An Iterative Neighborhood Local Search Algorithm for Capacitated Centered Clustering Problem

Yuying Xu¹, Ping Guo¹,²,* (Member, IEEE), and Yi Zeng¹,²
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 (e-mail: 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-BRKGALNS. 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-BRKGALNS 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-BRKGALNS+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 constrained clustering problem in the clustering problem, which is an abstract problem of location selection, and it 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 a wide range of 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 so that 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 so that 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, while for 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 CPMP and CCCP are similar, researchers often propose solutions to
CPMP and apply them to CCCP after making slight changes. In recent years, there have been many heuristic and meta-heuristic solutions for CPMP and CCCP.

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 solution was 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 determined new best solutions for seven classic instances and reported 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 combine simulated annealing algorithm, genetic algorithm and tabu search with local search. [18] proposes the strategy of parallel local search component. The related research on the local search for a specific solution of CCCP is relatively shallow, such as literature [17] and [20] are 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 a certain 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 or 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] adopts particular strategies and mechanisms to retain and deal with infeasible solutions to explore infeasible space. 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 searching for feasible shifts and swaps simultaneously. 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 checked again. The shift neighborhood has more adjustment space and can develop in a better direction.

2. The algorithm combines imprecise search and precise search. The imprecise 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 precise search checks the shifts and swaps ignored by the imprecise 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).

$$\text{minimize} \sum_{i \in I} \sum_{j \in J} ||a_i - \bar{x}_j|| y_{ij}$$  \hspace{1cm} (1)$$

$$\sum_{j \in J} y_{ij} = 1, \forall i \in I$$  \hspace{1cm} (2)$$

$$\sum_{i \in I} q_i y_{ij} \leq Q_j, \forall j \in J$$  \hspace{1cm} (3)$$

$$x_j \in \mathbb{R}^2, n_j \in \mathbb{N}, y_{ij} \in \{0, 1\}, \forall i \in I, \forall j \in J$$

$I$ 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.

![BRKGA: generation t+1 from generation t.](image)

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 $p$, $p - p_e - pm$ 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 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 probability 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 is always selecting 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 imprecise Iterative Neighborhood Local Search Algorithm (INLS) is explained. Finally, the precise Iterative Neighborhood Local Search Algorithm (PINLS) is introduced, and the difference between IINLS and PINLS is discussed.

A. ALGORITHM FRAMEWORK

| Algorithm 1: \(\text{A-BRKGA\_INLS;}\) //A-BRKGA with iterative neighborhood local search |
| --- |
| **Input:** the number of generation \(\text{max\_gen}\), 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 \(S \rightarrow \text{+}\); |
| (1) Randomly generate initial population \(\text{Pop}\) with \(p\) vectors; |
| (2) for \(i = 1\) to \(\text{max\_gen}\) do |
| (3) \(\text{Pop} \leftarrow \text{population\_update}(\text{Pop});\) // Ref. A-BRKGA |
| (4) if \((i\%K = 0)\) then |
| (5) \(\text{promisingPop} \leftarrow \text{Lable\_Propagation}(\text{Pop});\) // Ref. [20] |
| (6) for each solution \(S\) in \(\text{promisingPop}\) do |
| (7) \(S_{\text{searched}} \leftarrow \text{INLS}(S);\) // Imprecise Iterated Neighborhood Local Search |
| (8) if \((S_{\text{searched}} \leq S\) then \(S \rightarrow S_{\text{searched}}\); |
| (9) endfor |
| (10) endif |
| (11) endfor |
| (12) \(S \rightarrow \text{PINLS}(S);\) // Precise Iterated Neighborhood Local Search |
| (13) end |

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 (PINLS) is performed when the population evolution is finished to improve the search quality. PINLS 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 precise local search algorithm PINLS is implemented to get the 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 PINLS 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 I shows the coordinates and points demand. f denotes the value of the objective function. For the initial solution in Figure 2(a), we use the precise iterative neighborhood search (see (b)(c) for the process) and A-BRKGA_INLS combined with the imprecise iterative neighborhood search (see (d)(e)(f) for the process)) to solve. The initial solution f = 66.3.

**TABLE I. 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      |

Perform the precise iterative neighborhood search. First, search for the shift neighborhood of the current solution in increasing order of cluster labels. Traverse cluster 1, point 2 is moved to cluster 2, and f is reduced by 0.37. Figure 2(b) shows the updated solution. Continue to search the shift neighborhood of Figure 2(b). Traverse cluster 1, point 1 is moved to cluster 2, and f is reduced by 0.55. The solution is updated, as shown in Figure 2(c). Search for cluster 1, cluster 2, and cluster 3 in Figure 2(c). At this time, there is no better shift neighborhood solution. The shift neighborhood search ends. Because the clusters have no intersection, the swap neighborhood will not be searched. The current solution reaches the local optimum with f = 62.4. However, if an imprecise iterative neighborhood search is performed, since Δf < 0 (Equation 5), these two moves will not be performed. Until cluster 3 is traversed, point 4 in cluster 3 is moved to cluster 1 according to Δf = 5.18. Then search for the shift neighborhood of Figure 2(d), and move point 3 in cluster 1 to cluster 3 according to Δf = 2.24, as shown in Figure 2(e). The imprecise search reaches the local optimum with f = 52.2. That is to say, in this case 2(a), the result of precise iterative neighborhood search is only slightly improved. Due to capacity constraints or distance factors, the subsequent move that leads to a more significant improvement cannot be performed. However, the imprecise search can move across these points that cause a slight improvement. It is beneficial to the final solution. Figure 2(e) is the best solution in the evolution process. Finally, A-BRKGA_INLS applies precise iterative neighborhood search to Figure 2(e). Point 7 can be shifted to cluster 1 with f decreasing 2.4, but the imprecise search ignores it. Update it, and figure 2(f) 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 precise search for slight improvement is easy to fall into the local optimum early. In A-BRKGA_INLS, the imprecise search is used to improve the solution more, and then the precise search is used to improve the solution that is already good enough. By their combination, 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. IMPRECISE ITERATIVE NEIGHBORHOOD LOCAL SEARCH ALGORITHM (INLS)**

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 with getting 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 $n \times m$. 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, INLS, 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, \(A_{\text{old}}\) and \(B_{\text{old}}\) are the original cluster centers, and \(A_{\text{new}}\) and \(B_{\text{new}}\) 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 \(\Delta f\) 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).

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.

\[
\Delta f = d(A_{\text{old}}, u) + d(B_{\text{old}}, v) - d(A_{\text{new}}, v) - d(B_{\text{new}}, u) \\
\Delta f = d(A_{\text{old}}, u) - d(B_{\text{old}}, u)
\]

The process of INLS 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 \(S_1\) completely traverses searching shift neighborhood \(N1 (S_1)\), and no move can improve the solution (line 2-4), INLS updates the cluster boundaries and rectangular areas (line 5), and then goes to the swap neighborhood \(N2(S_1)\) (line 6). If a feasible swap can be performed, a new solution \(S_1+1\) is obtained. It does not immediately go to the shift neighborhood \(N1 (S_1+1)\) but continues to check the swap neighborhood \(N2 (S_1+1)\). 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 INLS does not affect the evolution of the individual in A-BRKGA.

INLS utilizes two evaluation functions to check feasible shifts and swaps, so the judgement of moves may be imprecise. This search process is called the imprecise search. Imprecise 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.

Algorithm 2: LocalSearchN1;

Input: a current solution \(S\), which is a solution in promisingPop or got from Algorithm 2 LocalSearchN1 or Algorithm 3 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 \(x \in\) cluster \(i\) 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 2 LocalSearchN1, the overlapped rectangle area of each two clusters overlap\([i][j]\]);
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;
PINLS keeps the same process as IINLS but calculates the objective function increment caused by swaps and shifts instead of using evaluation functions (in Algorithm 2 line 6, 7, Algorithm 3 line 6, 7, 8, \( \Delta f \) is not used). This step detects feasible shifts and swaps ignored by imprecise search due to the higher move conditions, improving the quality of solutions at a low time cost. The solution must undoubtedly be better or remain unchanged because PINLS accurately calculates the objective function.
point demand standard deviation \((q_{\text{Dev}})\). In Table III, 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 in the latest literature [20]. Then, comparative experiments were carried out from two perspectives to illustrate the superiority of our algorithm: the combination of imprecise search and precise search and iterative neighborhood search.

### A. ALGORITHM COMPARISON

Table IV and Table V 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 \((s_1^*)\), the average solution \((s_2)\) 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 \((\text{Gap})\), the deviation between the best solution and the average solution \((\text{Dev})\), and the difference between the best solutions of the two algorithms \((\text{Diff})\).

\[
\text{Gap} = \frac{s_1^* - \text{best-known}}{\text{best-known}} \times 100
\]

\[
\text{Dev} = \frac{s_1^* - s_2}{s_1^*} \times 100
\]

\[
\text{Diff} = \frac{s_1^*_{\text{CS}} - s_1^*_{\text{INLS}}}{s_1^*_{\text{INLS}}} \times 100
\]

\(\text{Gap}\) represents the gap between the current algorithm solution and the best-known solution, and \(\text{Dev}\) 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 IV, A-BRKGA\_INLS and A-BRKGA\_+CS have the same ability to obtain the best solutions \((\text{Diff} = 0)\), and the average gap between the best solutions and the best-known solutions is both 0.5% \((\text{Gap} = 0.5)\). Our method gets equal or better average solutions in 17 of 20 instances. Compared to A-BRKGA\_+CS \((\text{Dev} = 0.29)\), our method provides better robustness \((\text{Dev} = 0.20)\).

In Table V, 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 \((\text{Diff} = 2.16)\), and the gap with the best-known solution has decreased from \(\text{Gap} = 1.36\) to \(\text{Gap} = -0.75\). Especially on data sets pr2392 and fnl4641 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 IV, A-BRKGA\_INLS \((\text{Dev} = 1.06\) shows stronger robustness than A-BRKGA\_+CS \((\text{Dev} = 1.12)\). 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 differences between A-BRKGA\_INLS and A-BRKGA\_\+, and INLS provides better solutions than CS.

It can be seen from Table IV and Table V that A-BRKGA\_INLS has no significant increase in running time compared to A-BRKGA\_+CS.

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 VI.

\[
\text{Diff}^* = \frac{s_{1^*}^\text{A-BRKGA} - s_{1^*}^\text{INLS}}{s_{1^*}^\text{INLS}} \times 100
\]

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 \((T = 618.74\) s) is faster than A-BRKGA\_IVNS \((T = 679.96\) s) 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 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 V). In the u724_010 and u724_030 instances, A-BRKGA_INLS obtained the best solution in the 0-5th, 10th-15th time, respectively. In the u724_075, u724_125 and u724_200 instances, A-BRKGA_INLS obtained better solutions than those reported in Table IV 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 GB21MH to CCCP and gave experimental results. Table VII shows the comparative experimental results of A-BRKGA_INLS and GB21MH. Experiments show that our method obtains a better solution in eight instances, although the solution of GB21MH in some instances is greatly improved.

![Figure 5. Improvement of A-BRKGA_INLS in the u724 dataset for 50 continuous runs.](image)

B. EFFECTIVENESS ANALYSIS OF IMPRECISE SEARCH

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

In Table VIII, the second set of data shows experimental results of not using the imprecise search in 33 CCCP benchmark instances. (In each column, the three sets of data represent A-BRKGA_INLS, not using the imprecise search and not using iterative neighborhood in turn. The values in bold are better than the 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 VIII, not using the imprecise 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 imprecise search are worse than those of A-BRKGA_INLS. Not using the imprecise 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 imprecise search (T* = 65.4s) is much less than A-BRKGA_INLS (T* = 562.15s). Not using the imprecise 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 imprecise search effectively avoids the algorithm falling into local optimum prematurely and help the algorithm yield high-quality solutions.

C. EFFECTIVENESS ANALYSIS OF ITERATIVE NEIGHBORHOOD

Both IINLS and PINLS iteratively search the shift neighborhood and the swap neighborhood, instead of searching for both shifts and swaps simultaneously. In other words, IINLS and PINLS 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 VIII 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 imprecise search and precise 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 VIII, the average value of the best solutions of A-BRKGA\_INLS (\textit{average (sol')} = 391639.35) is much smaller than that of not using iterative neighborhood (\textit{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.

| 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 IV. Comparison of computational results in 20 CCCP benchmark instances

| instances | best-known | A-BRKGA_INLS | A-BRKGA+CS | Diff |
|-----------|------------|--------------|------------|------|
|           | sol        | T            | Dev        | Gap  | sol        | T            | Dev        | Gap  |
| ta25      | 1251.44    | 1251.45      | 0.21       | 5.37 | 0.00       | 1251.45      | 0.04       | 0.44 | 0.00 |
| ta50      | 4474.52    | 4474.52      | 0.13       | 8.54 | 0.00       | 4474.52      | 0.51       | 2.24 | 0.00 |
| ta60      | 5356.58    | 5356.58      | 2.19       | 8.52 | 0.00       | 5356.58      | 1.25       | 3.6  | 0.00 |
| ta70      | 6240.67    | 6240.67      | 3.09       | 9.50 | 0.00       | 6240.67      | 2.18       | 5.36 | 0.00 |
| ta80      | 5730.28    | 5730.28      | 2.46       | 12.62 | 0.00      | 5730.28      | 3.8        | 8.96 | 0.00 |
| ta90      | 9069.85    | 9069.85      | 1.23       | 10.40 | 0.00      | 9069.85      | 3.44       | 10.85 | 0.00 |
| ta100     | 8102.04    | 8102.04      | 3.62       | 13.79 | 0.00      | 8102.04      | 6.36       | 16.22 | 0.00 |
| sjc1      | 17359.75   | 17368.26     | 4.62       | 20.24 | 0.00      | 17359.75     | 10.2       | 21.73 | 0.00 |
| sjc2      | 33181.65   | 33192.32     | 33.18      | 53.32 | 0.00      | 33181.65     | 38.48      | 57.28 | 0.00 |
| sjc3a     | 45354.29   | 45464.46     | 90.97      | 125.08 | 0.00     | 45354.29     | 104.33     | 127.15 | 0.00 |
| sjc3b     | 40660.55   | 40775.23     | 100.06     | 147.81 | 0.00     | 40660.55     | 120.73     | 137.74 | 0.00 |
| sjc4a     | 61931.6    | 62133.71     | 169.00     | 190.01 | 0.03     | 61931.67     | 193.76     | 0.01 -0.02 | 0.00 |
| sjc4b     | 52196    | 52355.35     | 225.47     | 253.39 | 0.00     | 52202.48     | 75.91      | 201.21 | 0.01 |
| doni1     | 3017.37    | 3021.83      | 93.34      | 96.60  | 0.04     | 3018.17      | 131.93     | 147.19 | 0.03 |
| doni2     | 6080.7    | 6373.16      | 173.07     | 174.85 | 4.79     | 6372.81      | 370.35     | 4.80  | 0.01 |
| doni3     | 8343.49    | 8423.67      | 138.30     | 313.32 | 0.63     | 8390.58      | 418.35     | 674.28 | 0.56 |
| doni4     | 10777.64  | 10949.12     | 473.40     | 479.01 | 1.34     | 10817.87     | 188.2      | 1014.38 | 0.37 |
| doni5     | 11114.67  | 11214.91     | 735.56     | 747.56 | 0.17     | 11191.06     | 691.1      | 1006.37 | 0.69 |
| doni6     | 15453.83  | 15529.09     | 567.18     | 1005.17 | 0.49     | 15614.36     | 740.28     | 1023.04 | 1.04 |
| doni7     | 18484.13  | 18943.46     | 569.67     | 1008.63 | 2.48     | 18943.67     | 733.9      | 1060.46 | 2.49 |
| average   | 18209.05  | 18262.00     | 169.34     | 234.18 | 0.50     | 18263.53     | 304.13     | 0.50  | 0.00 |

# Table V. Comparison of computational results in 33 CCCP benchmark instances

| instances | best-known | A-BRKGA_INLS | A-BRKGA+CS | Diff |
|-----------|------------|--------------|------------|------|
| 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 |
| 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 |
| 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 |
| 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 |
| SJC708-70 | 123940.28  | 128465.74    | 748.41     | 799.92 | 3.65   | -1.49     | 125816.01 | 128236.76 | 625.21 | 708.24 | 1.92 | 0.00 |
| SJC708-80 | 102389.2   | 102317.94    | 102904.39  | 670.49 | 696.75 | 0.57       | -0.07    | 102389.2 | 103269.81 | 587.21 | 659.78 | 0.86 | 0.07 |
| instances | A-BRKGALNLS | A-BRKG | Diff* |
|-----------|-------------|--------|------|
|           | sol† | sol | T | Dev | Gap | sol† | sol | 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 |

**TABLE VI.** The local search component usage comparison test in 33 CCP benchmark instances

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
| instances | GB21 MOIL | A-BRKGAL INLS | Gap |
|-----------|-----------|---------------|-----|
| sje1      | 17,363.47 | 17359.75      | -0.02% |
| sje2      | 33,425.61 | 33181.65      | -0.73% |
| sje3a     | 45,470.41 | 45354.29      | -0.26% |

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

**TABLE VII. COMPARISON OF COMPUTATIONAL RESULTS WITH GB21 MOIL**
| instances | best-known | sol1 | sol | T | T |
|-----------|------------|------|-----|---|---|
| SJC324-25 | 52955.34   | 120.33 | 129.60 | .65 | .77 |
| SJC324-30 | 45705.73   | 138.83 | 158.49 | .38 | .52 |
| SJC500-45 | 11349.86   | 9619.39 | 9619.39 | .92 | .12 |
| SJC500-50 | 11839.42   | 8234.62 | 294.19 | .72 | .46 |
| SJC708-72 | 212865.01 | 312366.46 | 291207.95 | 33072.39 | .72 |
| SJC708-80 | 102389.20 | 102904.39 | 67045.91 | 566.14 | .46 |
| SJC818-85 | 11314.12 | 133822.13 | 81755.34 | .69 | .12 |
| SJC818-90 | 121736.45 | 84790.88 | 882.65 | .04 | .53 |
| lin318-00 | 180563.96 | 180563.96 | 17770.27 | .95 | .33 |
| lin318-01 | 189595.26 | 89633.18 | 69.58 | .36 | .97 |
| lin318-02 | 47982.41 | 48377.88 | 191.24 | .33 | .04 |
| lin318-03 | 33609.90 | 33457.98 | 320.94 | .22 | .12 |
| lin318-04 | 24425.22 | 24882.73 | 427.96 | .34 | .45 |
| r1304-03 | 2143153.40 | 2144207.98 | 45650.89 | .24 | .15 |
| r1304-05 | 79487.33 | 798880.12 | 78289.95 | .22 | .34 |
| r1304-10 | 501282.64 | 503820.93 | 982.59 | .72 | .85 |
| r1304-20 | 294324.00 | 300734.14 | 936.65 | .51 | .68 |
| r1304-30 | 218652.29 | 228139.04 | 925.97 | .23 | .99 |
| u724-01 | 101830.63 | 101804.48 | 103.74 | .49 | .97 |
| u724-02 | 95147.79 | 95394.96 | 256.74 | .04 | .29 |
| u724-03 | 55531.09 | 55530.67 | 775.83 | .53 | .06 |
| u724-07 | 39800.38 | 40190.17 | 962.67 | .10 | .88 |
| u724-12 | 29637.64 | 30047.99 | 958.11 | .69 | .13 |
| pr2392-02 | 2203931.22 | 2230104.44 | 514.53 | .53 | .04 |
| pr2392-07 | 1097300.86 | 1091512.22 | 778.47 | .21 | .96 |
| pr2392-15 | 741147.58 | 73049.11 | 734.49 | .49 | .99 |
| pr2392-30 | 509286.48 | 481040.51 | 807.58 | .94 | .79 |
| fn4461-02 | 1282694.80 | 1282694.48 | 734.76 | .67 | .08 |
| fn4461-03 | 560148.77 | 552314.84 | 515.29 | .40 | .90 |

**TABLE VIII. IMPRECISE SEARCH AND ITERATIVE NEIGHBORHOOD USAGE COMPARISON TEST**
| 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 imprecise search, and C represents not using iterative neighborhood.
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 imprecise local search to avoid the search falling into the local optimum prematurely. When the population evolution is completed, precise calculations are adopted to improve the quality of solutions further. 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 wider range of applications.

REFERENCES
[1] X. Wang and X. Qin, “Asymmetric intimacy and algorithm for detecting communities in bipartite networks”, Phys. A Stat. Mech. its Appl., 462, 569–578, 2016.
[2] Y. Cui and X. Wang, “Detecting one-mode communities in bipartite networks by bipartite clustering triangular”, Phys. A Stat. Mech. its Appl., 457, 307–315, 2016.
[3] S. L. Hakimi, “Optimum Locations of Switching Centers and the Absolute Centers and Medians of a Graph”, Oper. Res., 12(3), 450–459, 1964.
[4] J. M. Mulvey and M. P. Beck, “Solving capacitated clustering problems”, Eur. J. Oper. Res., 18(3), 339–348, 1984.
[5] M. Negreiros and A. Palhano, “The capacitated centred clustering problem”, Comput. Oper. Res., 33(6), 1639–1663, 2006.
[6] F. Boudahri, M. Bennekrouf, F. Belkaid, and Z. Suri, “Application of a Capacitated Centered Clustering Problem for Design of Agri-food Supply Chain Network”, Int. J. Comput. Sci. Issues, 9(4), 300–304, 2012.
[7] A. C. Pillai, J. Chick, L. Johanning, M. Khorasanchi, and V. de Laleu, “Offshore wind farm electrical cable layout optimization”, Eng. Optim., 47(12), 1689–1708, 2015.
[8] C. A. Chou, W. A. Chaolavithwongse, 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., 39(3), 609–619, 2012.
[9] F. Stefanello, O. C. B. de Ara´ujo, and F. M. Muller, “Matheuristics for the capacitated p-median problem”, Int. Trans. Oper. Res., 22(1), 149–167, 2015.
[10] P. Baumann, “A Binary Linear Programming-Based K-Means Approach for the Capacitated Centered Clustering Problem,” 2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), 335-339, 2019.
[11] M. Gn’agi and P. Baumann, “A matheuristic for large-scale capacitated clustering”, Comput. Oper. Res., 132, 2021.
[12] F. Mai, M. J. Fry, and J. W. Ohlmann, “Model-based capacitated clustering with posterior regularization”, Eur. J. Oper. Res., 271(2), 594–605, 2018.
[13] L. J’ano´s’ıkova, M. Herda, and M. Havír, “Hybrid genetic algorithms with selective crossover for the capacitated p-median problem”, Cent. Eur. J. Oper. Res., 25(3), 651–664, 2017.
[14] A. A. Chaves and L. A. N. Lorena, “Clustering search algorithm for the capacitated centered clustering problem”, Comput. Oper. Res., 37(3), 552–558, 2010.
[15] A. C. M. Oliveira and L. A. N. Lorena, “Hybrid evolutionary algorithms and clustering search, In: Abraham A., Grosan C., Ishibuchi H. (eds) Hybrid Evolutionary Algorithms”, Studies in Computational Intelligence, Springer, Berlin, Heidelberg., 75, 77–99, 2007.
[16] A. A. Chaves and L. A. Nogueira Lorena, “Hybrid evolutionary algorithm for the Capacitated Centered Clustering Problem”, Expert Syst. Appl., 38(5), 5013–5018, 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. XLIV Symposium Brazilian Oper. Res. Soc., Rio de Janeiro, Brazil, 2344–2359, 2012.
[18] D. Melo Morales, A. A. Chaves and A. L. Fazenda, “Parallel Clustering Search Applied to Capacitated Centered Clustering Problem,” 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 542-548, 2019.
[19] E. Barojas-payan and D. Sanchez-partida, “Extended GRASP-Capacitated K-Means Clustering Algorithm to Establish Humanitarian Support Centers in Large Regions at Risk in Mexico”, J. Optim., 2018, 1–14, 2018. DOI:10.1155/2018/3605298
[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., 124, 331–346, 2018. DOI: 10.1016/j.cie.2018.07.031
[21] J. F. Gonçalves and M. G. C. Resende, “Biased random-key genetic algorithms for combinatorial optimization”, J. Heuristics, 17(5), 487–525, 2011.
[22] A. Mjirda, R. Todosi´c, S. Hanaﬁpi, P. Hansen, and N. Mladenovi´c, “Sequential variable neighborhood descent variants: an empirical study on the traveling salesman problem”, Int. Trans. Oper. Res., 24(3), 615–633, 2017.
[23] J. Brimberg, N. Mladenovi´c, and D. Uro´sevi´c, “Solving the maximally diverse grouping problem by skewed general variable neighborhood search”, Inf. Sci. (Ny), 295, 650–675, 2015.
[24] Q. Zhou, U. Benlic, Q. Wu, and J. K. Hao, “Heuristic search to the capacitated clustering problem”, Eur. J. Oper. Res., 273(2), 464–487, 2019.
[25] I. Kaabachi, H. Yahyaoui, S. Krichen, and A. Dekdouk, “Measuring and evaluating hybrid metaheuristics for solving the multi-compartment vehicle routing problem”, Meas. J. Int. Meas. Confed., 141, 407–419, 2019.

[26] J. C. Bean, “Genetic Algorithms and Random Keys for Sequencing and Optimization”, ORSA Journal on Computing, 6(2), 154–160, 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., 128, 67–80, 2019.

[28] F. L. Biajoli, A. A. Chaves, L. Antonio, and N. Lorena, “A biased random-key genetic algorithm for the two-stage capacitated facility location problem”, Expert Syst. Appl., 115, 418–426, 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., 11(7), 1371–1384, 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”, AIP Conf. Proc., 1977(1):020052, 1–10, 2018. DOI:10.1063/1.5042908.

[31] W. M. Spears and K. A. De Jong, “On the virtues of parameterized uniform crossover, In: Proceedings of the 4th International Conference on Genetic Algorithms”, San Diego, CA, USA, 230–236, 1991.

[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. Nonlinear, Soft Matter Phys., 76(3), 1–11, 2007.

[33] J. Lee Rodgers and W. Alan Nice Wander, “Thirteen ways to look at the correlation coefficient”, Am. Stat., 42(1), 59–66, 1988.

[34] L. Lorena and E. Senne, “Local Search Heuristics for Capacitated p-Median Problems”, Networks Spat. Econ., 3(4), 407–419, 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., 14(4), 349–364, 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., 257(3), 845–858, 2017.

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

Ping Guo received his Ph.D. degree in Computer Software and Theory in 2004, from Chongqing University, China. Currently, he is a Professor at the Chongqing Key Laboratory of Software Theory and Technology, College of Computer Science of Chongqing University, Chongqing, China. His research interests include different aspects of artificial intelligence and biological computing. He has authored/co-authored more than 180 refereed publications. (Email:guoping@cqu.edu.cn)

YI ZENG was born in Wulanchabu, Inner Mongolia, China, in 1961. He received the bachelor 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 system.