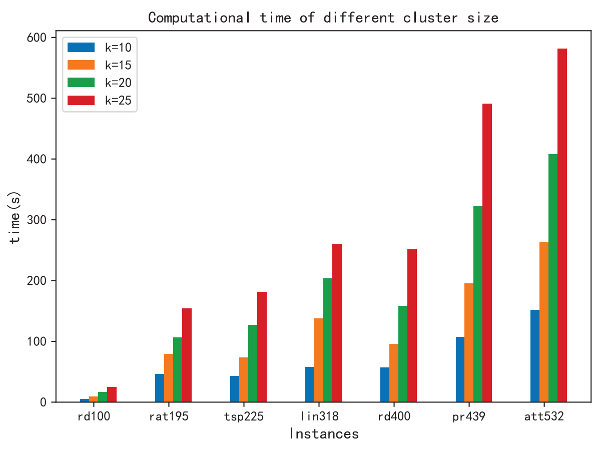


Fig. 11. Computational time on several instance classes with different cluster size.

7. A dynamic version of COP considering stochastic travel time

The OPs have a long history in the literature, and most existing studies in the literature focus on the static versions of OPs. However, the travel time may fluctuate under changing traffic conditions, and the assumption of deterministic travel time is unrealistic in many real-world settings. To account for transportation network and infrastructure uncertainty, OPs with stochastic travel time have been considered in the literature. For instance, Campbell et al. (2011) studied a variant of the OP in which the travel and service time are stochastic. Lau et al. (2012) introduced an extension of OP with dynamic travel time, and then proposed a local search algorithm to solve it. Evers et al. (2014) considered the OP with stochastic weights to reflect uncertainty in real-life applications, where the stochastic weights are associated with travel costs, travel time or fuel consumption on arcs. Dolinskaya et al. (2018) studied a novel OP with stochastic travel time by adapting paths between reward nodes as travel times. Wang et al. (2023) extended the classical OP to a dynamic OP with stochastic travel time and service time, and proposed three self-adaptive heuristic algorithms for solving the dynamic OP.

In this section, we study a dynamic version of COP, which considers stochastic travel time between nodes. In this dynamic version, we assume that the travel time between any two different nodes follows a normal distribution. To solve it, we follow the work of Irawan et al. (2021) and propose a simulation-based optimization method integrating HEA proposed in this work and Monte Carlo simulation (Zio and Zio, 2013).

As shown in Algorithm 3, the simulation-based optimization method performs Λ iterations, where each iteration consists of two stages. In the first stage, we use our HEA algorithm to solve the deterministic COP problem, where the stochastic travel time is transformed into a deterministic travel time. In the second stage, after the tour S is generated by the HEA algorithm, Monte Carlo simulation is called to estimate the expected total travel time of the tour S. Monte Carlo simulation is an iterative process, where in each iteration, we generate random numbers conforming to their normal distribution to represent the stochastic travel time. If the expected total travel time of the tour S estimated by Monte Carlo simulation does not exceed the maximum time limit T_{max} for more than $\lambda \times \gamma\%$ times (where λ is number of Monte Carlo simulations, and γ is the threshold of accepting a solution), we consider the solution S is feasible. Finally, the best found feasible solution S_{best} is updated with S if S is feasible and $f(S) > f(S_{best})$. At the end of the proposed simulation-based optimization method, the best found feasible solution S_{best} is returned as the final output.

Algorithm 3 The proposed simulation-based optimization method for a dynamic version of COP considering stochastic travel time

```
Input: a COP instance $G$; maximum time limit $T_{max}$; number of iterations $\Lambda$; number of Monte Carlo
simulations $\lambda$; threshold of accepting the solution $\gamma$
Output: the best found solution $S_{best}$
 1: for $\alpha = 1$ to $\Lambda$ do
 2:     /* Stage 1: Solve the deterministic problem */
 3:     Generate the travel time for each edge randomly conforming to their distribution for the COP
        instance $G$
 4:     $S \leftarrow$ HEA($G$) /* Use the HEA to solve the COP with the deterministic travel time */
 5:     /* Stage 2: Run Monte Carlo simulation $\lambda$ times */
 6:     $\theta \leftarrow 0$
 7:     for $\beta = 1$ to $\lambda$ do
 8:         Generate the travel time for each edge randomly conforming to their distribution
 9:         Compute the total travel time of the tour $S$, i.e., $T(S)$
10:         if $T(S) \leq T_{max}$ then
11:             $\theta \leftarrow \theta + 1$
12:         end if
13:     end for
14:     if $\theta \geq \lambda \times \gamma\%$ then
15:         if $f(S) > f(S_{best})$ then /* The tour $S$ has a high chance to be completed within the maximum
        time limit, and is considered feasible */
16:             $S_{best} \leftarrow S$
17:         end if
18:     end if
19: end for
20: return $S_{best}$
```

To demonstrate the practical usefulness of the proposed simulation-based optimization method, we apply it to deal with a practical route planning case using real data collected from an intra-city delivery platform in Wuhan city, China. In recent years, with the booming of the online orders, several intra-city retail platforms such as Meituan arise, where people can place their orders with mobile apps for a variety of commodities distributed in different sites. The intra-city retail is supported by an intra-city delivery platform as the underlying infrastructure, where all sites of suppliers providing commodities for customers, namely the points of pickups (POPs), constitute the intra-city delivery network. The real dataset (see Figure 12), including the location of POPs in Wuhan and traveling time between these POPs, is provided by the Meituan delivery platform. In real-life applications, the travel time between POPs may fluctuate due to changing traffic conditions. Thus, we assume that the travel time follows a normal distribution with mean μ equals to the time during off-peak hours and standard deviation $\sigma = 0.2\mu$.

In our case study, the demand of each customer consisting of several ordered commodities distributed in different sites is represented by a cluster, and the profit of a cluster is collected if and only if all POPs in the cluster are accessed. We assume that

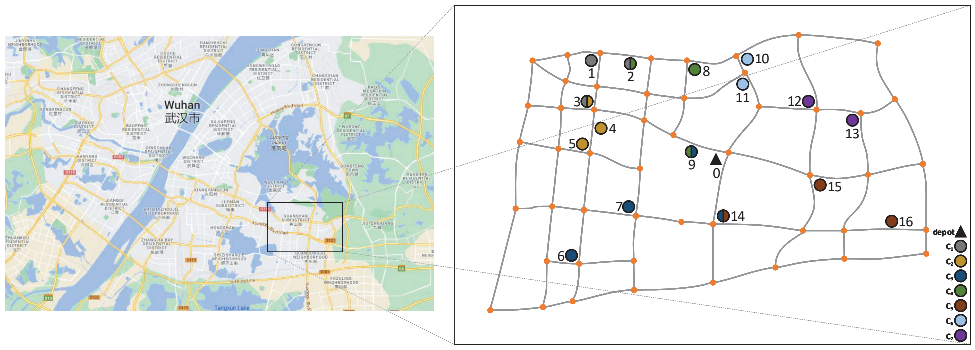


Fig. 12. An intra-city delivery platform in Wuhan city.

if a customer is assigned to a deliveryman, then he/she is responsible for the collection for the ordered commodities of this customer. The goal is to find a tour to visit a subset of customers (clusters) maximizing the total collected profits under the deliveryman's travel time limit. Consequently, this problem can be formulated as a dynamic version of the COP considering stochastic travel time.

Table 10
Summary of the results for the stochastic COP.

|V|	Clusters	T_{max}	Stochastic			
			avg $t(min)$	max $t(min)$	min $t(min)$	Feasible
17	7	75	71.08	89.77	51.81	80113/100000

Table 11
The routes and profits of the real case.

γ	route	profit
50	0-13-12-10-11-8-2-1-3-5-4-9	13
70	0-14-7-6-5-4-3-1-2-8-9	13
90	0-11-10-8-2-1-3-5-4-9	11

Here, we apply the proposed simulation-based optimization method to deal with this real-life case. For the simulation-based optimization method, we set $\Lambda = 100$, $\lambda = 100000$ and $\gamma = 50, 70, 90$. Table 10 presents a summary of the results in an iteration of the proposed simulation-based optimization method, including the instance size, the cluster size, the time limit T_{max}, the average total travel time estimated by the Monte Carlo simulation, the maximum and minimum total travel time estimated by the Monte Carlo simulation, and the number of times the total travel time (estimated by Monte Carlo simulation) is within the maximum time limit T_{max}, respectively. From Table 10, we observe that out of 100000 Monte Carlo simulations, there are 80113 times where the expected total travel time is within the maximum time limit T_{max}. Table 11 shows the different routes of the stochastic COP with $\gamma = 50, 70, 90$. Especially, when $\gamma = 90$, we need to reduce some visited clusters in order to make the route acceptable, as a result, the profits obtained are reduced slightly.

8. Conclusion

In this paper, we propose an effective hybrid evolutionary algorithm for addressing COP with the following original features: (i) an SBTS procedure reinforced by a reinforcement learning mechanism to ensure an accurate and fast search of the neighboring solutions, (ii) a dedicated backbone-based crossover operator to inherit good features, and (iii) a destroy-and-repair mutation operator to diversify the search.

Extensive computational results on 924 benchmark instances in the literature reveal that our proposed algorithm outperforms the existing algorithms by updating the best records (new lower bounds) for 14 instances while matching the best-known results for the remaining cases within a reasonable time. Additional analysis reveal the usefulness of the solution-based tabu strategy, the reinforcement learning strategy, and the crossover/mutation operator. We also presented an application of our approach to deal with a real-life delivery tour planning case (a dynamic version of the COP with stochastic travel time) in a large Chinese city. In the future, we plan to develop efficient and effective algorithms for other related problems, such as the SOP (Archetti et al., 2018) and CTOP (Yahiaoui et al., 2019).

Acknowledgment

We are grateful to the reviewers for their useful comments, which have helped us to significantly improve the paper. This work is partially supported by the National Natural Science Foundation Program of China [Grant No. 72122006, 71771099, 71821001].

References

Angelelli, E., Archetti, C., Vindigni, M., 2014. The clustered orienteering problem. European Journal of Operational Research 238, 404–414.

Archetti, C., Carrabs, F., Cerulli, R., 2018. The set orienteering problem. European Journal of Operational Research 267, 264–272.

Ayadi, W., Hao, J.K., 2014. A memetic algorithm for discovering negative correlation biclusters of dna microarray data. Neurocomputing 145, 14–22.

Battarra, M., Erdoğan, G., Vigo, D., 2014. Exact algorithms for the clustered vehicle routing problem. Operations Research 62, 58–71.

Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T., 2010. F-race and iterated f-race: An overview. Experimental methods for the analysis of optimization algorithms , 311–336.

Campbell, A.M., Gendreau, M., Thomas, B.W., 2011. The orienteering problem with stochastic travel and service times. Annals of Operations Research 186, 61–81.

Carrabs, F., 2021. A biased random-key genetic algorithm for the set orienteering problem. European Journal of Operational Research 292, 830–854.

Chao, I.M., Golden, B.L., Wasil, E.A., 1996a. A fast and effective heuristic for the orienteering problem. European journal of operational research 88, 475–489.

Chao, I.M., Golden, B.L., Wasil, E.A., 1996b. The team orienteering problem. European journal of operational research 88, 464–474.

Chisman, J.A., 1975. The clustered traveling salesman problem. Computers & Operations Research 2, 115–119.

Chou, X., Gambardella, L.M., Montemanni, R., 2021. A tabu search algorithm for the probabilistic orienteering problem. Computers & Operations Research 126, 105107.

Coello, C.A.C., Montes, E.M., 2002. Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. Advanced Engineering Informatics 16, 193–203.

Divsalar, A., Vansteenwegen, P., Sörensen, K., Cattrysse, D., 2014. A memetic algorithm for the orienteering problem with hotel selection. European Journal of Operational Research 237, 29–49.

Dolinskaya, I., Shi, Z.E., Smilowitz, K., 2018. Adaptive orienteering problem with stochastic travel times. Transportation Research Part E: Logistics and Transportation Review 109, 1–19.

Dontas, M., Sideris, G., Manousakis, E.G., Zachariadis, E.E., 2023. An adaptive memory matheuristic for the set orienteering problem. European Journal of Operational Research 309, 1010–1023.

Evers, L., Glorie, K., Van Der Ster, S., Barros, A.I., Monsuur, H., 2014. A two-stage approach to the orienteering problem with stochastic weights. Computers & Operations Research 43, 248–260.

Fischetti, M., Salazar González, J.J., Toth, P., 1997. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. Operations Research 45, 378–394.

Glover, F., Laguna, M., 1998. Tabu search, in: Handbook of combinatorial optimization. Springer, pp. 2093–2229.

Golden, B.L., Levy, L., Vohra, R., 1987. The orienteering problem. Naval Research Logistics 34, 307–318.

Gunawan, A., Lau, H.C., Vansteenwegen, P., 2016. Orienteering problem: A survey of recent variants, solution approaches and applications. European Journal of Operational Research 255, 315–332.

Hanafi, S., Mansini, R., Zanotti, R., 2020. The multi-visit team orienteering problem with precedence constraints. European journal of operational research 282, 515–529.

Hao, J.K., 2012. Memetic algorithms in discrete optimization, in: Handbook of memetic algorithms. Springer, pp. 73–94.

He, P., Hao, J.K., 2022. General edge assembly crossover-driven memetic search for split delivery vehicle routing. Transportation Science 55, 482–511.

Irawan, C.A., Eskandarpour, M., Ouelhadj, D., Jones, D., 2021. Simulation-based optimisation for stochastic maintenance routing in an offshore wind farm. European Journal of Operational Research 289, 912–926.

Kantor, M.G., Rosenwein, M.B., 1992. The orienteering problem with time windows. Journal of the Operational Research Society 43, 629–635.

Kim, H., Kim, B.I., 2022. Hybrid dynamic programming with bounding algorithm for the multi-profit orienteering problem. European Journal of Operational Research 303, 550–566.

Kim, H., Kim, B.I., Noh, D.j., 2020. The multi-profit orienteering problem. Computers & Industrial Engineering 149, 106808.

Lai, X., Yue, D., Hao, J.K., Glover, F., 2018. Solution-based tabu search for the maximum min-sum dispersion problem. Information Sciences 441, 79–94.

Lau, H.C., Yeoh, W., Varakantham, P., Nguyen, D.T., Chen, H., 2012. Dynamic stochastic orienteering problems for risk-aware applications. arXiv preprint arXiv:1210.4874 .

Lin, S., Kernighan, B.W., 1973. An effective heuristic algorithm for the traveling-salesman problem. Operations research 21, 498–516.

Lipowski, A., Lipowska, D., 2012. Roulette-wheel selection via stochastic acceptance. Physica A: Statistical Mechanics and its Applications 391, 2193–2196.

Lu, Y., Benlic, U., Wu, Q., 2020. An effective memetic algorithm for the generalized bike-sharing rebalancing problem. Engineering Applications of Artificial Intelligence 95, 103890.

Lu, Y., Benlic, U., Wu, Q., 2022. A hybrid evolutionary algorithm for the capacitated minimum spanning tree problem. Computers & Operations Research 144, 105799.

Moscato, P., Cotta, C., 2003. A gentle introduction to memetic algorithms, in: Handbook of metaheuristics. Springer, pp. 105–144.

Neri, F., Cotta, C., 2012. Memetic algorithms and memetic computing optimization: A literature review. Swarm and Evolutionary Computation 2, 1–14.

Pěnička, R., Faigl, J., Saska, M., 2019. Variable neighborhood search for the set orienteering problem and its application to other orienteering problem variants. European Journal of Operational Research 276, 816–825.

Sohrabi, S., Ziarati, K., Keshtkaran, M., 2020. A greedy randomized adaptive search procedure for the orienteering problem with hotel selection. European Journal of Operational Research 283, 426–440.

Sun, Z., Benlic, U., Li, M., Wu, Q., 2022. Reinforcement learning based tabu search for the minimum load coloring problem. Computers & Operations Research 143, 105745.

Tsiligirides, T., 1984. Heuristic methods applied to orienteering. Journal of the Operational Research Society 35, 797–809.

Vansteenwegen, P., Souffriau, W., Van Oudheusden, D., 2011. The orienteering problem: A survey. European Journal of Operational Research 209, 1–10.

Wang, B., Bian, Z., Mansouri, M., 2023. Self-adaptive heuristic algorithms for the dynamic and stochastic orienteering problem in autonomous transportation system. Journal of Heuristics , 1–61.

Wang, Y., Wu, Q., Glover, F., 2017. Effective metaheuristic algorithms for the minimum differential dispersion problem. European Journal of Operational Research 258, 829–843.

Woodruff, D.L., Zemel, E., 1993. Hashing vectors for tabu search. Annals of Operations Research 41, 123–137.

Yahiaoui, A.E., Moukrim, A., Serairi, M., 2019. The clustered team orienteering problem. Computers & Operations Research 111, 386–399.

Yu, Q., Fang, K., Zhu, N., Ma, S., 2019. A matheuristic approach to the orienteering problem with service time dependent profits. European Journal of Operational Research 273, 488–503.

Zhou, Q., Hao, J.K., Sun, Z., Wu, Q., 2020. Memetic search for composing medical crews with equity and efficiency. Applied Soft Computing 94, 106440.

Zhou, Q., Hao, J.K., Wu, Q., 2022. A hybrid evolutionary search for the generalized quadratic multiple knapsack problem. European Journal of Operational Research 296, 788–803.

Zhou, Y., Duval, B., Hao, J.K., 2018. Improving probability learning based local search for graph coloring. Applied Soft Computing 65, 542–553.

Zhou, Y., Hao, J.K., Duval, B., 2016. Reinforcement learning based local search for grouping problems: A case study on graph coloring. Expert Systems with Applications

64, 412–422.
Zio, E., Zio, E., 2013. Monte carlo simulation: The method. Springer.