Received March 9, 2022, accepted March 18, 2022, date of publication March 28, 2022, date of current version April 4, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3162692

An Iterative Neighborhood Local Search Algorithm for Capacitated Centered Clustering Problem

YUYING XU[1], PING GUO[1,2], (Member, IEEE), AND YI ZENG[1,2]
[1]College of Computer Science, Chongqing University, Chongqing 400044, China
[2]Chongqing Key Laboratory of Software Theory and Technology, Chongqing 400044, China

Corresponding author: Ping Guo (guoping@cqu.edu.cn)

ABSTRACT The Capacitated Centered Clustering Problem (CCCP) is NP-hard and has many practical applications. In recent years, many excellent CCCP solving algorithms have been proposed, but their ability to search in the neighborhood space of clusters is still insufficient. Based on the adaptive Biased Random-Key Genetic Algorithm (A-BRKGA), this paper proposes an efficient iterative neighborhood search algorithm A-BRKGA_INLS. The algorithm uses shift and swap heuristics to search neighborhood space iteratively to enhance the quality of solutions. The computational experiments were conducted in 53 instances. A-BRKGA_INLS improves the best-known solutions in 23 instances and matches the best-known solutions in 15 instances. Moreover, it achieves better average solutions on multiple instances while spending the same time as the A-BRKGA+CS.

INDEX TERMS CCCP, meta-heuristic, BRKGA, neighborhood search, genetic algorithm.

I. INTRODUCTION

Clustering problems have been applied to many research fields, such as machine learning, pattern recognition, community detection, image segmentation, genetics, microbiology, geology, remote sensing, etc [1], [2]. This paper mainly studies the capacitated centered clustering problem in the clustering problem, which is an abstract problem of location selection, and is also an important decision-making problem. Scientific and reasonable location selection can effectively save resources, reduce costs and ensure high-quality service. The location selection problem has many applications in production, logistics, and daily life. One of the most famous facility-location problems is the p-Median Problem [3]. This problem is defined as follows: Given n points, p medians are selected among them, and n points are assigned to their nearest median. The total distance between each point and its nearest median is minimized.

Another classic location problem is the Capacitated p-Median Problem (CPMP) [4], which has different applications in many practical situations. It can be described as follows: Given n points that each has a known demand, find p medians in n points and assign each point to one median. The total distance from n demand points to their corresponding medians is minimized and the sum of the demands of all points assigned to a median cannot exceed its capacity. Due to capacity constraints, some points may not be assigned to the nearest median.

The Capacitated Centered Clustering Problem (CCCP) [5] studied in this paper is a generalized CPMP, which divides n demand points into p clusters with limited capacity. The goal is to minimize the total distance between each point and the geometric center of its cluster. CCCP has been applied to many fields, such as the location design of garbage collection areas and sales centers [5], the network design of agricultural product supply chains [6], the site selection of offshore wind farms [7], and sibling reconstruction problem (SRP) in computational biology [8].

The main difference between CPMP and CCCP is the features of the locations of the p centers. In the CPMP, the location is determined at a median point. In the CCCP, the location is determined at a centroid. Compared with CPMP, the distance from the median to point in CPMP can be directly obtained from the initially constructed distance matrix. The difficulty of CCCP is that the distance from the cluster center to the point is constantly changing, and the distance calculation will consume more time. Since a given feasible solution to an instance of the CPMP can be converted into a feasible

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Pilato.

solution to an instance of the CCCP, researchers often propose solutions to CPMP and apply them to CCCP after making slight changes. Specifically, for each cluster of the feasible solution to be converted, the selected medians are replaced by the geometric centers of the assigned points. The objective function value is then calculated as the total distance between the newly computed cluster centers and their assigned points. There have been many heuristic and meta-heuristic solutions for CPMP and CCCP in recent years.

Stefanello *et al.* [9] combined meta-heuristics based on local search with mathematical programming techniques. This method applied Iterated Reduction Matheuristic Algorithm (IRMA) to eliminate variables that are unlikely to be good or optimal solutions from the model. The simplified mathematical model was obtained. Baumann [10] presented an extended version of K-Means, which uses binary linear programming to assign points to clusters. In the latest paper of CPMP [11], the author proposes a decomposition strategy for solving large-scale instances. In the local optimization stage, the new IRMA reduction method is used, in which priority is given to sub-problems with great potential to increase the objective function value. The cluster with the largest unused capacity is selected as the initial cluster. This mathematical algorithm is extended to CCCP. The new and most famous solutions were discovered in several CCCP instances. Mai *et al.* [12] used a Gaussian mixture modelling method to construct the solution of CPMP and proposed an improved heuristic. The improved heuristic uses the best improvement search mechanism to shift or swap points between different clusters.

Meta-heuristic algorithms are divided into three categories: meta-heuristics with exact approaches, meta-heuristics with other meta-heuristic components, and meta-heuristics with local search heuristics. Jánošíková *et al.* [13] considered the efficiency of the integer programming solver, combined genetic algorithm with integer programming and proposed two variants. The integer programming solver is used to generate elite individuals in the solving process of the genetic algorithm or as a post-processing technique to improve the best solution of CPMP. Chaves and Lorena [14] combined a simulated annealing algorithm with Clustering Search (CS) [15] proposed by Oliveira *et al.* to solve the CCCP. Chaves and Lorena [16] first used a genetic algorithm to generate solutions and then enhanced the quality of solutions by CS. Muritib [17] proposed a random best-fit construction method and a local search heuristic algorithm based on Tabu Search (TS). Considering CS is the most computationally demanding procedure, Melo Morales *et al.* [18] parallelized the local search component CS, using the Genetic Algorithm as a solutions generator, to solve the CCCP. Caballero Morales [19] proposed a genetic algorithm, which combines the Greedy Random Adaptive Search Process (GRASP) and the K-Means clustering algorithm. Recently, Chaves *et al.* [20] proposed an adaptive Biased Random-Key Genetic Algorithm (A-BRKGA) by improving the Biased Random-Key Genetic Algorithm (BRKGA) [21] and provided a local search component CS to intensify the exploitation of CCCP solutions. This method provided new best solutions for seven classic instances and reported the best solutions in other new instances.

Most researches for CCCP focus on improving the evolution process in the meta-heuristic algorithm, such as [20] improving the meta-heuristic framework by adding parameter control. Literature [14], [16] and [17] respectively combined simulated annealing algorithm, genetic algorithm and tabu search with local search. [18] proposed the strategy of parallel local search component. They implemented a master-slave system with a message passing approach. The related research on the local search for a specific solution of CCCP is relatively shallow, such as literature [17] and [20] both using simple search methods. A targeted local search can effectively improve the quality of solutions. Our method focuses on the design of local search schemes.

The variable neighborhood search algorithm (VNS) is an improved local search algorithm that uses multiple neighborhood structures defined by different functions to perform alternate searches, achieving a good balance between search concentration and evacuation. The variable neighborhood search is based on the following facts: (1) The local optimal solution of one neighborhood structure is not necessarily the local optimal solution of another neighborhood structure. (2) The global optimal solution must be the local optimal solution of all possible neighborhoods.

VNS algorithms mainly depend on the neighborhood structure, search mechanism, and neighborhood movement strategy. Usually, the search order between neighborhood structures is sure. Common search mechanisms include the first and best improvement search strategy. Once the first improvement strategy detects an improved solution, it is set as a new existing solution. The best improvement strategy selects the best of all improved solutions as the new solution. Neighborhood movement strategies mainly include returning to the first neighborhood, searching in the same neighborhood, and searching for the next neighborhood. The literature [22] reported the impact of different combinations of VNS on the solution quality when used as a local search.

The variable neighborhood search algorithm has a good effect in solving travelling salesman problems, location problems, vehicle routing problems, etc. At present, researchers have also proposed a variety of improved variable neighborhood algorithms, such as general variable neighborhood search (GVNS) and skewed general variable neighborhood search (SGVNS) [23]. In addition, related literature such as [24] and [25] relax the constraints and use penalty functions to evaluate infeasible solutions. In this way, the infeasible space is explored. Exploring greater solution space and increasing the diversity of solutions improves the possibility of finding a better solution.

The initialization and evolution rules of the population in A-BRKGA [20] can produce many solutions with different structures. Thus, the diversity of solutions is ensured. Through experimental analysis, we found that some instances in the data sets in [20] have many clusters with a few points, and the assignments of points at the edge of these clusters are highly variable. The local search component (CS) has less exploration in the neighborhood space between clusters. Based on A-BRKGA [20], this paper proposes an iterative neighborhood search algorithm called A-BRKGA_INLS to enhance the exploration of neighborhoods. The main innovations of this paper include:

(1) The algorithm separately searches the shift neighborhood and the swap neighborhood, instead of simultaneously searching for feasible shifts and swaps. The iterative variable neighborhood search method better balances the concentration of the search and the evacuation of the genetic algorithm. Based on the optimal shift neighborhood, the swap neighborhood is rechecked. The shift neighborhood has more adjustment space and can develop in a better direction.

(2) The algorithm combines inexact search and exact search. The inexact search in the evolution process has higher conditions for point movement. It skips the movement that can only cause a weak increase in the objective function, avoiding the early fixation of the cluster allocation and losing more exploration opportunities, thereby effectively avoiding falling into the local optimum prematurely. The exact search checks the shifts and swaps ignored by the inexact search, which further improves the quality of solutions.

The experimental results in 53 instances show that the proposed method has good performance. Compared with A-BRKGA+CS, A-BRKGA_INLS can find better solutions on multiple instances in approximately equal time.

The remainder of the paper is organized as follows. In Section II, a formal description of CCCP is given, and the basic idea of BRKGA is introduced. Then, based on BRKGA, the application of A-BRKGA to CCCP is described. In Section III, the algorithm A-BRKGA_INLS for CCCP is given. In Section IV, we introduce the experimental dataset. In Section V, we report the results of the computational analysis. Finally, in Section VI, the paper is summarized, and the future research direction is prospected.

II. BACKGROUND

A. MATHEMATICAL MODEL OF CCCP

CCCP can be formally expressed as an optimization problem, as shown in Equation (1).

$$\boldsymbol{minimize} \sum_{j\in J}\sum_{i\in I} ||a_i - \bar{x}_j|| y_{ij}$$

$$\sum_{j\in J} y_{ij} = 1, \quad \forall i \epsilon I \tag{1}$$

$$\sum_{i\in I} q_i y_{ij} \le Q_j, \quad \forall j \epsilon J \tag{2}$$

$$x_j \in \Re^2,\ n_j \in \mathrm{N},\ y_{ij} \in \{0,1\}, \quad \forall i \epsilon I,\ \forall j \epsilon J \tag{3}$$

I is the set of demand points; J is the set of cluster centers; a_i is the coordinate of point i; $\bar{x}_j$ is the coordinate of the geometric center of cluster j; $y_{ij} = 1$, if the point i is assigned to cluster j, and $y_{ij} = 0$ otherwise; q_i is the demand of the point i; Q_j is the capacity of cluster j.

The objective function (1) minimizes the total distance between each cluster centers and their assigned points. Constraints (2) require that each point must be assigned to one cluster. Constraints (3) impose that the sum of demands of all points in one cluster should not exceed the cluster capacity. Decision variables are defined.

B. BIASED RANDOM-KEY GENETIC ALGORITHM

BRKGA is a general search meta-heuristic algorithm proposed by Goncalves and Resende [21], which is based on the Random-Key Genetic Algorithm (RKGA) introduced by Bean [26]. Recently, BRKGA has been combined with many combinatorial optimization problems, such as the Permutation Flow-shop Scheduling Problem [27], the Two-stage Capacitated Facility Location Problem [28], the Network Hubs Location Problem [29], and the Vehicle Routing Problem [30].

In BRKGA, each gene on the chromosome is a decimal number in the interval [0, 1]. The number is called the random-key. N random-keys form a vector that represents a chromosome. The random-keys vector needs to be converted into a solution for the combinatorial optimization problem by a specific decoder to calculate its fitness.

The initial population of BRKGA consists of p random-keys vectors (individuals). All random-keys in each vector are generated independently and randomly. In each generation of population evolution, first, the fitness of the newly created random-keys vectors are calculated by the decoder. Then, the population is divided into an elite group and a non-elite group by fitness. Next, individuals of the next generation are generated, as shown in Figure 1. (1) pe elite individuals are directly copied into the next generation. (2) A small number of pm mutants are added to the next generation. The mutation in BRKGA is different from that in genetic algorithms. These mutants are generated the same way as the individuals in the initial population. (3) To make up a population of size p, $p-pe-pm$ individuals need to be produced. These individuals are generated by parameterized uniform crossover [31]: two individuals (parents) are selected randomly from the elite and the non-elite groups. It can be seen that BRKGA is an evolutionary algorithm that performs multiple iterations on random-keys vectors.

C. ADAPTIVE BIASED RANDOM-KEY GENETIC ALGORITHM FOR SOLVING CCCP

Chaves *et al.* [20] proposed A-BRKGA based on BRKGA and designed a special decoder for CCCP. This paper applies A-BRKGA to evolve populations and uses the same CCCP decoder. We will briefly introduce the encoding and decoding of CCCP instances and the difference between BRKGA and A-BRKGA.

For each CCCP instance, points are numbered from 1 to n (the number of points in the instance). In this way, one point can be indexed by a serial number. A random-key in the

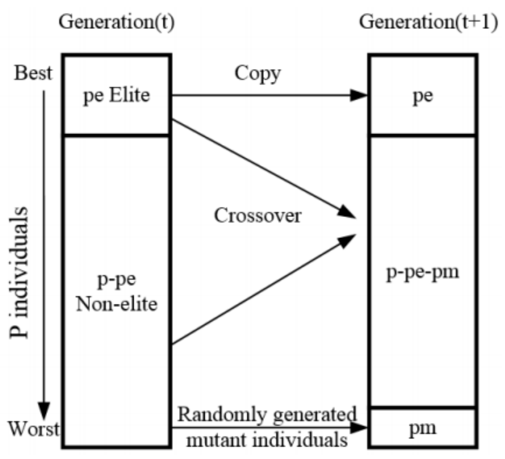


FIGURE 1. BRKGA: generation t+1 from generation t.

interval [0, 1] is randomly generated for each point. Besides, two random-keys are generated for the $n + 1$th and $n + 2$th positions. Therefore, the code length of CCCP is $n+2$.

CCCP decoder is based on the best-fit construction method [17]. The last two random-keys of the encoding related to the perturbation and crossover probability do not participate in the decoding process. First, the first n random-keys are sorted in descending order, and the last two random-keys remain unchanged. Then, the first m demand points of the random list are put into m clusters as the initial center of the cluster, and the remaining $n-m$ demand points are placed according to the best-fit principle. (The best-fit principle always selects the cluster closest to the demand point to join when meeting the capacity constraint.) The cluster center and occupied capacity are updated whenever a new point is added. When all the points are added to the appropriate cluster, a feasible solution is formed. The decoder will calculate the solution fitness, using the objective function (Equation (1)) as the fitness calculation equation.

Compared with BRKGA, A-BRKGA adds perturbation strategy of elite individuals and deterministic and adaptive parameters update strategy.

The perturbation strategy of elite individuals perturbs individuals with the same fitness in the elite group. The perturbation strategy increases the diversity of the elite group and avoids falling into the local optimum prematurely.

Deterministic parameter update is reflected in controlling the proportion of elite individuals and mutant individuals in the population. As the number of iterations increases, the proportion of elite individuals κ_e increases, and the proportion of mutant individuals κ_m decreases.

A-BRKGA inserts two random-keys at the positions $n + 1$ and $n + 2$ of the vector to make the parameters self-adaptive. Perturbation probability β and parameterized uniform crossover ρ_e are updated based on the two random-keys. The updates of β and ρ_e are both non-deterministic. Each perturbed chromosome has its perturbation probability. Adaptive parameter update allows the gene sequence to be changed to different degrees, which improves the possibility of producing solutions with different structures.

III. THE CAPACITATED CLUSTER CENTERED PROBLEM OF A-BRKGA

In this section, we will present A-BRKGA_INLS in detail. First, we provide a general framework of the algorithm, showing how the two local search algorithms are combined with A-BRKGA. Then, the specific process of inexact Iterative Neighborhood Local Search Algorithm (IINLS) is explained. Finally, the exact Iterative Neighborhood Local Search Algorithm (EINLS) is introduced, and the difference between IINLS and EINLS is discussed.

Algorithm 1 A-BRKGA_INLS; //A-BRKGA With Iterative Neighborhood Local Search

```
Input: the number of generation max_gen, the size of population $p$,
the search internal K, a null solution $S*$ which its objective function
value is set to INFINITY;
Output: the local optimal solution $S**$;
(1)  Randomly generate initial population $Pop$ with p vectors;
(2)  for $i = 1$ to max_gen do
(3)    $Pop$ ←population_update($Pop$); // Ref. A-BRKGA
(4)    if ($i$%K = 0) then
(5)      $promisingPop$←Lable_Propagation ($Pop$); // Ref. [20]
(6)      for each solution $S$ in $promisingPop$ do
(7)        $S_{searched}$←IINLS ($S$); // Inexact Iterated Neighborhood Local Search
(8)        if($S_{searched} < S*$) then $S* \leftarrow S_{searched}$;
(9)      endfor
(10)   endif
(11) endfor
(12) $S**$ ←EINLS ($S*$);// Exact Iterated Neighborhood Local Search
(13) Return $S**$;
```

A. ALGORITHM FRAMEWORK

We utilized A-BRKGA to evolve the population. In the process of population evolution, elite individuals are clustered at certain generations, and an Iterative Neighborhood Local Search (IINLS) is performed on the individuals with the highest fitness in each cluster. Another Iterative Neighborhood Local Search (EINLS) is performed when the population evolution is finished to improve the search quality. EINLS has the same framework as IINLS, but the criteria for moving or not moving is different.

The primary process of A-BRKGA_INLS is listed in **Algorithm 1**. Firstly, the initial population of size p is generated randomly (line 1). Secondly, in each generation, the population is updated by copying elite individuals, perturbing similar elite individuals, mutation, and crossover (line 3). Then, for every particular generation (K), the elite individuals are clustered by the Label Propagation (line 5). IINLS is applied to individuals with the highest fitness in each cluster to improve the quality of individuals (line 7, see Section III, Part B). The current best solution is stored in $S*$ (line 8). When the maximum number of generations (max_gen) is reached, the population evolution is finished. At this time, a exact local search algorithm EINLS is implemented to get the

TABLE 1. Points in the simple example.

Number	X	Y	Demand
1	8	20	1
2	10	10	1
3	11	36	1
4	13	21	1
5	16	27	1
6	19	32	1
7	20	14	1
8	28	29	1
9	32	13	1

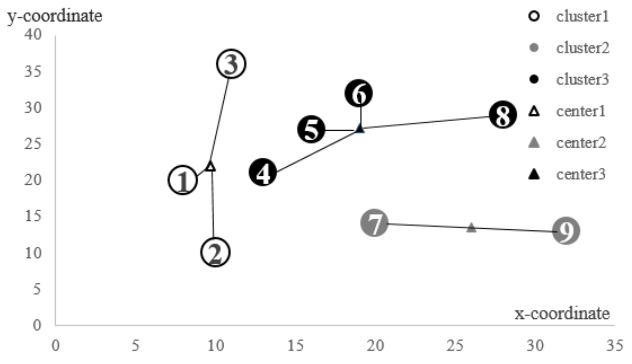
final best solution $S**$ (line 12, see Section III, Part C). The algorithm ends.

Compared with A-BRKGA+CS[20], A-BRKGA_INLS has two differences. (1) CS simultaneously searches for feasible shifts and swaps, and IINLS separately searches shift neighborhood and swap neighborhood to thoroughly explore the neighborhood space of solutions. (2) When the evolution is completed, A-BRKGA+CS no longer searches, while A-BRKGA_INLS applies EINLS to improve individuals' quality further.

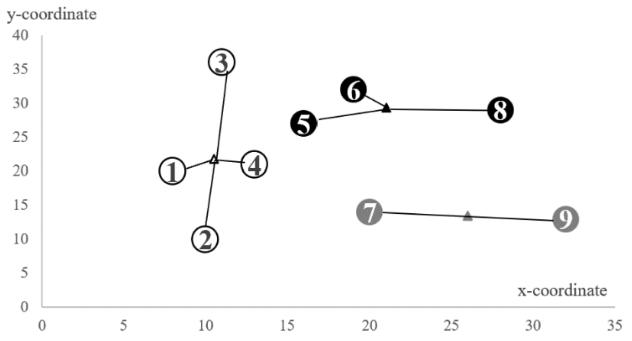
Figure 2 shows a simple example to introduce the scenario our algorithm aimed, where lines connect points in the same cluster. It is required to divide 9 points into 3 clusters with a capacity constraint of 4. Table 1 shows the coordinates and points demand. f denotes the value of the objective function. For the initial solution in Figure 2(a), we use A-BRKGA_INLS combined with the inexact iterative neighborhood search to solve (see (b)(c)(d) for the process)). The initial solution $f = 66.3$.

First, perform the inexact iterative neighborhood search. IINLS searches for the shift neighborhood of the current solution in increasing order of cluster labels. Until cluster 3 is traversed, point 4 in cluster 3 is moved to cluster 1 according to $\Delta f = 5.18$. The solution is updated, as shown in Figure 2(b). Then search for the shift neighborhood of Figure 2(b). Point 3 in cluster 1 is moved to cluster 3 according to $\Delta f = 2.24$, as shown in Figure 2(c). The inexact search reaches the local optimum with $f = 52.2$. The inexact search can move across these points that cause a slight improvement. It is beneficial to the final solution. Figure 2(c) is the best solution in the evolution process. Finally, A-BRKGA_INLS applies exact iterative neighborhood search to Figure 2(c). Point 7 can be shifted to cluster 1 with f decreasing 2.4, but the inexact search ignores it. Update it, and figure 2(d) shows the new solution. There is no better shift neighborhood solution and swap neighborhood solution, so iterations end with $f = 49.8$.

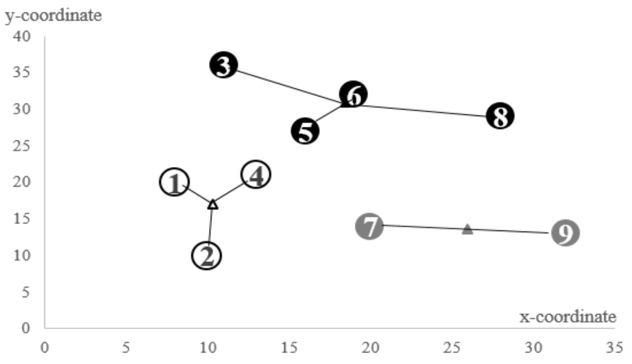
It can be seen that A-BRKGA_INLS is suitable for the situation where the exact search for slight improvement is easy to fall into the local optimum early. In A-BRKGA_INLS, the inexact search is used to improve the solution more, and then the exact search is used to improve the solution that is already good enough. By their combination,



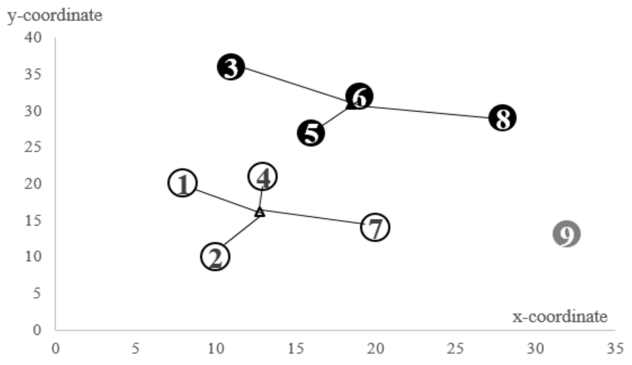
(a). Initial solution.



(b). Result (1) of the inexact iterative neighborhood local search.



(c). Result (2) of the inexact iterative neighborhood local search.



(d). Result of the exact iterative neighborhood local search.

FIGURE 2. Example of neighborhood search in A-BRKGA_INLS.

A-BRKGA_INLS explores the neighborhood space more fully as a whole. Subsequent chapters will detail the neighborhood iteration process of A-BRKGA_INLS.

B. INEXACT ITERATIVE NEIGHBORHOOD LOCAL SEARCH ALGORITHM (IINLS)

This paper uses the Label Propagation algorithm [32] to cluster the elite individuals, aiming at clustering the elite individuals with high similarity in the same cluster. Only the individual with the highest fitness in each cluster is selected for local search to speed up the evolution process (Individuals that already have been searched are not considered). Label Propagation uses the Pearson correlation coefficient [33] to measure if two random-keys vectors could return similar solutions.

Applying clustering search to each generation of the population usually does not help get a better solution. On the contrary, it will shorten the number of generations and fail to get good performance. Therefore, we only perform the clustering search for every K generations.

A point in cluster A is transferred to cluster B when the capacity of cluster B is not less than the sum of the occupied demand and the demand of the transferred point. The move is called a shift. The solution obtained by shifting on solution S is called a shift neighborhood solution of solution S. The shift neighborhood of solution S ($N1(S)$) comprises all possible shift neighborhood solutions. In one instance with n demand points and m clusters, each demand point can be shifted to other m-1 clusters. Therefore, the number of neighborhood solutions contained in $N1(S)$ does not exceed n×m. A point in cluster A is transferred to cluster B, and another point in cluster B is transferred to cluster A when the capacity of cluster A and cluster B is not less than the occupied demand plus the demand of the point swapped in minus the demand of the point swapped out. The move is called a swap. The solution obtained by swapping on solution S is called a swap neighborhood solution of solution S. The swap neighborhood of solution $S(N2(s))$ comprises all possible swap neighborhood solutions. Each point can be swapped with any point only if they are not in the same cluster. Therefore, the number of neighborhood solutions contained in $N2(S)$ does not exceed n^2.

This paper proposes a new iterative neighborhood local search method, IINLS, which combines shift neighborhood and swap neighborhood. The specific search processes of $N1$ and $N2$ are described in **Algorithm 2** and **Algorithm 3**. In **Algorithm 2** and **Algorithm 3**, whether to shift or swap is not determined by the precise objective function increment. For swaps, we use the evaluation function in [17] (Equation (4)), in which u and v are the points to be swapped, A_{old} and B_{old} are the original cluster centers, and A_{new} and B_{new} are new cluster centers after swapping. For shifts, we only make a simple judgment (Equation (5)). We utilize an auxiliary data structure to record the geometric centers of all clusters so that Δf can be calculated in O (1). Swaps or shifts are performed when $\Delta f > 0$ is satisfied. Besides, $q[x]$ represents the point's demand, $Q[j]$ represents the cluster's occupied demand and Q represents the cluster's capacity (**Algorithm 2** line7).

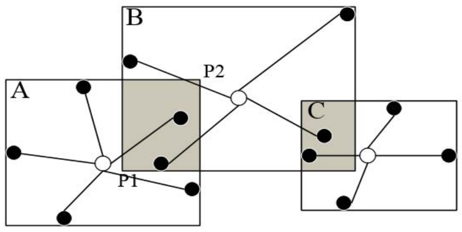


FIGURE 3. Example of overlapping clusters.

The swap neighborhood is larger than the shift neighborhood. To reduce the search scope of swap neighborhoods, we only check points in the overlapping areas. Figure 3 describes an example of three clusters where clusters A and C are not overlapping and so swaps between the two clusters are not considered. We utilize auxiliary data structures to store the boundary coordinates of clusters A, B, and C. The minimum coordinates P1(min_x, min_y) and the maximum coordinate P2(max_x, max_y) of the overlapping rectangular area can be calculated. For instances having many points in a single cluster, checking only points in the overlapping area can speed up the search of swap neighborhoods.

$$\Delta f = d(A_{old}, u)+d(B_{old}, v) - d(A_{new}, v)-d(B_{new}, u) \quad (4)$$

$$\Delta f = d\ (A_{old}, u) - d\ (B_{old}, u) \quad (5)$$

The process of IINLS is described in **Algorithm 4**. In this algorithm, the search starts from shift neighborhood $N1(S)$ of solution S. When it encounters a feasible shift, this move is executed, and a new solution S_1 is obtained. The search continues from shift neighborhood $N1(S_1)$ of solution S_1. Then, when the current solution S_i completely traverses searching shift neighborhood $N1(S_i)$, and no move can improve the solution (line 2-4), IINLS updates the cluster boundaries and rectangular areas (line 5), and then goes to the swap neighborhood $N2(S_i)$ (line 6). If a feasible swap can be performed, a new solution S_{i+1} is obtained. It does not immediately go to the shift neighborhood $N1(S_{i+1})$ but continues to check the swap neighborhood $N2(S_{i+1})$. Similarly, swap neighborhood search ends when the current solution S_j has no improved solution in $N2(S_j)$. If the solution obtained from the shift neighborhood has improved in the swap neighborhood, it is necessary to go back to the shift neighborhood to search again. The stop criterion is that there is no feasible move in both the shift neighborhood and the swap neighborhood of the current solution. At this time, the solution reaches a local optimum, and its fitness is calculated (lines 8, 9). The algorithm uses the array 'Changed' to mark whether clusters have been changed to update the overlapping areas. The Boolean named *Flag* is to record whether the solution has been improved. The search process does not change the random-keys vector, so IINLS does not affect the evolution of the individual in A-BRKGA.

IINLS utilizes two evaluation functions to check feasible shifts and swaps, so the judgement of moves may be inexact. This search process is called the inexact search. Inexact search improves the movement conditions, avoids missing significant growth due to slight growth, and further explores better neighborhood solutions, which effectively avoids the evolution falling into local optimum prematurely. On the other hand, IINLS can cause the objective function to descend continuously in the shift neighborhood. Although a swap is equivalent to two consecutive points shifts (point 1 and point 2), a shift does not bind the two points. If other moves are performed after moving point 1, the target cluster of point 2 will be affected, or even point 2 will not be shifted. In search of the shift neighborhood, the solution has more freedom to change direction, leading to better solutions.

Algorithm 2 LocalSearchN1

```
Input: a current solution $S$, which is a solution in $promisingPop$ or got
from Algorithm2 LocalSearchN1 or Algorithm3 LocalSearchN2;
Output: the local optimum solution, $Flag$, $Changed[]$;
(1)  $Flag = false$;
(2)  for $i \leftarrow 1$ to $m$ do /*m is the number of clusters to be partitioned. */
(3)   for $j \leftarrow 1$ to $m$ do
(4)    for $x \in clusteri$ do
(5)     if($i \neq j$ and $(Q[j] + q[x] \leq Q)$) then /*the cluster capacity constraint
must be satisfied. */
(6)     Compute $\Delta f$; /*according to the Equation (5). */
(7)     if($\Delta f > 0$) then
(8)      Move $x$ to $cluster\ j$;
(9)      Update $mean[i]$, $mean[j]$;
(10)     $Flag = true$, $Changed[i] = true$, $Changed[j] = true$;
(11)    end if
(12)   end if
(13)   end for
(14)  end for
(15) end for
(16) Return $< S, Flag, Changed[] >$;
```

Algorithm 3 LocalSearchN2

```
Input: a current solution $S$ which got from Algorithm LocalSearchN1, the
overlapped rectangle area of each two clusters $overlap[][]$;
Output: the local optimum solution, $Flag$, $Changed[]$;
(1) $Flag = false$;
(2) for $i \leftarrow 1$ to $m$ do
(3)  for $j \leftarrow i + 1$ to $m$ do
(4)   for $x \in cluster\ i : x \in Overlap[i][j]$ do
(5)    for $y \in cluster\ j$: $y \in Overlap[i][j]$ do
(6)     if $((Q[i] + q[y] - q[x] \leq Q)$ and $(Q[j]+q[x]-q[y] \leq Q))$ then
(7)    Compute $\Delta f$; /* according to the Equation (4). */
(8)     if ($\Delta f > 0$) then
(9)      Move $x$ to $cluster\ j$; Move $y$ to $cluster\ i$;
(10)     Update $mean[i]$, $mean[j]$;
(11)     $Flag = true$, $Changed[i] = true$, $Changed[j] = true$;
(12)     end if
(13)    end if
(14)   end for
(15)   end for
(16)  end for
(17) end for
(18) Return $< S, Flag, Changed[] >$;
```

Algorithm 4 IINLS;//Inexact Iterated Neighborhood Local Search

```
Input: an initial solution $S$ in $promisingPop$;
Output: the local optimum solution $S_{searched}$;
(1)  repeat
(2)   repeat
(3)    $(S, Flag, Changed[]) \leftarrow$ LocalSearchN1 $(S)$;
(4)   Until $Flag = false$;
(5)   $Overlap[][] \leftarrow$ UpdateOverlap $(Changed[])$;
(6)   $(S, Flag, Changed[]) \leftarrow$ LocalSearchN2 $(S, Overlap[][])$;
(7)  Until $Flag = false$;
(8)  $S_{searched} \leftarrow S$;
(9)  Compute the fitness of solution $S_{searched}$;
(10) Return $S_{searched}$;
```

C. EXACT ITERATIVE NEIGHBORHOOD LOCAL SEARCH ALGORITHM (EINLS)

At the end of population evolution, an exact local search algorithm for the current best solution is executed (EINLS, **Algorithm1** line 12). In EINLS, the objective function increment is accurately calculated to determine whether the shift or swap is feasible. The search process is called the exact search. Since there are only a few points in each cluster, a shift or swap causes a significant change in the cluster center. However, evaluation functions only consider the point being moved and ignore the influence on the remaining points in two clusters. Figure 4 shows an example that even if the condition $\Delta f > 0$ is satisfied, the movement will degrade the quality of the solution. It illustrates the importance of calculating the objective function increment precisely at the last step. As shown in Figure 4(a), the light gray points are in cluster 1, and the dark gray points are in cluster 2. Before shifting, the centers of clusters 1 and 2 are points $C1$ and $C2$ (triangle markers), respectively. It can be seen that the distance from point P to $C1$ is greater than the distance to $C2$, which satisfies the condition $\Delta f > 0$ (**Algorithm 2** line 8). If the inexact shift is performed, point P will be moved to cluster 2, and the centers of clusters 1 and 2 will become points $C1$' and $C2$', as shown in Figure 4(b). As a result, the sum of distances in two clusters increases, and the objective function increases. If we can accurately calculate the objective function increment, point P will not be shifted to cluster 2. Precise calculation avoids the degradation of the quality of the solution.

EINLS keeps the same process as IINLS but calculates the objective function increment caused by swaps and shifts instead of using evaluation functions (in **Algorithm2** line 6, 7, **Algorithm3** line 7, 8, Δf is not used). This step detects feasible shifts and swaps ignored by inexact search due to the higher move conditions, improving the quality of solutions at a low time cost. The solution must undoubtedly be better or unchanged because EINLS accurately calculates the objective function.

It should be noted that the search process is independent of coding and decoding in A-BRKGA. After the A-BRKGA component decodes the chromosome, each point is marked with the cluster label. The proposed methods, IINLS and EINLS, only use label information to handle the allocation of demand points.

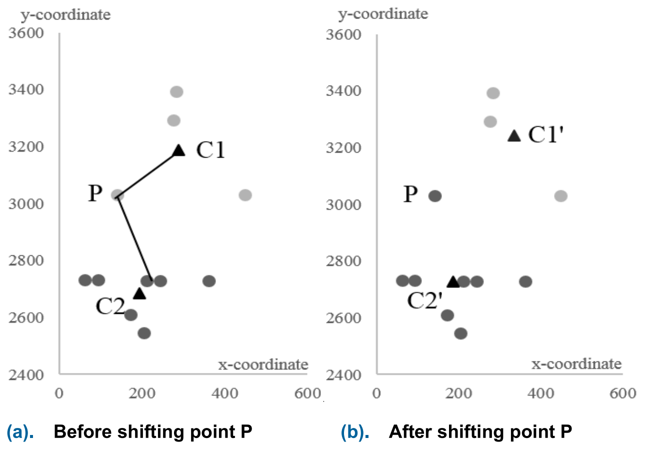


FIGURE 4. Example of performing an inexact shift.

IV. EXPERIMENTAL DATA SETS

A-BRKGA_INLS is coded in C++ and tested on Intel (R) Xeon (R) Platinum 8269CY 2.5 GHz processor with 8 GB of RAM. The parameter setting of A-BRKGA is the same as [20]. We use nine CCCP data sets in reference [20] to compare the proposed method with A-BRKGA + CS [20]. Nine data sets contain 53 CCCP instances. In Table 2, there are three data sets named TA, doni [5], and sjc [34] (including 20 instances). Table 3 contains the data set SJC-n proposed by Pereira, Lorena, and Senne [35] for the Maximum Coverage Location Problem (including eight instances) and five data sets (named lin318-m, u724-m, rl1304-m, pr2392-m, and fnl4461-m) for CPMP generated by Stefanello, de AraúJo, and Müller [36] (including 25 instances). All of the above instances are from (https://sites.google.com/site/antoniochaves/publications/data).

This paper does not compare data set x-n-m [36]. The data set is based on the Capacitated Vehicle Routing Problem, and the upper limit of cluster capacity is tight. In the experiment, it is found that many illegal solutions that do not meet the capacity constraints are generated. Iteration of a large-scale population takes much time, so we abandon the data set.

Table 2 and Table 3 show the essential characteristics of instances: number of points (n), number of clusters (m), cluster capacity (Q), average point demand (q_Avg), and point demand standard deviation (q_Dev). In Table 3, each row represents multiple instances with the same number of points, and the number of clusters in column m corresponds to the cluster capacity in column Q. In data set SJC-n, there are a few points whose demands exceed the cluster capacity. For this situation, we delete these demand points and reduce the number of clusters of this instance accordingly.

V. COMPARISON OF EXPERIMENTAL RESULTS

First, the experimental results are compared between A-BRKGA_INLS and A-BRKGA+CS [20]. Then, comparative experiments were carried out from two perspectives to illustrate the superiority of our algorithm: the combination of inexact search and exact search and iterative neighborhood search.

TABLE 2. Characteristics of 20 CCCP benchmark instances.

instances	*n*	*m*	*Q*	*q_Avg*	*q_Dev*
ta25	25	5	6	1	0
ta50	50	5	11	1	0
ta60	60	5	13	1	0
ta70	70	5	17	1	0
ta80	80	7	12	1	0
ta90	90	4	23	1	0
ta100	100	6	17	1	0
sjc1	100	10	720	58.07	51.86
sjc2	200	15	840	46.34	37.15
sjc3a	300	25	740	37.51	29.58
sjc3b	300	30	740	37.51	29.58
sjc4a	402	30	840	39.76	32.87
sjc4b	402	40	840	39.76	32.87
doni1	1000	6	200	1	0
doni2	2000	6	400	1	0
doni3	3000	8	400	1	0
doni4	4000	10	400	1	0
doni5	5000	12	450	1	0
doni6	10000	23	450	1	0
doni7	13221	30	450	1	0

A. ALGORITHM COMPARISON

Table 4 and Table 5 show the results of A-BRKGA+CS and A-BRKGA_INLS. The computational tests limit the running time in 1000s, and each instance was run continuously 20 times with different random seeds. The entries in the table are the best-known solution (*best-known*), the best solution (*sol**), the average solution (*sol*) over 20 runs, the average running time to find the best solution (T^*), the average running time of the instance (T) in seconds, the absolute difference between the best solution and the best-known solution (*Gap*), the deviation between the best solution and the average solution (*Dev*), and the difference between the best solutions of the two algorithms (*Diff*).

$$Gap = \frac{sol^* - best - known}{best - known} * 100$$
$$Dev = \frac{sol - sol^*}{sol^*} * 100$$
$$Diff = \frac{sol^*_{CS} - sol^*_{INLS}}{sol^*_{INLS}} 100$$

Gap represents the gap between the current algorithm solution and the best-known solution, and *Dev* reflects algorithm stability. We use bold to mark solutions that are better than the best-known solutions. Data for each entry is averaged to compare the overall performance. The average data are placed in the last row.

In Table 4, A-BRKGA_INLS and A-BRKGA+CS have the same ability to obtain the best solutions ($Diff = 0$),

TABLE 3. Characteristics of 33 CCCP benchmark instances.

instances	*n*	*m*	*Q*	*q_Avg*	*q_Dev*
SJC-324	324	{25,30}	{500,500}	37.51	57.11
SJC-500	500	{45,50}	{450,450}	39.41	61.4
SJC-708	708	{70,80}	{350,350}	34.17	54.85
SJC-818	818	{85,90}	{350,350}	35.66	66.18
lin318	318	{5,15,40,70,100}	{3710,1392,522,298,209}	52.49	29.01
u724	724	{10,30,75,125,200}	{4175,1479,592,355,222}	49.01	28.31
rl1304	1304	{10,50,100,200,300}	{4175,1479,592,355,222}	49.94	28.31
pr2392	2392	{20,75,150,300,500}	{7616,2031,1016,508,305}	50.94	29.05
fnl4461	4461	{20,100,250,500,1000}	{14025,2805,1122,561,281}	50.3	28.63

TABLE 4. Comparison of computational results in 20 CCCP benchmark instances.

instances	*best-known*	*A-BRKGA_INLS*						*A-BRKGA+CS*						*Diff*
		*sol**	*sol*	*T**	*T*	*Dev*	*Gap*	*sol**	*sol*	*T**	*T*	*Dev*	*Gap*	
ta25	1251.44	1251.45	1251.45	0.21	5.37	0.00	0.00	1251.45	1251.45	0.04	0.44	0.00	0.00	0.00
ta50	4474.52	4474.52	4474.52	0.13	8.54	0.00	0.00	4474.52	4474.52	0.51	2.24	0.00	0.00	0.00
ta60	5356.58	5356.58	5356.58	2.19	8.52	0.00	0.00	5356.58	5356.58	1.25	3.6	0.00	0.00	0.00
ta70	6240.67	6240.67	6240.67	3.09	9.50	0.00	0.00	6240.67	6240.72	2.18	5.36	0.00	0.00	0.00
ta80	5730.28	5730.28	5730.28	2.46	12.62	0.00	0.00	5730.28	5730.28	3.8	8.96	0.00	0.00	0.00
ta90	9069.85	9069.85	9069.85	1.23	10.40	0.00	0.00	9069.85	9069.85	3.44	10.85	0.00	0.00	0.00
ta100	8102.04	8102.04	8103.51	3.62	13.79	0.02	0.00	8102.04	8102.99	6.36	16.22	0.01	0.00	0.00
sjc1	17359.75	17359.75	17368.26	4.62	20.24	0.05	0.00	17359.75	17370.73	10.2	21.73	0.06	0.00	0.00
sjc2	33181.65	33181.65	33192.32	33.18	53.32	0.03	0.00	33181.65	33195.68	38.48	57.28	0.04	0.00	0.00
sjc3a	45354.29	45354.29	45464.46	90.97	125.08	0.24	0.00	45354.29	45494.34	104.33	127.15	0.31	0.00	0.00
sjc3b	40660.55	40660.55	40775.23	100.06	147.81	0.28	0.00	40660.55	40765.31	120.73	137.74	0.26	0.00	0.00
sjc4a	61931.6	61948.21	62133.71	169.00	190.01	0.30	0.03	61937.94	62199.04	172.73	193.76	0.42	0.01	-0.02
sjc4b	52196	52196.00	52355.35	225.47	253.39	0.31	0.00	52202.48	52351.85	175.91	201.21	0.29	0.01	0.01
doni1	3017.37	3018.51	3021.83	93.34	96.60	0.11	0.04	3018.17	3024.37	131.93	147.19	0.21	0.03	-0.01
doni2	6080.7	6371.92	6373.16	173.07	174.85	0.02	4.79	6372.81	6377.83	353.71	370.35	0.08	4.80	0.01
doni3	8343.49	8395.87	8423.67	138.30	313.32	0.33	0.63	8390.58	8428.76	418.35	674.28	0.46	0.56	-0.06
doni4	10777.64	10922.15	10949.12	473.40	479.01	0.25	1.34	10817.87	10860.63	188.2	1014.38	0.40	0.37	-0.95
doni5	11114.67	11133.20	11214.91	735.56	747.56	0.73	0.17	11191.06	11308.52	691.1	1006.37	1.05	0.69	0.52
doni6	15453.83	15529.09	15608.00	567.18	1005.17	0.51	0.49	15614.36	15766.85	740.28	1023.04	0.98	1.04	0.55
doni7	18484.13	18943.46	19104.62	569.67	1008.63	0.85	2.48	18943.67	19173.87	733.9	1060.46	1.22	2.49	0.00
average	18209.05	18262.00	18310.58	169.34	234.18	0.20	0.50	18263.53	18327.21	194.87	304.13	0.29	0.50	0.00

and the average gap between the best solutions and the best-known solutions is both 0.5% ($Gap = 0.5$). Our method gets equal or better average solutions in 17 of 20 instances. Compared to A-BRKGA+CS ($Dev = 0.29$), our method provides better robustness ($Dev = 0.20$).

In Table 5, A-BRKGA_INLS improves 23 best-known solutions and matches four best-known solutions. The quality of A-BRKGA_INLS solutions is 2.16% higher than A-BRKGA+CS solutions ($Diff = 2.16$), and the gap with the best-known solution has decreased from $Gap = 1.36$ to $Gap = -0.75$. Especially on data sets pr2392 and fnl4461 with thousands of points, the quality of our solutions has been significantly improved, exceeding 20% at the most. Besides, our method can get equal or better average solutions in 27 instances. In Table 4, A-BRKGA_INLS ($Dev = 1.06$) shows stronger robustness than A-BRKGA+CS ($Dev = 1.12$). We also apply the Wilcoxon signed-rank test (WSR) to analyze if a significant difference exists between the solutions of A-BRKGA_INLS and A-BRKGA+CS. The WSR shows p-value $= 0.001$. The result indicated that there is a significant difference between A-BRKGA_INLS and A-BRKGA+CS, and INLS provides better solutions than CS.

It can be seen from Table 4 and Table 5 that A-BRKGA_INLS has no significant increase in running time compared to A-BRKGA+CS.

TABLE 5. Comparison of computational results in 33 CCCP benchmark instances.

instances	*best-known*	*A-BRKGA_INLS*						*A-BRKGA+CS*						*Diff*
		*sol**	*sol*	*T**	*T*	*Dev*	*Gap*	*sol**	*sol*	*T**	*T*	*Dev*	*Gap*	
SJC324-25	52955.34	**52949.27**	53515.38	120.33	129.60	1.07	-0.01	52955.34	53668.40	122.76	140.69	1.35	0.00	0.01
SJC324-30	45705.73	45769.34	45989.29	138.83	158.49	0.48	0.14	45705.73	45972.70	126.57	144.76	0.58	0.00	-0.14
SJC500-45	93486.98	93903.63	96193.51	262.59	297.49	2.44	0.45	93486.98	95708.51	326.59	376.42	2.38	0.00	-0.44
SJC500-50	81393.42	81534.02	82342.64	294.19	303.72	0.99	0.17	81393.42	82607.28	326.17	355.17	1.49	0.00	-0.17
SJC708-70	125816.01	**123940.28**	128465.74	748.41	799.92	3.65	-1.49	125816.01	128236.76	625.21	708.24	1.92	0.00	1.51
SJC708-80	102389.2	**102317.94**	102904.39	670.49	696.75	0.57	-0.07	102389.20	103269.81	587.21	659.78	0.86	0.00	0.07
SJC818-85	131413.24	**129909.98**	133822.23	817.55	851.40	3.01	-1.14	131413.24	133977.13	746.33	800.73	1.95	0.00	1.16
SJC818-90	121736.45	**121263.24**	123126.46	847.90	882.63	1.54	-0.39	121736.45	123268.88	759.99	826.83	1.26	0.00	0.39
lin318-005	180563.96	180563.96	180563.96	17.77	35.33	0.00	0.00	180563.96	180563.96	35.61	126.69	0.00	0.00	0.00
lin318-015	89595.26	89595.26	89633.18	69.58	87.79	0.04	0.00	89595.26	89645.96	100.59	130.37	0.06	0.00	0.00
lin318-040	47982.41	**47967.06**	48377.88	191.24	223.44	0.86	-0.03	47982.41	48296.35	141.11	168.18	0.65	0.00	0.03
lin318-070	33069.9	**33051.90**	33457.98	320.94	351.20	1.23	-0.05	33069.90	33537.57	187.11	216.38	1.41	0.00	0.05
lin318-100	24611.21	**24425.05**	24882.73	427.96	449.62	1.87	-0.76	24611.21	24889.75	218.45	246.34	1.13	0.00	0.76
rl1304-010	2143153.4	**2143001.65**	2144270.98	45.65	204.15	0.06	-0.01	2143153.40	2151148.99	136.21	322.04	0.37	0.00	0.01
rl1304-050	794874.33	795554.62	798880.12	782.89	877.07	0.42	0.09	794874.33	798301.54	816.88	873.00	0.43	0.00	-0.09
rl1304-100	501282.64	**499842.11**	503829.13	982.59	1003.30	0.80	-0.29	501282.64	504579.71	951.80	1001.88	0.66	0.00	0.29
rl1304-200	294324	**293564.58**	300734.14	936.65	1003.90	2.44	-0.26	294324.00	300878.85	950.44	1004.03	2.23	0.00	0.26
rl1304-300	218652.29	219918.00	228139.04	925.97	1007.29	3.74	0.58	218652.29	228433.84	922.15	1005.11	4.47	0.00	-0.58
u724-010	181038.63	181038.63	181044.48	103.74	110.47	0.00	0.00	181038.63	181069.90	451.64	610.14	0.02	0.00	0.00
u724-030	95147.79	95138.96	95394.96	256.74	298.32	0.27	-0.01	95147.79	95431.89	604.14	645.34	0.30	0.00	0.01
u724-075	55351.09	55150.82	55330.67	775.83	798.58	0.33	-0.36	55351.09	55623.35	606.99	692.06	0.49	0.00	0.36
u724-125	39900.38	39869.77	40190.17	962.67	997.58	0.80	-0.08	39900.38	40397.59	774.81	866.08	1.25	0.00	0.08
u724-200	29637.64	29441.00	30047.99	958.11	1002.54	2.06	-0.66	29637.64	29875.26	924.76	1001.36	0.80	0.00	0.67
pr2392-020	2230931.22	2230931.22	2231014.44	514.53	621.73	0.00	0.00	2230931.22	2232952.79	943.49	978.91	0.09	0.00	0.00
pr2392-075	1097300.86	**1091998.14**	1098152.22	778.47	1004.36	0.56	-0.48	1097300.86	1106598.72	928.20	1008.72	0.85	0.00	0.49
pr2392-150	741147.58	**722251.66**	730491.91	734.49	1010.52	1.14	-2.55	741147.58	756309.74	871.60	1006.67	2.05	0.00	2.62
pr2392-300	509286.57	**481408.68**	486667.81	780.57	1024.05	1.09	-5.47	509286.57	520158.12	624.27	1016.64	2.13	0.00	5.79
pr2392-500	375707.76	**347586.07**	354164.18	621.92	1024.93	1.89	-7.48	375707.76	380956.32	745.46	1019.98	1.40	0.00	8.09
fnl4461-0020	1282694.8	**1282595.75**	1282699.48	734.76	1001.41	0.01	-0.01	1282694.80	1285027.13	929.17	1069.74	0.18	0.00	0.01
fnl4461-0100	560148.77	**551182.37**	552314.84	515.29	1011.87	0.21	-1.60	564817.89	566854.91	596.01	1010.15	0.36	0.83	2.47
fnl4461-0250	348101.42	**342449.58**	343412.93	700.97	1033.50	0.28	-1.62	355562.03	357274.11	949.10	1031.81	0.48	2.14	3.83
fnl4461-0500	237491.7	**233741.56**	235384.09	692.94	1050.64	0.70	-1.58	274599.97	279755.91	814.19	1026.39	1.88	15.63	17.48
fnl4461-1000	159672.99	159935.82	160843.50	818.36	1085.12	0.57	0.16	201733.22	204655.69	940.87	1020.42	1.45	26.34	26.13
average	394744.39	391630.06	393826.74	562.15	679.96	1.06	-0.75	397511.01	400603.86	599.57	700.33	1.12	1.36	2.16

In summary, A-BRKGA_INLS provides stronger robustness and better results than A-BRKGA+CS. A-BRKGA_INLS scans the neighborhood better, so it produces high-quality solutions.

We also compare A-BRKGA_INLS with A-BRKGA to show the efficiency of the local search components INLS. Results are listed in Table 6.

$$Diff' = \frac{sol^*_{\text{A-BRKGA}} - sol^*_{INLS}}{sol^*_{INLS}} * 100$$

Compared with A-BRKGA, INLS is able to improve the results for 28 of 33 instances, with an average difference of 3.88%. The average gap decreases from 3.02% to -0.75%. The average deviations of 33 tested instances of A-BRKGA_INLS and A-BRKGA are 1.06% and 1.50%, respectively. In relation of computational time, A-BRKGA ($T = 618.74s$) is faster than A-BRKGA_IVNS ($T = 679.96s$) due to the local search component. The reported data for these instances supports the claims that INLS generates high-quality solutions.

Random seeds affect the searched solutions through the evolutionary process. In order to study the effect of random seeds on the best solution, we increased the number of consecutive runs to 50. The current best solution was recorded every five times. As shown in Figure 5, each recorded data is compared with the best result of 20 consecutive runs (sol^* of A-BRKGA_INLS in Table 5). In the u724_010 and u724_030 instances, A-BRKGA_INLS obtained the best solution in the 0-5th and 10th-15th time, respectively. In the u724_075, u724_125 and u724_200

TABLE 6. The local search component usage comparison test in 33 CCCP benchmark instances.

instances	*A-BRKGA_INLS*						*A-BRKGA*						*Diff'*
	*sol**	*sol*	*T**	*T*	*Dev*	*Gap*	*sol**	*sol*	*T**	*T*	*Dev*	*Gap*	
SJC324-25	**52949.27**	53515.38	120.33	129.60	1.07	-0.01	53183.83	53779.79	96.35	103.24	1.12	0.43	0.44
SJC324-30	45769.34	45989.29	138.83	158.49	0.48	0.14	45725.49	46028.10	113.28	122.98	0.66	0.04	-0.10
SJC500-45	**93903.63**	96193.51	262.59	297.49	2.44	0.45	95285.56	97571.72	221.08	226.38	2.40	1.92	1.47
SJC500-50	**81534.02**	82342.64	294.19	303.72	0.99	0.17	81873.84	82405.72	258.45	262.34	0.65	0.59	0.42
SJC708-70	**123940.28**	128465.74	748.41	799.92	3.65	-1.49	127417.31	131896.11	445.11	449.34	3.52	1.27	2.81
SJC708-80	**102317.94**	102904.39	670.49	696.75	0.57	-0.07	102786.07	104012.62	529.06	533.19	1.19	0.39	0.46
SJC818-85	**129909.98**	133822.23	817.55	851.40	3.01	-1.14	132499.81	135232.39	616.89	621.53	2.06	0.83	1.99
SJC818-90	**121263.24**	123126.46	847.90	882.63	1.54	-0.39	122651.29	123993.99	665.78	668.88	1.09	0.75	1.14
lin318-005	180563.96	180563.96	17.77	35.33	0.00	0.00	180563.96	180563.96	22.53	45.67	0.00	0.00	0.00
lin318-015	89595.26	89633.18	69.58	87.79	0.04	0.00	89596.10	89656.52	68.34	81.04	0.07	0.00	0.00
lin318-040	**47967.06**	48377.88	191.24	223.44	0.86	-0.03	48149.30	48375.23	146.13	152.07	0.47	0.35	0.38
lin318-070	**33051.90**	33457.98	320.94	351.20	1.23	-0.05	33521.15	33833.09	206.31	212.24	0.93	1.36	1.42
lin318-100	**24425.05**	24882.73	427.96	449.62	1.87	-0.76	24812.71	25254.38	237.86	240.47	1.78	0.82	1.59
rl1304-010	**2143001.65**	2144270.98	45.65	204.15	0.06	-0.01	2144630.08	2163800.92	225.64	244.15	0.89	0.07	0.08
rl1304-050	**795554.62**	798880.12	782.89	877.07	0.42	0.09	796913.54	800546.75	755.37	760.54	0.46	0.26	0.17
rl1304-100	**499842.11**	503829.13	982.59	1003.30	0.80	-0.29	507419.51	510188.94	996.00	1002.91	0.55	1.22	1.52
rl1304-200	**293564.58**	300734.14	936.65	1003.90	2.44	-0.26	302406.94	310501.65	997.57	1004.81	2.68	2.75	3.01
rl1304-300	**219918.00**	228139.04	925.97	1007.29	3.74	0.58	231739.11	245553.66	995.58	1006.74	5.96	5.99	5.38
u724-010	181038.63	181044.48	103.74	110.47	0.00	0.00	181038.63	181093.38	132.69	141.92	0.03	0.00	0.00
u724-030	**95138.96**	95394.96	256.74	298.32	0.27	-0.01	95369.57	95543.26	277.87	286.36	0.18	0.23	0.24
u724-075	**55150.82**	55330.67	775.83	798.58	0.33	-0.36	55554.14	56118.12	564.12	568.94	1.02	0.37	0.73
u724-125	**39869.77**	40190.17	962.67	997.58	0.80	-0.08	40350.32	40788.64	805.55	809.94	1.09	1.13	1.21
u724-200	**29441.00**	30047.99	958.11	1002.54	2.06	-0.66	29691.46	30343.36	995.83	1001.27	2.20	0.18	0.85
pr2392-020	2230931.22	2231014.44	514.53	621.73	0.00	0.00	2230939.06	2232799.36	698.33	711.94	0.08	0.00	0.00
pr2392-075	**1091998.14**	1098152.22	778.47	1004.36	0.56	-0.48	1101365.79	1109903.32	996.02	1003.81	0.78	0.37	0.86
pr2392-150	**722251.66**	730491.91	734.49	1010.52	1.14	-2.55	753531.44	781787.89	985.55	1008.85	3.75	1.67	4.33
pr2392-300	**481408.68**	486667.81	780.57	1024.05	1.09	-5.47	553214.29	573891.61	975.74	1009.34	3.74	8.63	14.92
pr2392-500	**347586.07**	354164.18	621.92	1024.93	1.89	-7.48	438626.13	453559.56	959.38	1029.70	3.40	16.75	26.19
fnl4461-0020	**1282595.75**	1282699.48	734.76	1001.41	0.01	-0.01	1282810.10	1283180.94	989.79	1002.03	0.03	0.01	0.02
fnl4461-0100	**551182.37**	552314.84	515.29	1011.87	0.21	-1.60	567218.76	577815.70	878.68	1010.22	1.87	1.26	2.91
fnl4461-0250	**342449.58**	343412.93	700.97	1033.50	0.28	-1.62	376257.53	381010.28	860.70	1026.72	1.26	8.09	9.87
fnl4461-0500	**233741.56**	235384.09	692.94	1050.64	0.70	-1.58	274599.97	279755.91	810.33	1021.21	1.88	15.63	17.48
fnl4461-1000	**159935.82**	160843.50	818.36	1085.12	0.57	0.16	201733.22	205056.04	854.18	1047.80	1.65	26.34	26.13
average	391630.06	393826.74	562.15	679.96	1.06	-0.75	403135.64	408055.85	587.34	618.74	1.50	3.02	3.88

Note: The values in bold are better than the best solutions of A-BRKGA.

instances, A-BRKGA_INLS obtained better solutions than those reported in Table 4 in the 10-15th, 20-25th, 35-40th time, respectively. It can be seen that affected by the random seed, A-BRKGA_INLS can perform better as the number of runs increases.

In addition, literature [11] applied the proposed method GB21^{MH} to CCCP and gave experimental results. Table 7 shows the comparative experimental results of A-BRKGA_ INLS and GB21^{MH}. Experiments show that our method obtains a better solution in eight instances, although the solution of GB21^{MH} in some instances is greatly improved.

B. EFFECTIVENESS ANALYSIS OF INEXACT SEARCH

The final exact search method can significantly improve the quality of solutions. However, it does not mean that applying the exact search method to evolution can also produce excellent solutions. To prove it, we compare A-BRKGA_INLS with the method not using the inexact search.

In Table 8, the second set of data shows experimental results of not using the inexact search in 33 CCCP benchmark instances. (In each column, the three sets of data represent A-BRKGA_INLS, not using the inexact search and not using iterative neighborhood. The values in bold are better than the

TABLE 7. Comparison of computational results with GB21.

instances	GB21^{MH}	A-BRKGA_INLS	*Gap*
sjc1	17,363.47	17359.75	-0.02%
sjc2	33,425.61	33181.65	-0.73%
sjc3a	45,470.41	45354.29	-0.26%
sjc3b	40,839.40	40660.55	-0.44%
sjc4a	62,030.00	61948.21	-0.13%
sjc4b	52,551.74	52196	-0.68%
fnl4461-0020	1,286,767.42	1282595.75	-0.32%
fnl4461-0100	551,405.33	551182.37	-0.04%
fnl4461-0250	337,433.96	342449.58	1.49%
fnl4461-0500	225,789.57	233741.56	3.52%
fnl4461-1000	148,551.56	159935.82	7.66%

TABLE 8. Inexact search and iterative neighborhood usage comparison test.

instances	*best-known*	*sol**	*sol*	*T**	*T*
SJC324-25	52955.34	52949.27/53116.64/52965.80	53515.38/53375.00/53549.12	120.33/51.4/91.07	129.60/152.65/96.77
SJC324-30	45705.73	45769.34/45798.24/45801.69	45989.29/45960.88/45945.85	138.83/58.72/103.49	158.49/171.38/121.52
SJC500-45	93486.98	93903.63/93994.42/93696.93	96193.51/95661.48/96212.91	262.59/2.23/204.00	297.49/292.69/224.17
SJC500-50	81393.42	81534.02/81657.98/81696.41	82342.64/82228.75/82244.32	294.19/2.67/241.86	303.72/336.37/246.20
SJC708-70	125816.01	123940.28/**124380.45**/**125660.58**	128465.74/127559.39/128719.58	748.41/6.23/434.03	799.92/720.04/494.86
SJC708-80	102389.20	102317.94/102446.61/**102214.79***	102904.39/103202.18/103083.18	670.49/6.21/525.68	696.75/751.97/572.46
SJC818-85	131413.24	129909.98/**130798.14**/**130687.92**	133822.23/133111.41/132095.98	817.55/8.34/691.07	851.40/915.87/722.43
SJC818-90	121736.45	121263.24/**121558.30**/**121345.63**	123126.46/122019.34/122315.16	847.90/8.82/675.04	882.63/954.79/705.53
lin318-005	180563.96	180563.96/180563.96/180563.96	180563.96/180563.96/180563.96	17.77/0.27/9.95	35.33/37.67/31.27
lin318-015	89595.26	89595.26/89622.57/89621.35	89633.18/89641.93/89647.21	69.58/23.03/36.97	87.79/87.09/64.60
lin318-040	47982.41	47967.06/48079.06/48005.06	48377.88/48282.77/48239.80	191.24/1.33/130.34	223.44/202.52/151.17
lin318-070	33069.90	33051.90/33192.85/33242.91	33457.98/33524.83/33418.09	320.94/22.15/222.12	351.20/299.36/235.12
lin318-100	24611.21	24425.05/24840.39/**24474.09**	24882.73/25032.63/24743.25	427.96/33.45/344.79	449.62/392.67/388.69
rl1304-010	2143153.40	2143001.65/2143282.16/2143239.35	2144270.98/2144073.44/2144619.36	45.65/0.99/57.24	204.15/276.50/207.92
rl1304-050	794874.33	795554.62/795552.94/**794450.25***	798880.12/797247.74/796790.28	782.89/6.53/922.34	877.07/871.87/997.52
rl1304-100	501282.64	499842.11/**499004.77***/**501101.01**	503829.13/501345.26/503676.69	982.59/12.70/1278.85	1003.30/1005.19/1403.43
rl1304-200	294324.00	293564.58/**286588.98***/295794.22	300734.14/288528.76/304110.34	936.65/21.57/1268.87	1003.90/1006.00/1406.30
rl1304-300	218652.29	219918.00/224148.49/221126.37	228139.04/227623.71/229283.12	925.97/23.51/1226.99	1007.29/1009.67/1407.85
u724-010	181038.63	181038.63/181038.63/181038.63	181044.48/181053.59/181093.90	103.74/49.9/72.08	110.47/149.03/94.27
u724-030	95147.79	95138.96/95177.72/95301.13	95394.96/95309.26/95436.57	256.74/2.04/229.64	298.32/371.71/243.96
u724-075	55351.09	55150.82/**55246.39**/**55335.04**	55330.67/55425.32/55531.08	775.83/5.83/600.24	798.58/774.74/626.64
u724-125	39900.38	39869.77/39905.14/**39846.80***	40190.17/40122.06/40199.49	962.67/10.82/888.24	997.58/1008.61/925.50
u724-200	29637.64	29441.00/**29530.33**/**29185.81***	30047.99/29730.21/29485.29	958.11/13.69/1032.4	1002.54/1004.57/1068.10
pr2392-020	2230931.22	2230931.22/2231237.82/2230939.45	2231014.44/2232206.24/2232102.52	514.53/5.31/504.18	621.73/1007.84/526.63
pr2392-075	1097300.86	1091998.14/**1089789.86***/**1094369.47**	1098152.22/1093552.49/1098722.82	778.47/21.29/906.41	1004.36/1012.43/1004.42
pr2392-150	741147.58	722251.66/**723146.58**/**730129.66**	730491.91/727508.75/735487.52	734.49/48.99/575.35	1010.52/1021.30/1010.51
pr2392-300	509286.57	481408.68/**481879.12**/**491160.09**	486667.81/483144.1/494039.63	780.57/94.01/794.78	1024.05/1020.53/1009.19
pr2392-500	375707.76	347586.07/**350686.80**/**354622.08**	354164.18/353130.26/361016.47	621.92/527.9/704.51	1024.93/1081.26/1028.38
fnl4461-0020	1282694.80	1282595.75/1282618.24/**1282685.78**	1282699.48/1282657.42/1282817.73	734.76/7.42/708.21	1001.41/1081.79/919.88
fnl4461-0100	560148.77	551182.37/**550770.55***/**553010.93**	552314.84/551291.24/553971.40	515.29/40.66/490.97	1011.87/1077.57/1011.01
fnl4461-0250	348101.42	342449.58/**342241.80***/**346337.47**	343412.93/342559.74/346918.53	700.97/77.92/575.3	1033.50/1075.47/1043.13
fnl4461-0500	237491.70	233741.56/**233734.21**/238936.81	235384.09/233921.12/239702.46	692.94/237.94/773.42	1050.64/1089.87/1043.01
fnl4461-1000	159672.99	159935.82/**159553.98***/164487.76	160843.50/160168.85/165762.59	818.36/726.26/952.02	1085.12/1121.80/1060.70
average	394744.39	391630.06/370267.93/414047.50	393826.74/392750.43/394895.34	562.15/65.46/553.71	679.96/708.57/669.49

Note: The meaning of data in the form of A/B/C in Table VIII is: A represents A-BRKGA_INLS, B represents not using the inexact search, and C represents not using iterative neighborhood.

best-known solutions, and the values marked with * are better than A-BRKGA_INLS.) Each instance was run 20 times continuously. As shown in Table 8, not using the inexact search outperforms the best-known solutions in 15 instances but outperforms A-BRKGA_INLS solutions in only six instances. In most instances, the best solutions of not using the inexact search are worse than those of A-BRKGA_INLS. Not using the inexact search can get some excellent solutions using iterative neighborhood, but it is not as excellent as A-BRKGA_INLS in the overall result. Moreover, the average running time to find the best solution for not using the inexact search ($T* = 65.4s$) is much less than A-BRKGA_INLS

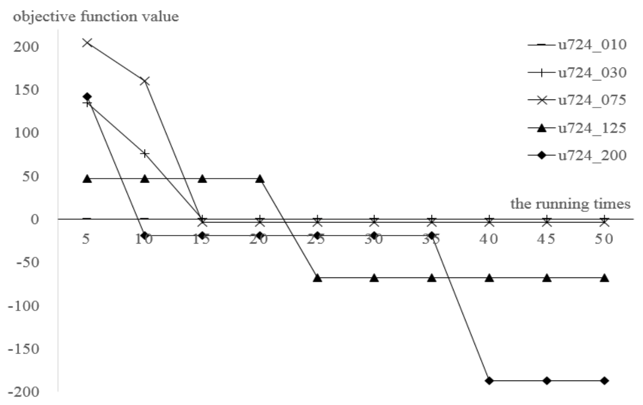


FIGURE 5. Improvement of A-BRKGA_INLS in the u724 dataset for 50 continuous runs.

($T^* = 562.15s$). Not using the inexact search reaches local optimum within tens of seconds in most instances, and results are not improved at the later stage of evolution.

The reported results support the claims that the inexact search effectively avoids the algorithm falling into local optimum prematurely and helps the algorithm yield high-quality solutions.

C. EFFECTIVENESS ANALYSIS OF ITERATIVE NEIGHBORHOOD

Both IINLS and EINLS iteratively search the shift neighborhood and the swap neighborhood, instead of searching for both shifts and swaps simultaneously. In other words, IINLS and EINLS search the shift neighborhood until there is no feasible shift and then go to the swap neighborhood to perform feasible swaps. Since the CCCP is strictly limited by capacity, the order of shifting and swapping is very important. After a large number of shifts, INLS searches swaps that cannot be converted into two shifts due to capacity constraints. Based on the optimal shift neighborhood, INLS searches swaps to make the swap neighborhood optimal. In INLS, the swap neighborhood does not affect the shift neighborhood. The shift neighborhood has more adjustment space and can develop in a better direction. However, if shifts and swaps are searched alternately frequently, the solution is not optimal in any neighborhood in the whole process, except at the end. In order to prove that iterative neighborhood search provides better results than simultaneous search, we conducted a comparative experiment.

The third set of data in each column of Table 8 shows the experimental results of not using iterative neighborhood. Similarly, each instance was run 20 times continuously. Not using iterative neighborhood utilizes the A-BRKGA to evolve population but performs two local search algorithms that simultaneously search for swaps and shifts. Not using iterative neighborhood combines inexact and exact search, the same as A-BRKGA_INLS. In 13 instances, not using iterative neighborhood finds better solutions than the best-known solutions, but these solutions are not as excellent as A-BRKGA_INLS solutions. In one instance, the solution outperforms A-BRKGA_INLS but is worse than the best-known solution. In four instances (bold and marked with *), not using iterative neighborhood outperforms both the best-known solutions and A-BRKGA_INLS solutions. Solutions of other instances are poor. As shown in the last row of Table 8, the average value of the best solutions of A-BRKGA_INLS ($average(sol^*) = 391639.35$) is much smaller than that of not using iterative neighborhood ($average(sol^*) = 414047.50$).

In summary, A-BRKGA_INLS iteratively searches for the neighborhood space of solutions, which has more advantages than not using iterative neighborhood on these data sets. Therefore, iterative neighborhood search is effective.

VI. CONCLUSION

This paper presents an optimized local search algorithm based on A-BRKGA to solve the CCCP. Unlike local search processes in the previous literature, A-BRKGA_INLS iteratively searches the shift neighborhood and the swap neighborhood. Until there are no viable shifts in the shift neighborhood, the swap neighborhood is searched. Thus, A-BRKGA_INLS explores the neighborhood of solutions more fully. In the stage of population evolution, evaluation functions are used to perform the inexact local search to avoid the search falling into the local optimum prematurely. When the population evolution is completed, precise calculations are adopted to further improve the quality of solutions. The performance of the algorithm was tested on a general benchmark containing 53 instances.

Experimental results show that the algorithm is effective for solving CCCP. Based on 53 instances, 23 new best-known solutions are provided, and 15 instances match the current best-known solutions. Compared with A-BRKGA + CS, the difference in time cost between the two algorithms is small.

Measuring the performance of the overlapping areas is our next research issue. Future research can focus on heuristic algorithms for data sets with tight cluster capacity limits such as x-n-m, so the algorithm can fully explore the neighborhood space on the premise of constructing a small number of feasible solutions. Second, multi-objective CCCP can be another research direction to expand the CCCP model to a broader range of applications.

REFERENCES

[1] X. Wang and X. Qin, "Asymmetric intimacy and algorithm for detecting communities in bipartite networks," *Phys. A, Stat. Mech. Appl.*, vol. 462, pp. 569–578, Nov. 2016.

[2] Y. Cui and X. Wang, "Detecting one-mode communities in bipartite networks by bipartite clustering triangular," *Phys. A, Stat. Mech. Appl.*, vol. 457, pp. 307–315, Sep. 2016.

[3] S. L. Hakimi, "Optimum locations of switching centers and the absolute centers and medians of a graph," *Oper. Res.*, vol. 12, no. 3, pp. 450–459, Jun. 1964.

[4] J. M. Mulvey and M. P. Beck, "Solving capacitated clustering problems," *Eur. J. Oper. Res.*, vol. 18, pp. 339–348, Dec. 1984.

[5] M. Negreiros and A. Palhano, "The capacitated centred clustering problem," *Comput. Oper. Res.*, vol. 33, no. 6, pp. 1639–1663, Jun. 2006.

[6] F. Boudahri, M. Bennekrouf, F. Belkaid, and Z. Sari, "Application of a capacitated centered clustering problem for design of agri-food supply chain network," *Int. J. Comput. Sci. Issues*, vol. 9, no. 4, pp. 304–1300, Jul. 2012.

[7] A. C. Pillai, J. Chick, L. Johanning, M. Khorasanchi, and V. de Laleu, "Offshore wind farm electrical cable layout optimization," *Eng. Optim.*, vol. 47, no. 12, pp. 1689–1708, Dec. 2015.

[8] C.-A. Chou, W. A. Chaovalitwongse, T. Y. Berger-Wolf, B. DasGupta, and M. V. Ashley, "Capacitated clustering problem in computational biology: Combinatorial and statistical approach for sibling reconstruction," *Comput. Oper. Res.*, vol. 39, no. 3, pp. 609–619, Mar. 2012.

[9] F. Stefanello, O. C. B. de Arújo, and F. M. Müller, "Matheuristics for the capacitated p-median problem," *Int. Trans. Oper. Res.*, vol. 22, pp. 149–167, Jan. 2015.

[10] P. Baumann, "A binary linear programming-based K-means approach for the capacitated centered clustering problem," in *Proc. IEEE Int. Conf. Ind. Eng. Manage. (IEEM)*, Dec. 2019, pp. 335–339.

[11] M. Gnägi and P. Baumann, "A matheuristic for large-scale capacitated clustering," *Comput. Oper. Res.*, vol. 132, Aug. 2021, Art. no. 105304.

[12] F. Mai, M. J. Fry, and J. W. Ohlmann, "Model-based capacitated clustering with posterior regularization," *Eur. J. Oper. Res.*, vol. 271, no. 2, pp. 594–605, Dec. 2018.

[13] L. Jánošíková, M. Herda, and M. Haviar, "Hybrid genetic algorithms with selective crossover for the capacitated p-median problem," *Central Eur. J. Oper. Res.*, vol. 25, no. 3, pp. 651–664, Sep. 2017.

[14] A. A. Chaves and L. A. N. Lorena, "Clustering search algorithm for the capacitated centered clustering problem," *Comput. Oper. Res.*, vol. 37, no. 3, pp. 552–558, Mar. 2010.

[15] A. C. M. Oliveira and L. A. N. Lorena, "Hybrid evolutionary algorithms and clustering search," in *Hybrid Evolutionary Algorithms*. Berlin, Germany: Springer, 2007, pp. 77–99.

[16] A. A. Chaves and L. A. Nogueira Lorena, "Hybrid evolutionary algorithm for the capacitated centered clustering problem," *Expert Syst. Appl.*, vol. 38, no. 5, pp. 5013–5018, May 2011.

[17] A. Einstein, F. Muritiba, M. Negreiros, M. F. De Souza, and A. Min-Sum, "A Tabu search algorithm for the capacitated centred clustering problem," in *Proc. 44th Symp. Brazilian Oper. Res. Soc.*, Rio de Janeiro, Brazil, 2012, pp. 2344–2359.

[18] D. Melo Morales, A. A. Chaves, and A. L. Fazenda, "Parallel clustering search applied to capacitated centered clustering problem," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2019, pp. 542–548.

[19] S.-O. Caballero-Morales, E. Barojas-Payan, D. Sanchez-Partida, and J.-L. Martinez-Flores, "Extended GRASP-capacitated K-means clustering algorithm to establish humanitarian support centers in large regions at risk in Mexico," *J. Optim.*, vol. 2018, pp. 1–14, Dec. 2018.

[20] A. A. Chaves, J. F. Gonçalves, and L. A. N. Lorena, "Adaptive biased random-key genetic algorithm with local search for the capacitated centered clustering problem," *Comput. Ind. Eng.*, vol. 124, pp. 331–346, Oct. 2018.

[21] J. F. Gonçalves and M. G. Resende, "Biased random-key genetic algorithms for combinatorial optimization," *J. Heuristics*, vol. 17, no. 5, pp. 487–525, Oct. 2011.

[22] A. Mjirda, R. Todosijević, S. Hanafi, P. Hansen, and N. Mladenović, "Sequential variable neighborhood descent variants: An empirical study on the traveling salesman problem," *Int. Trans. Oper. Res.*, vol. 24, no. 3, pp. 615–633, May 2017.

[23] J. Brimberg, N. Mladenović, and D. Urošević, "Solving the maximally diverse grouping problem by skewed general variable neighborhood search," *Inf. Sci.*, vol. 295, pp. 650–675, Feb. 2015.

[24] Q. Zhou, U. Benlic, Q. Wu, and J.-K. Hao, "Heuristic search to the capacitated clustering problem," *Eur. J. Oper. Res.*, vol. 273, no. 2, pp. 464–487, Mar. 2019.

[25] I. Kaabachi, H. Yahyaoui, S. Krichen, and A. Dekdouk, "Measuring and evaluating hybrid metaheuristics for solving the multi-compartment vehicle routing problem," *Measurement*, vol. 141, pp. 407–419, Jul. 2019.

[26] J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization," *ORSA J. Comput.*, vol. 6, no. 2, pp. 154–160, May 1994.

[27] C. E. Andrade, T. Silva, and L. S. Pessoa, "Minimizing flowtime in a flowshop scheduling problem with a biased random-key genetic algorithm," *Expert Syst. Appl.*, vol. 128, pp. 67–80, Aug. 2019.

[28] F. L. Biajoli, A. A. Chaves, and L. A. N. Lorena, "A biased random-key genetic algorithm for the two-stage capacitated facility location problem," *Expert Syst. Appl.*, vol. 115, pp. 418–426, Jan. 2019.

[29] L. S. Pessoa, A. C. Santos, and M. G. C. Resende, "A biased random-key genetic algorithm for the tree of hubs location problem," *Optim. Lett.*, vol. 11, no. 7, pp. 1371–1384, Oct. 2017.

[30] H. Prasetyo, A. L. Putri, and G. Fauza, "Biased random key genetic algorithm design with multiple populations to solve capacitated vehicle routing problem with time windows," in *Proc. AIP Conf.*, 2018, pp. 1–11.

[31] W. M. Spears and K. A. De Jong, "On the virtues of parameterized uniform crossover," in *Proc. 4th Int. Conf. Genetic Algorithms*, San Diego, CA, USA, 1991, pp. 230–236.

[32] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 76, no. 3, pp. 1–11, Sep. 2007.

[33] J. Lee Rodgers and W. A. Nicewander, "Thirteen ways to look at the correlation coeffificient," *Amer. Statistician*, vol. 42, no. 1, pp. 59–66, Jun. 1998.

[34] L. Lorena and E. Senne, "Local search heuristics for capacitated p-median problems," *Netw. Spat. Econ.*, vol. 3, no. 4, pp. 407–419, Dec. 2003.

[35] M. A. Pereira, L. A. N. Lorena, and E. L. F. Senne, "A column generation approach for the maximal covering location problem," *Int. Trans. Oper. Res.*, vol. 14, no. 4, pp. 349–364, Jul. 2007.

[36] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian, "New benchmark instances for the capacitated vehicle routing problem," *Eur. J. Oper. Res.*, vol. 257, no. 3, pp. 845–858, Mar. 2017.

YUYING XU received the B.S. degree from the College of Computer Science, Chongqing University, China, in 2019, where she is currently pursuing the M.S. degree. Her main research interests include evolutionary computing and membrane computing.

PING GUO (Member, IEEE) received the Ph.D. degree in computer software and theory from Chongqing University, China, in 2004. Currently, he is a Professor with the Chongqing Key Laboratory of Software Theory and Technology, College of Computer Science of Chongqing University, Chongqing, China. He has authored/coauthored more than 180 refereed publications. His research interests include different aspects of artificial intelligence and biological computing.

YI ZENG was born in Wulanchabu, Inner Mongolia, China, in 1961. He received the bachelor's degree in computer software and theory from Wuhan University, China, in 1982. He is currently a Professor with the College of Computer Science, Chongqing University, Chongqing, China. He has authored or coauthored more than 70 refereed publications. His research interests include different aspects of software engineering, software testing, and management information systems.