# Neighborhood decomposition-driven variable neighborhood search for capacitated clustering

Xiangjing Lai[a], Jin-Kao Hao[b,*], Zhang-Hua Fu[c,d], Dong Yue[a]

[a]*Institute of Advanced Technology, Nanjing University of Posts and Telecommunications, Nanjing 210023, China*

[b]*LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

[c]*Institute of Robotics and Intelligent Manufacturing, The Chinese University of Hong Kong, Shenzhen 518172, China*

[d]*Shenzhen Institute of Artificial Intelligence and Robotics for Society, Shenzhen 518172, China*

***Computers & Operations Research*, May 2021**

**https://doi.org/10.1016/j.cor.2021.105362**

**Abstract**

The capacitated clustering problem (CCP) is a general model relevant for a variety of important applications in areas such as parallel computing and very large scale integration design. However, the problem is known to be NP-hard, and thus computationally challenging. In this work, we present an original and highly effective variable neighborhood search algorithm for the problem, which is characterized by its neighborhood decomposition technique and a probability-based diversification strategy. The proposed algorithm is assessed via extensive experiments on 110 benchmark instances commonly used in the literature. Computational results show that the algorithm significantly outperforms the existing state-of-the-art algorithms in the literature. This work advances the state-of-the-art of solving the capacitated clustering problem and can be useful for the related practical applications. The key feature of the algorithm, i.e., combining the neighborhood decomposition-driven local search with the perturbation, is of general interest and can help to design effective heuristic algorithms for other important clustering problems.

*Keywords*: Capacitated clustering, graph partitioning, heuristic search, neighborhood decomposition, combinatorial optimization.

* Corresponding author.
*Email addresses:* `laixiangjing@gmail.com` (Xiangjing Lai), `jin-kao.hao@univ-angers.fr` (Jin-Kao Hao), `fuzhanghua@cuhk.edu.cn` (Zhang-Hua Fu), `medongy@vip.163.com` (Dong Yue).

## 1 Introduction

Clustering problems represent a class of relevant models with a variety of practical applications. The goal of a clustering problem is to group a given set of items into a number of fixed or variable $K$ ($K \geq 2$) clusters to optimize an objective function under some possible imperative constraints. Examples of clustering problems include semi-supervised graph clustering [16], constrained graph clustering in biological networks [40], graph partitioning [1,19,26,43], and various $p$-center and $p$-median problems [5,7,8,15,27,36]. In general, clustering problems are NP-hard and thus computationally challenging.

The capacitated clustering problem (CCP) studied in this work is a typical clustering problem with a number of applications. Representative applications that can be conveniently formulated by CCP concern facility locations [9], parallel computing [19], very large scale integration design [41], and creation of peer review groups [6].

CCP generalizes three NP-hard problems: the graph partitioning problem (CPP) [1,11,13,30], the handover minimization problem [28,32], and the maximally diverse grouping problem (MDGP) [3,12,23,33]. Consequently, solving CCP is a computationally difficult task and represents a formidable challenge from the perspective of designing effective search algorithms.

The CCP problem can be described as follows. Given a weighted complete graph $G = (V, E, c, w)$ and a positive integer $K$, where $V = \{v_1, v_2, \ldots, v_N\}$ is the set of $N$ vertices, $E$ represents the set of $N(N-1)/2$ edges, $c = \{c_{ij} \geq 0 : \{v_i, v_j\} \in E\}$ is the set of edge weights, and $w = \{w_i \geq 0 : v_i \in V\}$ is the set of vertex weights, the capacitated clustering problem (CCP)[9] involves partitioning the vertex set $V$ into $K$ disjoint clusters $C_1, C_2, \ldots, C_K$ such that the sum of vertex weights (i.e., $\sum_{v \in C_g} w(v)$) of each cluster $C_g$ ($g = 1, 2, \ldots, K$) lies in a given interval $[L, U]$, while maximizing the sum of the edge weights in the same clusters, where $L$ and $U$ are called the lower and upper bounds of the capacity of each cluster, respectively. An illustrative example for CCP is given in Fig. 1.

Formally, CCP can be stated as follows [9,28]:

$$(CCP) \quad \text{Maximize} \quad f = \sum_{g=1}^{K} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} c_{ij} X_{ig} X_{jg} \tag{1}$$

$$\text{Subject to} \quad \sum_{g=1}^{K} X_{ig} = 1, i = 1, 2, \ldots, N \tag{2}$$

$$L \leq \sum_{i=1}^{N} w_i X_{ig} \leq U, g = 1, 2, \ldots, K \tag{3}$$

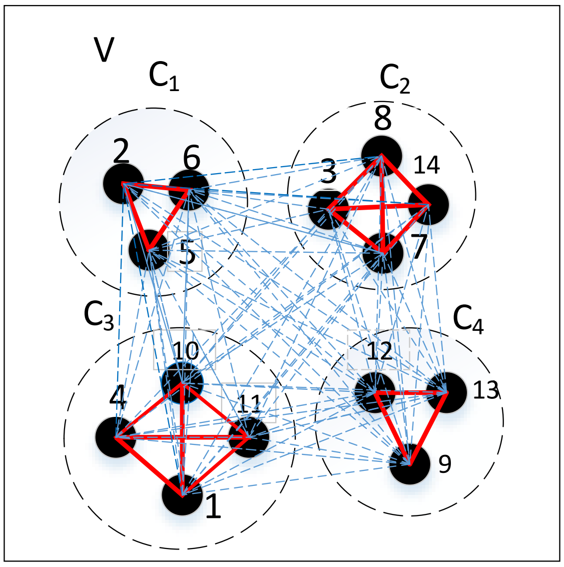


Fig. 1. An illustrative example for CCP. Given a weighted complete graph $G = (V, E, c, w)$ with $|V| = 14$, $w_i = 1.0$ for all vertices $v_i$ in $V$, the number of clusters $K = 4$, and lower and upper bounds $[L, U] = [3, 4]$, the corresponding CCP consists of partitioning the set $V$ of vertices into 4 clusters $C_1$, $C_2$, $C_3$ and $C_4$, such that $L \le \sum_{v \in C_g} w(v) \le U$ for $\forall g \in \{1, 2, 3, 4\}$, while the sum of the edge (indicated in red) weights in the same clusters is maximized.

$$X_{ig} \in \{0, 1\}, i = 1, 2, \ldots, N; g = 1, 2, \ldots, K \tag{4}$$

where $X_{ig}$ is a binary variable that takes the value of 1 if the vertex $v_i$ is located in cluster $C_g$ and 0 otherwise. Thus, the objective function $f$, which is to be maximized, adds up the edge weights $c_{ij}$ for edges whose endpoints $i$ and $j$ belong to the same cluster $C_g$ ($X_{ig} = X_{jg} = 1$). The set of constraints (2) guarantees that each vertex belongs to exactly one cluster, and the set of constraints (3) forces the sum of vertex weights of each cluster lies in $[L, U]$.

Due to the importance of CCP, various search methods have been proposed in the literature. As the review in Section 2 shows, most existing studies focus on heuristic algorithms which aim to find satisfactory solutions as fast as possible, without optimality guarantee of the attained solutions. In particular, the most effective algorithms are based on neighborhood search (also called stochastic local search [20]) whose performance critically depends on the adopted neighborhoods as well as the way the neighborhoods are examined. Indeed, given that these algorithms need to evaluate a set of candidate solutions at each iteration, the search becomes very time-consuming for solving large problems. Thus, research on the design of new algorithms as well as efficient neighborhood examination methods becomes highly relevant.

In this work, we aim to advance the state-of-the-art of CCP in terms of practical solving of large problem instances. Inspired by a related work on the maximally diverse grouping problem [24], some early studies about the neighborhood decomposition strategies [10,21,38] and the technique of *don't look bits* [2,39], we propose a new heuristic algorithm called the neighborhood decomposition-driven variable neighborhood search algorithm for CCP. Extensive experimental results show that the proposed algorithm outperforms significantly the state-of-the-art CCP algorithms on the 110 benchmark instances widely used in the literature.

The remaining parts of paper are organized as follows. In Section 2, we review representative recent studies on CCP. In Section 3, we describe the proposed algorithm. In Section 4, experimental results and comparisons are reported to assess the algorithm. Section 5 shows an analysis of key algorithmic components. Last section summarizes the main findings of this work and provides research perspectives.

## 2 Literature review

Since the introduction of CCP, a large number of studies have been devoted to the problem. Useful information about the studies on CCP prior to 2011 can be found, for instance, in [9,28]. In this section, we focus on the most recent developments on solving methods for CCP.

Among the existing algorithms, only one provides exact solutions [25], based on linear and quadratic models solved by commercial optimizers (CPLEX and Gurobi). However, the test instances studied are quite small ($N \leq 50$), compared to the instances tested in this study ($240 \leq N \leq 2000$). To handle large instances, heuristic algorithms are typically used, which can be roughly divided into four categories.

The first category is based on the greedy randomized adaptive search procedure (GRASP) metaheuristic [37], which iterates a stochastic greedy construction procedure and a subsequent neighborhood search procedure. In [9], Deng and Bard proposed the first reactive GRASP procedure for CCP. In [28,29], Martínez-Gavara et al. introduced a simplified version of Deng and Bard's GRASP method and several variants, where a special restricted candidate list strategy was used by the greedy construction procedure and different neighborhoods (e.g., the insertion neighborhood, the swap neighborhood, or a new 2-1 exchange neighborhood) were employed in the neighborhood search procedure.

The second category is based on the tabu search (TS) metaheuristic [14]. In [28], Martínez-Gavara et al. proposed a TS algorithm based on the 2-1 exchange neighborhood, and a hybrid local search algorithm integrating a simplified GRASP procedure and the TS algorithm (GRASP+TS). They also presented an adapted version of the TS algorithm with strategic oscillation initially designed for the related MDGP, where the insertion and swap neighborhoods are adopted as the basic neighborhood structures. In [42], Zhou et al. introduced a penalty-based TS algorithm (FITS) that explores both feasible and infeasible regions.

The third category relies on the variable neighborhood search (VNS) metaheuristic [18,31]. In [22], based on the insertion neighborhood, the swap neighborhood, and the 2-1 exchange neighborhood, Lai and Hao introduced an iterated variable neighborhood search (IVNS) algorithm by integrating organically an extended variable neighborhood descent method and a randomized shake procedure. In [4], Brimberg et al. proposed a general variable neighborhood search (GVNS) algorithm and a skewed general variable neighborhood search (SGVNS) algorithm based on the same three neighborhoods.

The fourth category relies on the hybrid population-based evolutionary framework which combines a local search procedure and a crossover operator. In [9], Deng and Bard mixed a GRASP procedure, a path relinking procedure and a variable neighborhood descent method. In [42], in addition to the FITS algorithm, Zhou et al. proposed a memetic algorithm by combining the FITS algorithm serving as a local search procedure and a clustering-based crossover operator.

According to the computational results reported in the above studies, we identify five state-of-the-art algorithms: GRASP+TS [28], FITS [42], IVNS [22], GVNS [4], and SGVNS [4]. These algorithms will be used as our reference algorithms for the computational studies of Section 4.

Our literature review indicates that the best performing CCP algorithms are all neighborhood search algorithms which explore iteratively one or more neighborhoods (e.g., insertion neighborhood, swap neighborhood, and 2-1 exchange neighborhood). Specifically, for a given neighborhood, such an algorithm needs, at each iteration, to examine all or some neighbor solutions to identify the solution of interest (e.g., the best solution among all neighbor solutions or an improving solution better than the current solution). The search becomes very time-consuming when the neighborhood contains many neighbor solutions (this is typically the case of the swap neighborhood and the 2-1 exchange neighborhood). Thus, the issue of a fast examination of the considered neighborhoods becomes critical and directly impacts the performance of the search algorithm.

In this work, to speed up neighborhood examination, we design a neighborhood decomposition strategy for the CCP. This strategy divides a given neighborhood into a number of disjoint subsets (called neighborhood blocks) of neighbor solutions and identifies each promising neighborhood block with a 0-1 state variable. This decomposition accelerates neighborhood examination by checking only the promising neighborhood blocks. This is in sharp contrast to existing algorithms in the literature that do not make a distinction between promising neighbor solutions and non-promising neighbor solutions and thus waste computation time by repetitively re-examining non-promising neighbor solutions.

## 3 Neighborhood decomposition-driven variable neighborhood search

The proposed neighborhood decomposition-driven variable neighborhood search (NDVNS) algorithm is based on the general variable neighborhood search metaheuristic [18,31]. The primary innovative ingredients of the algorithm include its neighborhood decomposition strategy designed for CCP to accelerate the search process and a probabilistic perturbation strategy to control the tradeoff between search intensification and diversification.

The current neighborhood decomposition strategy dynamically partitions a neighborhood into a number of disjoint neighborhood blocks, and enables the search algorithm to only examine the promising neighborhood blocks which are identified by a 0-1 state value. By ignoring the other blocks, the algorithm significantly increases its computational efficiency. The current neighborhood decomposition strategy is related to early candidate list based neighborhood decomposition strategies and the *don't look bits* technique. The candidate list approach was used to decompose a given neighborhood into coordinated subsets so that the search algorithm only focuses on some subsets with desirable features [10,21,38]. The *don't look bits* technique was initially developed to speed up local search procedures for the traveling salesman problem [2] and subsequently adapted to the quadratic assignment problem (QAP) [39]. In particular, taking QAP as an example, to avoid scanning a full neighborhood, the *don't look bits* technique employs a dynamically updated 0-1 vector to distinguish the promising items from the unpromising items. Then the search examines only the promising items, which significantly speeds up the algorithm. On the other hand, unlike these early approaches, the current decomposition strategy does not employ any candidate list and uses a simple 0-1 state matrix to perfectly identify the visited subsets that do not contain an improving solution.

Basically, for a given problem instance, i.e., a double-weighted complete graph $G = (V, E, c, w)$, a positive integer $K$, and the lower bound $L$ and upper bound $U$ of clusters on the capacity, the proposed algorithm explores the search space $\Omega$ composed of all feasible $K$-partitions of the vertex set $V$ satisfying the capacity constraints of clusters, i.e., $\Omega = \{\{C_1, C_2, \ldots, C_K\} : V = \cup_{i=1}^{i=K} C_i, C_i \cap C_j = \emptyset \; \forall i \neq j, L \leq |C_g| \leq U, \forall g\}$, where $|C_g| = \sum_{v \in C_g} w(v)$.

The main framework and components of the NDVNS algorithm are described in the following subsections.

### *3.1 Main framework of the NDVNS Algorithm*

**Algorithm 1:** Neighborhood decomposition-driven variable neighborhood search (NDVNS) for capacitated clustering

```
Input: A double-weighted complete graph $G = (V, E, c, w)$, an integer
       $K$, time limit $t_{max}$, and parameters $\alpha$, $Q$, $k_{min}$, $k_{max}$
Output: The best feasible $K$-partition of $G$ found ($s^*$)
1  $s \leftarrow InitialSolution(G, K)$
2  $s \leftarrow$ NDVND$_3(s)$                    /* Local search, Algorithm 2 */
3  $s^* \leftarrow s$
4  $k \leftarrow k_{min}$
5  while $time() < t_{max}$ do
6      $s' \leftarrow Shake(s, k)$    /* Perturb the solution $s$, Section 3.3 */
7      $s'' \leftarrow$ NDVND$_2(s')$                /* Local search, Algorithm 2 */
8      if $(\frac{f(s'')}{f(s)} + \alpha \cdot d(s'', s) > 1) \wedge (\frac{f(s'')}{f(s^*)} + \alpha \cdot d(s'', s^*) > 1)$ then
9          $s \leftarrow s''$
10         if $f(s'') > f(s^*)$ then
11             $s^* \leftarrow s''$
12             $k \leftarrow k_{min}$
13         else
14             $r \leftarrow rand(0, 1)$
15             if $r < Q$ then
16                 /* $Q$ is a parameter                         */
17                 $k \leftarrow k_{min}$
18             else
19                 $k \leftarrow k + 1$
20             end
21         end
22     else
23         $k \leftarrow k + 1$
24     end
25     if $k \geq k_{max}$ then
26         $k \leftarrow k_{min}$
27     end
28 end
29 return $s^*$
```

The NDVNS algorithm (see the flowchart in Fig. 2) combines an initialization procedure aiming at generating a feasible solution, two local search procedures (i.e., NDVND$_2$ and NDVND$_3$) and a shake procedure aiming at diversifying the search process.

Algorithm 1 shows the main framework of the NDVNS algorithm, where $s$ and

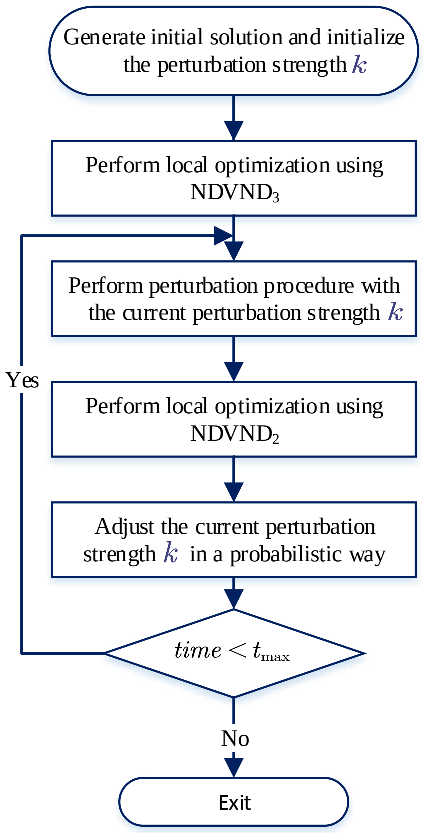

Generate initial solution and initialize the perturbation strength $k$
Perform local optimization using NDVND$_3$
Perform perturbation procedure with the current perturbation strength $k$
Perform local optimization using NDVND$_2$
Adjust the current perturbation strength $k$ in a probabilistic way
$time < t_{\max}$
Yes
No
Exit

Fig. 2. The main flowchart of the proposed NDVNS algorithm.

$s^*$ denote respectively the current solution and the best solution found so far, $d(\cdot,\cdot)$ is a distance function defined by the partition distance metric [3,17,35], and $k$ is the current perturbation strength of the perturbation strategy (see Section 3.3).

The algorithm starts from an initial solution generated by the two-stage initialization procedure of [22] (line 1 of Algorithm 1), and then performs a local search procedure (i.e., NDVND$_3$) to locally improve the initial solution (line 2). After that, the perturbation strength $k$ of the shake procedure $Shake(\cdot,\cdot)$ is initially set to the minimum value $k_{min}$ and the search process enters a "while" loop in which several operations are iteratively performed to improve the current solution until the time limit ($t_{max}$) is reached (lines 5-28).

At each "while" loop, the current solution $s$ is first perturbed by the $Shake(\cdot,\cdot)$ procedure and is then improved by a fast local search procedure (i.e., NDVND$_2$) (lines 6 and 7). Then, the resulting solution $s''$ is conditionally accepted according to its quality and distances to $s$ and $s^*$ (lines 8–9), similar to the SGVNS algorithm of [4]. Moreover, the value of $k$ is set to $k_{min}$ and the recorded best solution is updated if an improved solution is found, i.e., $s^* \leftarrow s$ and $k \leftarrow k_{min}$, and the value of $k$ is increased as $k \leftarrow k+1$, otherwise. In addition, the algorithm employs a probability $Q$ (a parameter) to control the perturbation strength $k$ to maintain a suitable tradeoff of search intensification and diversification (lines 15-20). That is, the value of $k$ switches to $k_{min}$ with probability $Q$ if the recorded best solution $s^*$ has not been improved during the current iteration. Finally, the value of $k$ is reset to $k_{min}$ as long as $k$ reaches the allowed maximum value $k_{max}$ (lines 25–27). We describe below the components of the algorithm.

## *3.2 Local Optimization Methods of the NDVNS Algorithm*

This subsection presents the neighborhood decomposition-driven variable neighborhood descent (NDVND) methods and the local optimization procedure.

### *3.2.1 General procedure of variable neighborhood descent method*

Variable neighborhood descent (VND) is a local search approach that explores local optimal solutions with several ordered neighborhoods $N_\theta$ ($\theta = 1, 2, \ldots, \theta_{max}$). Specifically, the VND method starts with the first neighborhood $N_\theta$ ($\theta = 1$), and then switches to the next neighborhood $N_{\theta+1}$ when a local optimum with respect to the current neighborhood $N_\theta$ is attained. Moreover, VND switches immediately to the first neighborhood $N_1$ from the current neighborhood $N_\theta$ ($\theta = 2, 3, \ldots, \theta_{max}$) as soon as an improving solution is found. Finally, VND stops when the search process reaches the last neighborhood $N_{\theta_{max}}$ and no improving solution can be found in $N_{\theta_{max}}$.

### *3.2.2 Neighborhoods and neighborhood decomposition*

Like previous studies [4,22,28], our NDVNS algorithm employs three complementary neighborhoods, i.e., the insertion neighborhood $N_1$, the swap neighborhood $N_2$, and the 2-1 exchange neighborhood $N_3$.

The insertion neighborhood $N_1$ is generated by the $OneMove$ operator. Given a solution $s = \{C_1, C_2, \ldots, C_K\}$, the $OneMove$ operator (denoted by $< v, C_i, C_j >$) transfers a node $v$ from its current cluster $C_i$ to another cluster $C_j$ ($1 \leq j \neq i \leq K$), such that the resulting solution denoted by $s \oplus < v, C_i, C_j >$ is still feasible. As such, the neighborhood $N_1(s)$ is composed of all possible feasible solutions which can be obtained by applying the $OneMove$ operator to $s$, i.e.,

$$\begin{aligned} N_1(s) = \{s \oplus < v, C_i, C_j > : v \in C_i, |C_i| - w(v) \geq L, \\ |C_j| + w(v) \leq U, i \neq j\} \end{aligned} \tag{5}$$

Clearly, the size of $N_1(s)$ is bounded by $O(N \times K)$.

Meanwhile, according to the formulation of CCP (see the illustrative example in Fig. 1) and the definition of $N_1(s)$, it is easy to observe that the neighborhood $N_1(s)$ can be partitioned into $K \times (K-1)$ disjoint neighborhood blocks $B_1[i][j](s)$ $(1 \leq i, j \leq K,\ i \neq j)$, i.e., $N_1(s) = \cup_{1 \leq i \neq j \leq K} B_1[i][j](s)$, where the neighborhood block $B_1[i][j](s)$ is defined as:

$$B_1[i][j](s) = \{s \oplus < v, C_i, C_j > : v \in C_i, |C_i| - w(v) \geq L, \\ |C_j| + w(v) \leq U\} \tag{6}$$

Moreover, these neighborhood blocks can be characterized by a $K \times K$ binary asymmetric state matrix $M_1$ (see example in Fig. 3 (a)), where entry $M_1[i][j]$ $(i \neq j)$ takes 0 if the corresponding block $B_1[i][j](s)$ has been previously checked by the algorithm and does not contain any improving solution, and takes 1 otherwise. The diagonal entries of $M_1$ always take 0. Thus, the state matrix distinguishes the promising neighborhood blocks (marked with $M_1[i][j] = 1$) from non-promising neighborhood blocks (marked with $M_1[i][j] = 0$). With an appropriate update of the state matrix $M_1$ as we detail in Section 3.2.4, we can focus on the blocks $B_1[i][j](s)$ $(i \neq j)$ with $M_1[i][j] = 1$ and speed up the search without missing improving solutions (i.e., guaranteeing the correctness of the neighborhood search process).

Since it is not necessary to examine the neighborhood blocks $B_1[i][j](s)$ with $M_1[i][j] = 0$ $(i \neq j)$, the complexity of examining neighborhood $N_1(s)$ can be reduced from $O(K \times (K-1) \times P)$ to $O(m \times P)$, where $m$ is the number of blocks $B_1[i][j](s)$ with $M_1[i][j] = 1$ $(i \neq j)$ and $P = Max_{M_1[i][j]=1, i \neq j}\{|B_1[i][j](s)|\}$. As such, the complexity reduction (i.e., search speed-up) becomes significant when $m$ becomes much less than the total number of neighborhood blocks (i.e., $m << K \times (K-1)$). We observe that this remains true especially when the number $K$ of clusters is large. The same justification holds for the two other neighborhoods $N_2$ and $N_3$.

The swap neighborhood $N_2$ is induced by the $Swap(\cdot, \cdot)$ operator. Given two vertices $v$ and $u$ located in different clusters of the current solution $s = \{C_1, C_2, \ldots, C_K\}$, $Swap(v, u)$ generates a neighbor solution of $s$ by swapping the clusters of $v$ and $u$ if the resulting solution is feasible. Thus, the swap neighborhood $N_2(s)$ is given by:

$$N_2(s) = \{s \oplus Swap(v, u) : v \in C_i, u \in C_j, L \leq |C_i| + w(u) - w(v), \\ |C_j| + w(v) - w(u) \leq U, i \neq j\} \tag{7}$$

whose size is bounded by $O(N^2)$.

The neighborhood $N_2(s)$ can be partitioned into $K \times (K-1)/2$ disjoint blocks $B_2[i][j](s)$ $(1 \leq i < j \leq K)$ since $B_2[i][j](s)$ $(i \neq j)$ is the same as $B_2[j][i](s)$, i.e., $N_2(s) = \cup_{1\le i<j\le K} B_2[i][j](s)$, where each $B_2[i][j](s)$ is defined as:

$$B_2[i][j](s) = \{s \oplus Swap(v,u) : v \in C_i, u \in C_j, L \le |C_i| + w(u) - w(v), \\ |C_j| + w(v) - w(u) \le U\} \tag{8}$$

Thus, we can characterize the neighborhood $N_2(s)$ with a $K \times K$ binary symmetric state matrix $M_2$ (see example in Fig.3 (b)), where the entry $M_2[i][j]$ ($i < j$) corresponds to the neighborhood block $B_2[i][j](s)$ and takes 0 if $B_2[i][j](s)$ has been previously examined and does not contain any improving solution, and takes 1 otherwise.

The neighborhood $N_3$ is induced by the 2-1 exchange operator $Exchange(\cdot,\cdot,\cdot)$. Given three vertices $v$, $u$, and $z$ in the current solution $s = \{C_1, C_2, \ldots, C_K\}$, where $v$ and $u$ are located in the same cluster $C_i$ and $z$ is located in another cluster $C_j$, the $Exchange(\cdot,\cdot,\cdot)$ operator transfers the vertices $v$ and $u$ from $C_i$ to $C_j$ and simultaneously transfers vertex $z$ from $C_j$ to $C_i$, such that the resulting solution is still feasible. As such, the neighborhood $N_3(s)$ can be written as:

$$N_3(s) = \{s \oplus Exchange(v,u,z) : v,u \in C_i, z \in C_j, L \le |C_i| - w(u) - \\ w(v) + w(z), |C_j| + w(v) + w(u) - w(z) \le U, i \ne j\} \tag{9}$$

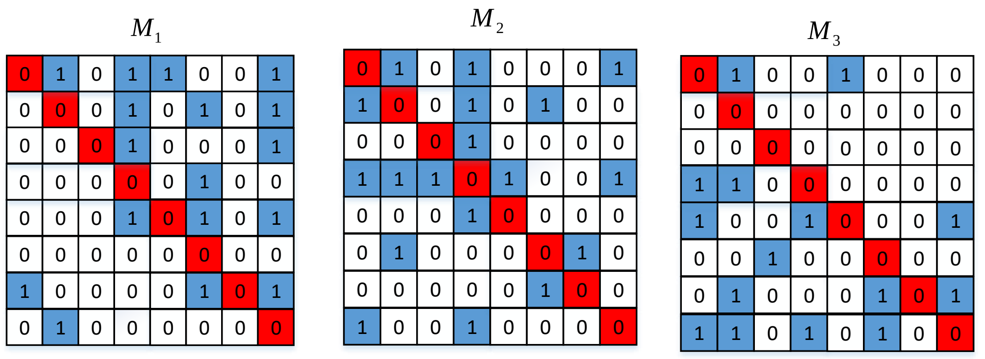
The size of $N_3(s)$ is bounded by $O(N^3)$.

Similar to the neighborhood $N_1(s)$, the neighborhood $N_3(s)$ can be partitioned into $K \times (K-1)$ disjoint blocks $B_3[i][j](s)$ ($i \ne j$), i.e., $N_3(s) = \cup_{1\le i\ne j\le K} B_3[i][j](s)$, where each $B_3[i][j](s)$ is defined as:

$$B_3[i][j](s) = \{s \oplus Exchange(v,u,z) : v,u \in C_i, z \in C_j, L \le |C_i| \\ -w(u) - w(v) + w(z), |C_j| + w(v) + w(u) - w(z) \le U\} \tag{10}$$

The neighborhood $N_3$ is also associated with a $K \times K$ binary asymmetric state matrix $M_3$ (see Fig.3 (c)), the entry $M_3[i][j]$ corresponds to the block $B_3[i][j](s)$ and its value has the same meaning as in $M_1[i][j]$ and $M_2[i][j]$.

The above neighborhood decomposition technique is based on the following key observation. For many clustering or grouping problems including CCP, the objective function is given by the sum of subunit objectives defined on $K$ individual clusters. This particular feature makes most neighborhood blocks mutually independent in terms of the move value $\Delta_f(s) = f(s \oplus Move) - f(s)$ (i.e., the change of objective function value between the current solution $s$ and a neighbor solution $s \oplus Move$ generated by transforming $s$ with the $Move$ operator). As a result, when a given neighborhood block is exploited, the move



$M_1$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

$M_2$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

$M_3$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

(a) A state matrix of $N_1$ (b) A state matrix of $N_2$ (c) A state matrix of $N_3$

Fig. 3. Three illustrative examples for the state matrices of the neighborhoods $N_1$, $N_2$ and $N_3$, where $M_1$ and $M_3$ are asymmetric, $M_2$ are symmetric, and the value of $K$ is 8. The diagonal elements of matrices are indicated in red, and the elements taking the value of 1 are indicated in blue. $M_\theta[i][j]$ $(\theta = 1, 2, 3)$ corresponds to the neighborhood block $B_\theta[i][j]$ and $M_\theta[i][j] = 0$ if $B_\theta[i][j]$ has been previously checked by the algorithm and does not contain any improving solution, and $M_\theta[i][j] = 1$ otherwise.

values of neighborhood moves will not be affected for most other neighborhood blocks. Our neighborhood decomposition technique enables the search algorithm to explicitly take advantage of this feature. By concentrating on the promising blocks $B_\theta[i][j]$ $(i \neq j, \theta = 1, 2, 3)$ which are identified by $M_\theta[i][j] = 1$ (i.e., the unexamined blocks or the blocks affected in the previous iterations), the algorithm will increase considerably its computational efficiency and search effectiveness, as confirmed by the computational results reported in Section 4. Neighborhood decomposition has recently contributed to effectively solve the maximally diverse grouping problem [24].

### *3.2.3 Neighborhood decomposition-driven VND*

Based on the above standard VND framework and the neighborhoods $N_1$, $N_2$, and $N_3$ as well as their decompositions presented in Section 3.2.2, we introduce the neighborhood decomposition-driven VND algorithm (i.e., NDVND$_{\theta_{max}}$ where $\theta_{max}$ represents the number of neighborhoods used) as follows. First, NDVND$_{\theta_{max}}$ (see Algorithm 2) initializes all state matrices $M_\theta$ $(\theta = 1, 2, \ldots, \theta_{max})$ (lines 3 and 4), and then explores dynamically the given neighborhoods (lines 5–14). For each neighborhood $N_\theta$ $(\theta = 1, 2, \ldots, \theta_{max})$, the search is performed with the $LNS_\theta$ procedure described in Algorithms 3 and 4, where the neighborhood blocks $B_\theta[i][j](s)$ with $M_\theta[i][j] = 1$ are orderly examined and the state matrices $M_\theta$ $(\theta = 1, 2, \ldots, \theta_{max})$ are accordingly updated.

The proposed NDVNS algorithm employs two neighborhood decomposition-driven VND procedures for local optimization. The first one (NDVND$_2$ with $\theta_{max} = 2$) uses neighborhoods $N_1$ and $N_2$, while the second one (NDVND$_3$ with

$\theta_{max} = 3$) explores neighborhoods $N_1$, $N_2$, $N_3$. Since NDVND$_3$ explores one more neighborhood, it is much more time-consuming than NDVND$_2$. Thus, NDVND$_3$ is performed only once at the the beginning of the present NDVNS algorithm, while NDVND$_2$ is used as the main search procedure.

In addition, NDVND$_2$ and NDVND$_3$ uses the incremental technique of [22] to evaluate efficiently the quality of a neighbor solution in $N_\theta(s)$ ($\theta \in \{1, 2, 3\}$). For this, a $N \times K$ matrix $\gamma$ is maintained during the search process, where $\gamma[i][g] = \Sigma_{u \in C_g} c_{iu}$ ($1 \leq i \leq N$,$1 \leq g \leq K$). With the help of this matrix, the quality of a neighbor solution can be rapidly assessed in $O(1)$, and the matrix $\gamma$ can be updated in $O(N)$ after each solution transition.

**Algorithm 2:** Neighborhood Decomposition-driven Variable Neighborhood Descent Method (NDVND$_{\theta_{max}}$) with the $\theta_{max}$ neighborhoods

```
1  Function NDVND_{θmax}(s_0)
   Input: Solution $s_0$
   Output: The local optimum solution $s$
2  $s \leftarrow s_0$
3  Initialize the state matrices $M_1, \ldots, M_{\theta_{max}}$
4   /* $M_\theta[i][j] \leftarrow 1, M_\theta[i][i] \leftarrow 0,\ 1 \leq \theta \leq \theta_{max}$        */
5  $\theta \leftarrow 1$ /* $\theta$ denotes the index of current neighborhood       */
6  while $\theta \leq \theta_{max}$ do
7      $(Improve, s, M_1, \ldots, M_{\theta_{max}}) \leftarrow LSN_\theta(s, M_1, \ldots, M_{\theta_{max}})$
8      /* Algorithms 3 or 4                                                        */
9      if $(\theta > 1) \wedge (Improve = \textbf{true})$ then
10         $\theta \leftarrow 1$
11     else
12         $\theta \leftarrow \theta + 1$
13     end
14 end
15 return $s$
```

### 3.2.4 *Update of state matrices and the related principle*

In the NDVNS algorithm, all three neighborhoods $N_1$, $N_2$ and $N_3$ are examined block by block, as shown in Algorithms 3 and 4, and the associated state matrices $M_1$, $M_2$ and $M_3$ are accordingly updated as the search progresses.

The updating rule of these state matrices can be described as follows. For a neighborhood $N_\theta(s)$ ($\theta = 1, 2, 3$), the entry $M_\theta[i][j]$ ($i \neq j$) is first set to 0 once the neighborhood block $B_\theta[i][j](s)$ has been checked. Then, the entries $M_\theta[i][t]$, $M_\theta[t][i]$, $M_\theta[j][t]$, and $M_\theta[t][j]$ ($1 \leq t \leq K$, $t \neq i, j$) all are set to 1 if an improving solution is found in the block $B_\theta[i][j](s)$, and keep unchanged otherwise, as illustrated in Algorithms 3 and 4. Clearly, the time complexity

**Algorithm 3:** Local optimization with the neighborhood $N_1$

```
Function LSN_1($< s, M_1, \ldots, M_{\theta_{max}} >$)
Input: $s$, $M_1$, ..., $M_{\theta_{max}}$
Output: $< Improve, s, M_1, \ldots, M_{\theta_{max}} >$
$Improve \leftarrow$ true
while $Improve$ = true do
    $Improve \leftarrow$ false
    for $i \leftarrow 1$ to $K$ do
        for $j \leftarrow 1$ to $K$ do
            if $M_1[i][j] = 1$ then
                $M_1[i][j] \leftarrow 0$
                for each $s^{'} \in B_1[i][j](s)$ do
                    if $f(s^{'}) > f(s)$ then
                        $s \leftarrow s^{'}$
                        $Improve \leftarrow$ true
                    end
                end
            end
            if $Improve$ = true then
                Update $M_1$, ..., $M_{\theta_{max}}$          /* Section 3.2.4 */
            end
        end
    end
end
return $< Improve, s, M_1, \ldots, M_{\theta_{max}} >$
```

of updating the state matrices $M_1$, $M_2$ and $M_3$ is bounded by $O(K)$.

According to the objective of CCP, the move value $\Delta_f$ changes only for the moves of a few of neighborhood blocks. Furthermore, given that the blocks $B_\theta[i][j](s)$ with $M_\theta[i][j] = 0$ ($\theta = 1, 2, 3,\ i \neq j$) have been checked previously without finding any improving solution, ignoring the candidate solutions included in these neighborhood blocks will save a significant amount of computational effort. Thus, the above updating rule speeds up the search process without compromising solution quality. Moreover, it is worth noting that according to this updating rule, a larger number ($K$) of clusters means usually a smaller proportion of blocks whose states need to be updated, implying a more significant search speedup in this case. In other words, the advantage of the neighborhood decomposition technique becomes even more evident when the number $K$ of clusters is large. This is indeed confirmed by our computational results of Section 4.

**Algorithm 4:** Local optimization with the neighborhood $N_\theta$ ($\theta = 2$ or 3)

```
1  Function $LSN_\theta(< s, M_1, \ldots, M_{\theta_{max}} >)$
   Input: $s$, $M_1$, ..., $M_{\theta_{max}}$
   Output: $< Improve, s, M_1, \ldots, M_{\theta_{max}} >$
2  $Improve \leftarrow$ false
3  for $i \leftarrow 1$ to $K$ do
4      for $j \leftarrow 0$ to $K$ do
5          if $M_\theta[i][j] = 1$ then
6              $M_\theta[i][j] \leftarrow 0$
7              /* Also, $M_\theta[j][i] \leftarrow 0$ if $\theta = 2$ */
8              for each $s' \in B_\theta[i][j](s)$ do
9                  if $f(s') > f(s)$ then
10                     $s \leftarrow s'$
11                     $Improve \leftarrow$ true
12                 end
13             end
14             if $Improve$ = true then
15                 Update $M_1$, ..., $M_{\theta_{max}}$        /* Section 3.2.4 */
16                 return $< Improve, s, M_1, \ldots, M_{\theta_{max}} >$
17             end
18         end
19     end
20 end
21 return $< Improve, s, M_1, \ldots, M_{\theta_{max}} >$
```

## *3.3 Shake Procedure*

To enhance its diversification ability, our NDVNS algorithm employs a *Shake* procedure to perturb the solutions returned by the local search procedure NDVND$_2$. The *Shake* procedure is composed of $k$ consecutive swap operations, where $k$ is the perturbation strength, which is probabilistically adjusted during the search process as described in Algorithm 1. For each swap operation, two vertices $v$ and $u$ located in different clusters are first selected randomly, and then their positions are swapped to generate a feasible solution.

## *3.4 Discussions*

Compared to the existing CCP algorithms in the literature, the primary innovation of our approach is the neighborhood decomposition-driven VND methods and the probabilistic strategy to determine the perturbation strength.

First, NDVNS is the first heuristic algorithm that applies the neighborhood decomposition technique to solve CCP, which proves to be quite successful.

Among the three adopted neighborhoods, this study is the first to introduce an effective decomposition of the 2-1 exchange neighborhood, which is generalizable to local search algorithms for other problems.

Second, NDVNS reinforces the VNS method by employing a probabilistic strategy to tune the perturbation strength. This strategy is of general nature and can be usefully combined with other VNS procedures to control search intensification and diversification.

Finally, compared to the work on MDGP in [24], the current work shares the idea of neighborhood decomposition as well as the general variable neighborhood search framework. Meanwhile, given that CCP and MDGP are two different problems, NDVNS is different from the algorithm presented in [24]. In particular, NDVNS features the new 2-1 exchange neighborhood (with its decomposition) and the probabilistic perturbation strategy. As it is shown in Section 4, the NDVNS algorithm integrating these features, along with the problem-specific design of other search components, generally outperforms the state-of-the-art CCP algorithms available in the literature.

## 4 Experimental Results and Comparisons

This section is dedicated to an performance assessment of our NDVNS algorithm, based on computational experiments on benchmark instances commonly used in the literature.

### *4.1 Benchmark Instances*

The test suite is composed of 110 commonly-used benchmark instances (available at `http:www.mi.sanu.ac.rs/~nenad/ccp` or from [28]), belonging to five groups:

- RanReal240: This set contains 20 small instances with $N = 240$, $K = 12$, $L = 75$, and $U = 125$, where the vertex weights are an integer randomly generated in the intervals $[1, 10]$ and the edge weights are a real number randomly generated in $[0, 100]$.
- RanReal480: This set contains 20 medium-sized instances with $N = 480$, $K = 20$, $L = 100$, and $U = 150$, where the vertex weights and the edge weights are generated as in the set RanReal240.
- RanReal960: This set contains 30 large instances with $N = 960$, including 10 instances with $K = 30$, $L = 120$ and $U = 180$, 10 instances with $K = 40$, $L = 90$ and $U = 135$, and 10 instances with $K = 60$, $L = 60$ and $U = 90$,

where the vertex weights are an integer randomly generated in $[1, 10]$ and the edge weights are a real number randomly generated in $[0, 100]$.

- MDG-a: This set contains 20 large instances with $N = 2000$, $K = 50$, $L = 150$ and $U = 250$, the vertex weights are an integer generated randomly in $[1, 10]$, and the edge weights are an integer generated randomly in $[0, 10]$.
- MDG-a-40: This set contains 20 large instances with $N = 2000$, $K = 40$, $L = 200$ and $U = 300$, with the same vertex weights and edge weights as the MDG-a instances.

## *4.2 Experimental Protocol*

Table 1
Settings of parameters

| Parameters | Section | Description | Values |
|---|---|---|---|
| $k_{min}$ | 3.1 | minimum strength of the shake procedure | 1 |
| $k_{max}$ | 3.1 | maximum strength of the shake procedure | $N/K$ |
| $Q$ | 3.1 | a parameter used in the diversification mechanism | 0.2 |
| $\alpha$ | 3.1 | a parameter used in the acceptance criterion | 0.01 |

Table 1 indicates the parameter setting, which was obtained empirically. Our experiment shows that among these parameters, $Q$ is the most critical. We present a detailed analysis of this parameter in Section 5. It worth mentioning that this parameter setting (default setting) was used consistently in our experiments to solve all 110 instances, though fine-tuning some parameters on an instance-by-instance basis would lead to improved results.

To evaluate the NDVNS algorithm, we used five state-of-the-art CCP algorithms as the reference methods, including GRASP+TS [28], FITS [42], IVNS [22], GVNS [4], and SGVNS [4]. The source codes of GRAPS+TS and IVNS are available at `http://www.info.univ-angers.fr/pub/hao/CCP.html`, while the source codes of GVNS and SGVNS are provided by their authors at `http:www.mi.sanu.ac.rs/~nenad/ccp`. The source code of the NDVNS algorithm will be available at `http://www.info.univ-angers.fr/pub/hao/NDVNS.html`. All compared algorithms (written in C++) were compiled by the same g++ compiler with the "-O3" option.

All the computational experiments are based on the same computing platform with an Intel E5-2670 processor, running Linux. Due to the stochastic feature of the algorithms, each algorithm was run 20 times with different random seeds for each instance, and the stopping condition for one run is a maximum time limit $t_{max}$ set to be $N$ seconds, where $N$ is the number of vertices in the benchmark instance. To run the reference algorithms, we used the parameter settings that were calibrated by their authors and provided in the corresponding papers.

### 4.3 Computational Results on the Small and Medium Instances

The computational results of the compared algorithms on the 40 small or medium-sized instances with $N = 240$ or 480 (i.e., the sets Ran240 and Ran480) are summarized in Table 2. Column 1 of Table 2 gives the name of each instance. Columns 2-7 and 8-13 report respectively the best and average objective values ($f_{best}$ and $f_{avg}$) over 20 runs of each reference algorithm and the NDVNS algorithm, which are considered as two important performance indicators of algorithms. The row "Avg." shows the average values for each column, and the row "#best" indicates the number of instances for which the associated algorithm obtains the best value in term of $f_{avg}$ or $f_{best}$ among all the compared algorithms. To check whether there exists a significant difference between the NDVNS algorithm and each reference algorithm in terms of $f_{avg}$ and $f_{best}$, we report the $p$-values from the non-parametric Friedman tests in the last row, where a $p$-value smaller than 0.05 indicates a significant difference between the compared results. In addition, for each instance, the best value in terms of $f_{best}$ and $f_{avg}$ among the compared results are indicated in bold.

Table 2 shows that for the instances with $N \leq 480$, the NDVNS algorithm outperforms the reference algorithms in terms of the best objective value $f_{best}$. Specifically, for the 40 instances tested, the reference algorithms respectively obtained the best value (in bold) in terms of $f_{best}$ for 0, 10, 0, 2, and 9 instances, against 26 instances for the NDVNS algorithm. Moreover, the small $p$-values ($< 0.05$) confirm the statistical significance of these differences in terms of $f_{best}$. On the other hand, the NDVNS algorithm outperforms significantly GRASP+TS, FITS, GVNS, and IVNS in terms of $f_{avg}$, but performs worse than SGVNS. Specifically, the reference algorithms obtained respectively the best value (in bold) in terms of $f_{avg}$ for 0, 5, 0, 0, and 24 instances, against 11 instances for the NDVNS algorithm. We conclude that for the instances with $N \leq 480$, NDVNS is the best algorithm for attaining the best objective values ($f_{best}$) while SGVNS is the leading algorithm in terms of average results ($f_{avg}$).

### 4.4 Computational Results on the Large-Scale Instances

The computational results of the compared algorithms on the 70 large-scale instances (RanReal960, MDG-a, and MDG-a-40) are summarized in Tables 3-5 respectively with the same information as in Table 2.

Table 3 indicates that for the RanReal960 instances, the NDVNS algorithm outperforms all reference algorithms. In terms of $f_{best}$, NDVNS obtained the

Table 2. Comparisons of the proposed NDVNS algorithm with the state-of-the-art algorithms on the 40 RanReal instances with $N = 240, 480$ and $K = 12, 20$ (i.e., the sets RanReal240 and RanReal480), where the best results among the compared algorithms are indicated in bold in terms of $f_{best}$ and $f_{avg}$.

| | $f_{best}$ | | | | | | $f_{avg}$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance | GRASP+TS | FITS | GVNS | IVNS | SGVNS | NDVNS | GRASP+TS | FITS | GVNS | IVNS | SGVNS | NDVNS |
| RanReal240_01 | 222810.00 | 224941.48 | 224865.59 | 224909.66 | **224991.62** | 224949.51 | 221724.79 | 224802.06 | 224210.83 | 224662.19 | **224814.37** | 224689.70 |
| RanReal240_02 | 202090.84 | **204624.36** | 204162.57 | **204624.36** | **204624.36** | 204563.81 | 200790.17 | 204359.38 | 203578.52 | 204264.77 | **204460.21** | 204214.81 |
| RanReal240_03 | 196579.03 | 198954.91 | 198325.22 | 198953.39 | 198968.52 | **198976.88** | 195053.24 | **198799.84** | 197890.20 | 198610.16 | 198764.53 | 198681.37 |
| RanReal240_04 | 222971.50 | 225627.16 | 225184.60 | 225602.83 | **225683.17** | 225618.17 | 220694.59 | 225364.97 | 224699.99 | 225143.17 | **225468.30** | 225185.91 |
| RanReal240_05 | 193205.62 | **195564.48** | 194953.42 | 195449.78 | 195502.11 | 195539.89 | 191597.68 | 195320.28 | 194504.84 | 195198.55 | 195277.85 | **195401.63** |
| RanReal240_06 | 214234.61 | **216747.32** | 216606.01 | 216702.92 | 216744.91 | **216747.32** | 212912.04 | 216487.02 | 215763.06 | 216461.87 | **216582.02** | 216173.65 |
| RanReal240_07 | 207375.32 | **209305.70** | 209099.87 | 209102.72 | 209154.37 | 209273.70 | 206170.82 | 209029.23 | 208531.74 | 208909.32 | 209004.11 | **209063.40** |
| RanReal240_08 | 202784.06 | **205246.82** | 204901.30 | 205151.55 | **205246.82** | **205246.82** | 201715.83 | 204961.05 | 204253.34 | 204889.43 | 204970.01 | **205099.31** |
| RanReal240_09 | 207693.87 | 209159.16 | 208767.78 | 209083.53 | 209050.16 | **209186.90** | 206182.58 | 208952.48 | 208247.16 | 208835.09 | 208849.23 | **208987.58** |
| RanReal240_10 | 190200.01 | **192986.21** | 192618.97 | 192955.92 | 192978.46 | 192948.92 | 188189.99 | **192811.13** | 191822.84 | 192625.24 | 192803.32 | 192523.02 |
| RanReal240_11 | 203364.04 | **204722.75** | 204345.37 | **204722.75** | 204630.14 | **204722.75** | 201749.12 | **204559.39** | 203959.71 | 204448.00 | 204523.11 | 204398.23 |
| RanReal240_12 | 200333.50 | **201117.11** | 200734.01 | 201028.32 | 201074.54 | 200978.99 | 199070.70 | 200797.67 | 200253.32 | 200808.83 | **200958.23** | 200737.16 |
| RanReal240_13 | 201108.59 | 202335.99 | 202038.43 | 202291.55 | 202338.43 | **202345.12** | 199404.67 | 202139.57 | 201324.44 | 202016.87 | 202147.58 | **202276.80** |
| RanReal240_14 | 227468.88 | 228870.89 | 228638.42 | 228833.77 | **228971.03** | 228870.89 | 225655.27 | 228554.78 | 228011.53 | 228509.80 | **228601.70** | 228396.82 |
| RanReal240_15 | 188345.98 | 191255.87 | 190517.80 | 191212.67 | 191235.85 | **191263.28** | 187228.97 | 190923.28 | 190100.95 | 190851.45 | 190944.65 | **190945.28** |
| RanReal240_16 | 203029.88 | 204054.99 | 203550.14 | 204054.99 | **204081.46** | 204074.95 | 200499.46 | 203710.39 | 203125.46 | 203721.09 | 203873.05 | **203918.76** |
| RanReal240_17 | 193529.71 | **195561.36** | 194904.42 | 195376.18 | 195482.75 | 195509.13 | 191835.76 | **195243.32** | 194367.98 | 194971.81 | 195179.89 | 195236.75 |
| RanReal240_18 | 192339.64 | 195100.39 | 194567.39 | 195058.65 | 195100.39 | **195167.14** | 190866.13 | 194872.13 | 193968.19 | 194768.26 | **194984.15** | 194886.16 |
| RanReal240_19 | 196905.06 | **199225.98** | 198745.58 | 199199.66 | 199216.94 | 199216.46 | 195640.94 | **199040.43** | 198201.50 | 198855.80 | 199020.65 | 198895.83 |
| RanReal240_20 | 210631.70 | 212268.52 | 211997.40 | 212264.10 | 212276.27 | **212323.22** | 208865.66 | 212049.85 | 211370.04 | 212005.52 | 212093.34 | **212130.13** |
| RanReal480_01 | 545668.36 | 555489.92 | 553991.55 | 554338.83 | 555793.77 | **556639.68** | 540568.89 | 554376.54 | 552286.37 | 553689.96 | 555241.85 | **555566.55** |
| RanReal480_02 | 501911.69 | 511280.50 | 509102.66 | 510182.84 | 511543.39 | **511666.95** | 496863.58 | 509757.15 | 507704.63 | 509206.46 | **510562.04** | 510495.78 |
| RanReal480_03 | 489475.62 | 497295.19 | 495712.55 | 496325.47 | 497469.82 | **497846.57** | 484361.46 | 496059.50 | 494154.03 | 495447.78 | **496912.03** | 495871.46 |
| RanReal480_04 | 512283.76 | 522305.16 | 520103.37 | 521500.43 | 522603.58 | **522748.49** | 507423.96 | 521062.13 | 519091.59 | 520629.45 | **522109.62** | 520295.34 |
| RanReal480_05 | 473902.30 | 484084.66 | 483098.17 | 484059.38 | 484427.71 | **484742.51** | 469433.52 | 482867.74 | 480688.48 | 482785.11 | **483849.50** | 482831.18 |
| RanReal480_06 | 525154.13 | 533991.27 | 532349.86 | 533672.54 | 534854.50 | **535503.61** | 519394.89 | 533036.36 | 531185.95 | 532634.40 | **534116.68** | 533138.78 |
| RanReal480_07 | 537665.73 | 545470.73 | 543881.85 | 545049.90 | 546021.46 | **546951.97** | 532780.90 | 544651.12 | 542316.03 | 543940.84 | **545552.45** | 545548.47 |
| RanReal480_08 | 523533.03 | 532417.42 | 530879.18 | 532498.92 | 532919.04 | **533098.56** | 518717.75 | 531667.91 | 529996.90 | 531466.59 | **532484.80** | 531631.32 |
| RanReal480_09 | 545770.15 | 556868.85 | 554529.84 | 555865.08 | 557015.33 | **557283.90** | 540851.84 | 555634.40 | 553376.73 | 555133.56 | **556374.85** | 555998.60 |
| RanReal480_10 | 507938.39 | 520257.54 | 517716.48 | 519254.08 | 520050.26 | **520481.15** | 504335.05 | 518071.71 | 516504.03 | 518334.39 | **519300.06** | 518812.29 |
| RanReal480_11 | 513872.14 | 523991.29 | 522832.50 | 523692.55 | **524612.99** | 524059.91 | 510058.34 | 522816.94 | 521118.41 | 522286.36 | **523441.91** | 522416.53 |
| RanReal480_12 | 494166.33 | 501915.56 | 500306.91 | 501257.98 | 502311.18 | **502656.68** | 487968.01 | 500776.79 | 498878.68 | 500079.78 | **501732.55** | 501171.81 |
| RanReal480_13 | 527226.32 | 535025.51 | 533602.93 | 535147.03 | 535433.73 | **535633.80** | 522118.55 | 533823.79 | 532192.01 | 533799.68 | **534597.90** | 533605.55 |
| RanReal480_14 | 504748.63 | 514107.62 | 513131.26 | 514289.38 | **514754.52** | 514696.90 | 498713.57 | 513053.25 | 511238.18 | 513205.86 | **513812.63** | 512600.42 |
| RanReal480_15 | 507714.79 | 517205.02 | 515929.15 | 516882.46 | 518189.67 | **518605.32** | 504238.44 | 516018.38 | 514632.06 | 515774.99 | **517120.91** | 516455.65 |
| RanReal480_16 | 541051.74 | 549552.63 | 548667.87 | 549062.80 | 550020.05 | **550482.48** | 535460.06 | 548462.13 | 546878.21 | 548118.45 | **549501.45** | 549108.44 |
| RanReal480_17 | 528069.56 | 537924.55 | 535537.34 | 537724.54 | **538413.66** | 538331.26 | 524318.85 | 536745.39 | 534520.72 | 536697.21 | **537646.31** | 536829.04 |
| RanReal480_18 | 520131.03 | 525822.76 | 524382.73 | 525927.41 | 526060.02 | **526466.23** | 512918.86 | 524712.42 | 522994.01 | 524679.62 | **525643.80** | 523932.37 |
| RanReal480_19 | 513423.61 | 522316.22 | 521405.92 | 521771.19 | 523072.19 | **523219.84** | 508355.24 | 521267.22 | 519722.66 | 520981.05 | 522079.34 | **522114.62** |
| RanReal480_20 | 510421.43 | 518349.10 | 516872.49 | 518104.06 | 519030.59 | **519492.21** | 506585.04 | 517430.77 | 515563.34 | 517117.02 | 518171.15 | **518475.86** |
| Avg. | 360028.26 | 365583.57 | 364588.97 | 365329.65 | 365798.74 | **365953.30** | 356782.88 | 364876.75 | 363580.72 | 364664.14 | **365339.30** | 364968.56 |
| #best | 0 | 10 | 0 | 2 | 9 | **26** | 0 | 5 | 0 | 0 | **24** | 11 |
| *p-value* | 2.54E-10 | 8.58E-04 | 2.54E-10 | 1.26E-07 | 6.49E-03 | | 2.54E-10 | 7.50E-01 | 2.54E-10 | 1.60E-03 | 1.10E-02 | |

Table 3. Comparisons of the proposed NDVNS algorithm with the state-of-the-art algorithms on the 30 RanReal instances with $N = 960$ and $K = 30, 40, 60$, where the best results among the compared algorithms are indicated in bold in terms of $f_{best}$ and $f_{avg}$.

| | $f_{best}$ | | | | | | $f_{avg}$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance | GRASP+TS | FITS | GVNS | IVNS | SGVNS | NDVNS | GRASP+TS | FITS | GVNS | IVNS | SGVNS | NDVNS |
| RanReal960_01.30 | 1302543.22 | 1333878.00 | 1332036.22 | 1332375.23 | 1336665.95 | **1340369.47** | 1291204.57 | 1332712.78 | 1327073.81 | 1329411.05 | 1335446.91 | **1338452.93** |
| RanReal960_02.30 | 1399613.73 | 1434529.49 | 1426791.19 | 1428044.02 | 1433360.74 | **1435819.84** | 1390367.02 | **1433886.30** | 1423733.02 | 1425389.43 | 1432047.47 | 1433258.34 |
| RanReal960_03.30 | 1358836.08 | 1392101.18 | 1387866.70 | 1390305.75 | 1395487.59 | **1398554.78** | 1352211.68 | 1390924.21 | 1385807.54 | 1388440.80 | 1393201.91 | **1397154.11** |
| RanReal960_04.30 | 1380481.67 | 1414344.67 | 1405324.24 | 1407646.15 | 1412648.79 | **1414919.86** | 1368890.28 | 1412460.01 | 1402324.80 | 1403544.53 | 1411605.44 | **1412464.18** |
| RanReal960_05.30 | 1330822.53 | 1365975.96 | 1362127.27 | 1363634.81 | 1368244.88 | **1372686.88** | 1325777.12 | 1365612.74 | 1359136.52 | 1361226.08 | 1366758.19 | **1370958.72** |
| RanReal960_06.30 | 1384180.44 | 1413476.58 | 1411006.67 | 1413239.83 | 1417387.67 | **1420632.38** | 1376947.36 | 1412750.08 | 1407559.72 | 1410740.78 | 1414582.86 | **1419467.40** |
| RanReal960_07.30 | 1307177.23 | 1334504.35 | 1330185.72 | 1334760.33 | 1340008.67 | **1341829.68** | 1298766.83 | 1334263.93 | 1328256.40 | 1332221.59 | 1338015.99 | **1340275.06** |
| RanReal960_08.30 | 1429185.56 | 1463737.39 | 1460663.08 | 1461955.43 | 1465223.13 | **1469545.99** | 1423559.02 | 1462602.72 | 1457841.83 | 1458971.76 | 1463741.46 | **1466830.82** |
| RanReal960_09.30 | 1345054.26 | 1381577.32 | 1378025.58 | 1381773.61 | 1383808.27 | **1387514.11** | 1337716.88 | 1379280.98 | 1374688.23 | 1376775.51 | 1381619.80 | **1385397.31** |
| RanReal960_10.30 | 1346791.35 | 1379905.83 | 1376643.20 | 1378708.62 | 1384879.18 | **1386801.94** | 1339619.95 | 1378772.67 | 1373304.95 | 1375602.35 | 1382692.19 | **1384724.79** |
| RanReal960_01.40 | 1003828.57 | 1035642.67 | 1033924.75 | 1035920.06 | 1041309.74 | **1042735.72** | 998600.42 | 1034626.82 | 1030740.76 | 1032252.51 | 1039766.18 | **1040717.99** |
| RanReal960_02.40 | 1083570.20 | 1110547.59 | 1109463.37 | 1109971.06 | 1116191.87 | **1117471.02** | 1078837.80 | 1110074.36 | 1107175.40 | 1107446.56 | 1114832.29 | **1115602.03** |
| RanReal960_03.40 | 1055667.54 | 1083240.15 | 1082030.22 | 1081863.50 | 1086482.20 | **1089012.05** | 1048691.15 | 1082948.77 | 1077034.60 | 1079126.11 | 1084643.46 | **1087325.53** |
| RanReal960_04.40 | 1067561.21 | **1103897.12** | 1095016.97 | 1095312.16 | 1101681.52 | 1102222.98 | 1062345.63 | **1101073.37** | 1090876.67 | 1093282.27 | 1099911.61 | 1100256.11 |
| RanReal960_05.40 | 1032397.85 | 1059158.09 | 1056947.72 | 1060197.52 | 1063110.82 | **1066415.88** | 1025479.31 | 1058478.46 | 1054206.46 | 1056694.74 | 1061384.97 | **1064390.34** |
| RanReal960_06.40 | 1071581.47 | 1100368.74 | 1097581.21 | 1097171.57 | 1104641.18 | **1107955.38** | 1067083.42 | 1100064.66 | 1095148.70 | 1095176.04 | 1101826.91 | **1105069.06** |
| RanReal960_07.40 | 1009592.48 | 1043376.95 | 1033729.55 | 1035283.33 | 1041749.32 | **1043921.14** | 1001861.97 | 1039933.23 | 1030351.23 | 1033832.11 | 1040048.74 | **1041464.08** |
| RanReal960_08.40 | 1111574.84 | 1139865.05 | 1137662.34 | 1136873.61 | 1142526.37 | **1144615.34** | 1105353.09 | 1138054.46 | 1134480.64 | 1135397.85 | 1140665.27 | **1141877.82** |
| RanReal960_09.40 | 1041792.13 | 1072116.14 | 1068850.92 | 1070778.57 | 1074619.79 | **1076786.59** | 1034546.35 | 1071426.07 | 1065592.65 | 1066941.06 | 1072735.34 | **1075606.23** |
| RanReal960_10.40 | 1044783.86 | 1072919.65 | 1069919.92 | 1070508.58 | 1077364.51 | **1079212.31** | 1039104.29 | 1071480.21 | 1067628.95 | 1067681.56 | 1074401.50 | **1077123.00** |
| RanReal960_01.60 | 701475.52 | 727690.58 | 725386.78 | 726988.50 | 733190.77 | **733900.43** | 696805.31 | 727222.73 | 723565.85 | 724946.22 | 731400.44 | **731787.78** |
| RanReal960_02.60 | 747208.49 | 773921.97 | 770551.20 | 770449.77 | 776021.36 | **776085.50** | 742477.71 | 772572.49 | 768160.68 | 768690.81 | 774605.32 | **774619.17** |
| RanReal960_03.60 | 729297.62 | 756677.95 | 755581.66 | 754009.92 | 759491.06 | **760728.09** | 725109.89 | 755442.64 | 751360.85 | 751446.17 | 757871.25 | **758924.49** |
| RanReal960_04.60 | 740886.98 | 765253.27 | 763611.42 | 764207.26 | 769137.77 | **770170.61** | 735212.70 | 764696.30 | 760580.84 | 762573.42 | 767014.55 | **768057.59** |
| RanReal960_05.60 | 720834.30 | 743715.56 | 742147.50 | 740969.22 | 748127.79 | **748244.40** | 713528.27 | 743165.17 | 738845.83 | 739350.58 | 746494.10 | **747006.91** |
| RanReal960_06.60 | 737868.00 | 763029.06 | 761468.81 | 763296.98 | 766505.39 | **768180.66** | 732621.14 | 761957.51 | 758327.43 | 760310.08 | 765019.53 | **765223.34** |
| RanReal960_07.60 | 702194.79 | 725993.23 | 723514.49 | 725295.80 | 728860.48 | **731064.84** | 696061.71 | 725733.17 | 720341.39 | 722576.48 | 726764.50 | **728984.94** |
| RanReal960_08.60 | 763404.07 | 791334.42 | 788995.32 | 788780.69 | 793474.88 | **794354.56** | 758541.82 | 790285.98 | 785680.65 | 786851.43 | **791966.22** | 791262.07 |
| RanReal960_09.60 | 725949.51 | 750858.18 | 747919.82 | 748331.26 | 752945.57 | **754578.20** | 720001.72 | 749209.83 | 745292.43 | 745747.41 | 751523.98 | **753033.20** |
| RanReal960_10.60 | 723950.05 | 749883.45 | 748987.54 | 748649.34 | 753154.39 | **755076.90** | 720415.01 | 748985.05 | 746071.57 | 747293.79 | 751855.09 | **753154.35** |
| Avg. | 1046670.19 | 1076117.35 | 1072798.71 | 1073910.08 | 1078943.39 | **1081046.92** | 1040257.98 | 1075023.26 | 1069706.35 | 1071331.37 | 1077148.12 | **1079015.66** |
| #best | 0 | 1 | 0 | 0 | 0 | **29** | 0 | 2 | 0 | 0 | 1 | **27** |
| *p-value* | 4.32E-08 | 3.19E-07 | 4.32E-08 | 4.32E-08 | 4.32E-08 | | 4.32E-08 | 2.07E-06 | 4.32E-08 | 4.32E-08 | 3.19E-07 | |

Table 4
Comparisons of the proposed algorithm with the state-of-the-art algorithms on the 20 MDG-a instances with $N = 2000$ and $K = 50$, where the best results among the compared algorithms are indicated in bold in terms of $f_{best}$ and $f_{avg}$.

| | $f_{best}$ | | | | | | $f_{avg}$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance | GRASP+TSFITS | | GVNS | IVNS | SGVNS | NDVNS | GRASP+TSFITS | | GVNS | IVNS | SGVNS | NDVNS |
| MDG-a_21 | 369103 | 383164 | 388242 | 379513 | 389845 | **391259** | 363528.40 | 382387.05 | 387751.50 | 377608.90 | 389471.25 | **390647.15** |
| MDG-a_22 | 367627 | 380817 | 385196 | 382052 | 386956 | **388476** | 364577.70 | 380489.60 | 384455.70 | 377874.60 | 386325.55 | **387718.60** |
| MDG-a_23 | 369009 | 380630 | 386046 | 379366 | 388159 | **389363** | 365738.50 | 379334.85 | 385496.80 | 375544.75 | 387518.10 | **388908.60** |
| MDG-a_24 | 370891 | 381783 | 387680 | 379047 | 388799 | **390269** | 363101.70 | 380560.55 | 386803.60 | 377026.10 | 388546.25 | **389655.40** |
| MDG-a_25 | 380735 | 390623 | 396571 | 388897 | 398327 | **399478** | 375489.75 | 386346.05 | 396070.55 | 386168.00 | 397873.45 | **399002.40** |
| MDG-a_26 | 383826 | 393222 | 400519 | 392935 | 402295 | **403457** | 380869.35 | 392916.10 | 399804.40 | 390523.15 | 401795.95 | **402740.95** |
| MDG-a_27 | 365606 | 377617 | 380535 | 378129 | 381291 | **383780** | 362665.50 | 376721.60 | 379874.45 | 373828.35 | 380868.30 | **382951.05** |
| MDG-a_28 | 370385 | 380748 | 386077 | 380642 | 386986 | **389025** | 364047.95 | 379732.95 | 385251.70 | 376880.50 | 386453.65 | **388516.50** |
| MDG-a_29 | 366279 | 378239 | 382417 | 376229 | 383418 | **385316** | 361792.25 | 377540.40 | 381512.10 | 371959.05 | 382756.50 | **384816.05** |
| MDG-a_30 | 380142 | 389452 | 395759 | 389449 | 397325 | **398211** | 377323.00 | 389254.10 | 394851.85 | 387218.35 | 396433.40 | **397776.80** |
| MDG-a_31 | 369944 | 379407 | 385176 | 379815 | 386039 | **388375** | 365812.55 | 378922.70 | 384524.35 | 375535.30 | 385337.85 | **387562.05** |
| MDG-a_32 | 372470 | 383333 | 391777 | 382573 | 393268 | **394611** | 368952.30 | 382737.35 | 391092.15 | 380767.40 | 392862.30 | **394031.35** |
| MDG-a_33 | 368150 | 379047 | 383038 | 379938 | 384568 | **385806** | 363019.50 | 376641.80 | 382235.55 | 374720.60 | 383941.90 | **385212.70** |
| MDG-a_34 | 373454 | 386544 | 393926 | 387577 | 394413 | **396725** | 370093.50 | 385459.25 | 392935.35 | 382141.40 | 393869.45 | **395823.10** |
| MDG-a_35 | 376886 | 386347 | 393507 | 384031 | 394771 | **396054** | 373400.00 | 385302.60 | 392811.75 | 382317.85 | 393963.25 | **395455.20** |
| MDG-a_36 | 384916 | 396368 | 400766 | 395439 | 402072 | **403604** | 376033.60 | 394103.05 | 399943.25 | 390926.50 | 401657.85 | **403111.10** |
| MDG-a_37 | 372157 | 386195 | 387114 | 385363 | 388769 | **390289** | 369583.10 | 385797.30 | 386444.35 | 382197.80 | 388192.80 | **389612.25** |
| MDG-a_38 | 378374 | 388712 | 394526 | 385978 | 396163 | **397407** | 371013.30 | 387807.15 | 393843.30 | 384883.95 | 395810.45 | **396710.05** |
| MDG-a_39 | 374068 | 385975 | 390260 | 387159 | 391915 | **393415** | 371801.55 | 383036.10 | 389648.75 | 381376.55 | 390930.80 | **392799.15** |
| MDG-a_40 | 387742 | 396321 | 404520 | 394522 | 405680 | **407084** | 384148.60 | 395505.80 | 403704.15 | 392126.10 | 405348.25 | **406615.75** |
| Avg. | 374088 | 385227 | 390683 | 384433 | 392053 | **393600.20** | 369649.61 | 384029.82 | 389952.78 | 381081.26 | 391497.87 | **392983.31** |
| #best | 0 | 0 | 0 | 0 | 0 | **20** | 0 | 0 | 0 | 0 | 0 | **20** |
| *p-value* | 7.74E-06 | 7.74E-06 | 7.74E-06 | 7.74E-06 | 7.74E-06 | | 7.74E-06 | 7.74E-06 | 7.74E-06 | 7.74E-06 | 7.74E-06 | |

Table 5
Comparisons of the proposed algorithm with the state-of-the-art algorithms on the 20 MDG-a-40 instances with $N = 2000$ and $K = 40$, where the best results among the compared algorithms are indicated in bold in terms of $f_{best}$ and $f_{avg}$.

| | $f_{best}$ | | | | | $f_{avg}$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instance | GRASP+TS | GVNS | IVNS | SGVNS | NDVNS | GRASP+TS | GVNS | IVNS | SGVNS | NDVNS |
| MDG-a_21.40 | 437628 | 458203 | 449022 | 459655 | **461300** | 435028.95 | 457507.35 | 445994.60 | 459018.80 | **460489.15** |
| MDG-a_22.40 | 434004 | 454485 | 447832 | 455658 | **457462** | 429590.60 | 453702.10 | 444332.10 | 455257.05 | **456962.95** |
| MDG-a_23.40 | 433885 | 455745 | 448024 | 456318 | **458604** | 429239.65 | 455005.25 | 444144.45 | 455876.95 | **457849.00** |
| MDG-a_24.40 | 432413 | 457319 | 446320 | 458037 | **460224** | 426743.05 | 456093.20 | 444043.70 | 457644.95 | **459361.60** |
| MDG-a_25.40 | 443602 | 467531 | 458432 | 468641 | **470546** | 438365.10 | 466648.55 | 455520.40 | 468224.85 | **469807.70** |
| MDG-a_26.40 | 451858 | 472182 | 466960 | 473489 | **474918** | 442170.55 | 471496.70 | 462133.90 | 472921.25 | **474434.10** |
| MDG-a_27.40 | 426581 | 449333 | 443892 | 450753 | **451999** | 423135.80 | 448546.30 | 440947.35 | 450218.55 | **451516.85** |
| MDG-a_28.40 | 433276 | 455484 | 449530 | 455923 | **458620** | 429343.70 | 454500.60 | 446270.70 | 455332.30 | **457840.85** |
| MDG-a_29.40 | 426057 | 451006 | 444445 | 451617 | **454117** | 422597.40 | 450200.25 | 439232.10 | 450966.00 | **453453.80** |
| MDG-a_30.40 | 445550 | 466359 | 459836 | 466361 | **468982** | 438784.85 | 465028.80 | 456448.10 | 465895.50 | **468292.20** |
| MDG-a_31.40 | 431695 | 454500 | 447463 | 455835 | **457570** | 428142.10 | 454009.15 | 442404.05 | 455311.90 | **456756.95** |
| MDG-a_32.40 | 441853 | 462193 | 454347 | 463455 | **464911** | 431007.95 | 461403.60 | 449071.55 | 462833.70 | **464399.70** |
| MDG-a_33.40 | 433898 | 451683 | 446617 | 452969 | **454717** | 428970.00 | 450944.00 | 442452.45 | 452552.90 | **454069.95** |
| MDG-a_34.40 | 373598 | 393515 | 383611 | 395198 | **396610** | 368232.75 | 392736.80 | 381652.00 | 394776.85 | **395850.30** |
| MDG-a_35.40 | 445028 | 463392 | 455965 | 464883 | **466869** | 440927.55 | 462869.85 | 451299.60 | 464526.40 | **466308.65** |
| MDG-a_36.40 | 452963 | 471675 | 467716 | 473135 | **474665** | 445388.75 | 471188.60 | 460190.30 | 472601.50 | **474018.35** |
| MDG-a_37.40 | 437611 | 456642 | 453007 | 458099 | **459380** | 428557.25 | 455860.90 | 449737.50 | 457121.15 | **458843.50** |
| MDG-a_38.40 | 445151 | 465807 | 455905 | 466562 | **468630** | 441406.40 | 464713.85 | 453176.75 | 465750.15 | **468039.25** |
| MDG-a_39.40 | 441583 | 459938 | 457019 | 460785 | **462909** | 437246.90 | 459021.00 | 449321.25 | 460127.15 | **462352.60** |
| MDG-a_40.40 | 455220 | 476752 | 466277 | 477949 | **479308** | 449236.45 | 475739.55 | 460905.65 | 477365.60 | **478777.80** |
| Avg. | 436173 | 457187.20 | 450111.00 | 458266.10 | **460117.05** | 430705.79 | 456360.82 | 445963.93 | 457716.18 | **459471.26** |
| #best | 0 | 0 | 0 | 0 | **20** | 0 | 0 | 0 | 0 | **20** |
| *p-value* | 7.74E-06 | 7.74E-06 | 7.74E-06 | 7.74E-06 | | 7.74E-06 | 7.74E-06 | 7.74E-06 | 7.74E-06 | |

best value for 29 out of 30 instances, against 0, 1, 0, 0 and 0 instances for the reference algorithms. In terms of $f_{avg}$, NDVNS reported the best value for 27 out of 30 instances, while the reference algorithms obtained the best value only for 0, 2, 0, 0 and 1 instances respectively. Moreover, the small $p$-values further confirm the statistical significance of the differences between the NDVNS algorithm and the reference algorithms.

Moreover, Tables 4 and 5 show an even stronger dominance of NDVNS over the reference algorithms for the largest MDG-a and MDG-a-40 instances with

$N = 2000$, by reporting the best $f_{best}$ and $f_{avg}$ values for all the instances.

Given that the computational experiments and comparisons are based on the same computing platform and the same stopping condition, these results clearly indicate that the NDVNS algorithm is the best algorithm for solving large instances with at least $N = 960$ vertices and $K = 30$ clusters. Interestingly, the improvement of NDVNS over the existing CCP algorithms increases with the instance size. This implies that NDVNS could be a useful tool that can be applied to solve large scale practical problems with a high number of candidate elements to be grouped into many clusters.

## 5 Discussions and Analyses

This section investigates the impacts of key components of the NDVNS algorithm, and the spatial distribution of high-quality local optimum solutions.

### *5.1 Parameter Sensitivity Analysis*

The NDVNS algorithm relies on two important parameters $Q$ and $\alpha$, which are studied in this subsection.

#### *5.1.1 Sensitivity Analysis of the Parameter Q*

As explained in Section 3.1, a particular feature of the NDVNS algorithm concerns the use of the probability $Q$ to control the perturbation strength $k$ of the shake procedure where a smaller $Q$ means a stronger diversification ability of the algorithm and vice-versa. To analyze the impact of $Q$ on the performance of the algorithm and find a proper $Q$ value, we carried out an experiment based on 40 representative instances. In this experiment, we varied the value of $Q$ in the range $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ and ran the corresponding NDVNS algorithm 20 times for each possible value of $Q$ and each instance according to the experimental protocol of Section 4.2. The computational results are summarized in Table 6. The first column and first row of the table give respectively the names of instances and the values of $Q$, and columns 2–11 respectively report the average objective value ($f_{avg}$) over 20 independent runs of the NDVNS algorithm for each instance and each $Q$ value. In addition, the row "Avg." shows the average result for each column, and the row "#best" shows the number of instances for which the associated $Q$ value generated the best result in terms of $f_{avg}$ among the tested $Q$ values.

Table 6 indicates that the performance of the NDVNS algorithm is sensitive

Table 6. Influence of parameter $Q$ on the average results $f_{avg}$ with the best result indicated in bold for each instance.

| Instance/$Q$ | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RanReal480_01 | 554802.93 | 555023.54 | **555789.72** | 554723.46 | 555232.19 | 555670.98 | 555104.65 | 555408.22 | 555018.77 | 555421.39 | 555377.40 |
| RanReal480_02 | 509014.98 | 509535.67 | 509772.55 | 509667.82 | 509935.59 | **510124.19** | 509838.16 | 509927.32 | 510029.50 | 509294.21 | 509880.39 |
| RanReal480_03 | 495281.64 | 495859.26 | 495506.48 | 495597.58 | 496063.48 | 496022.28 | 495582.78 | 496247.92 | 496229.91 | 495787.66 | **496265.75** |
| RanReal480_04 | 519536.11 | **521138.39** | 520455.13 | 520506.16 | 520903.78 | 520933.39 | 520792.86 | 519886.15 | 520393.51 | 520352.16 | 520245.59 |
| RanReal480_05 | 482128.38 | 482756.49 | 483209.88 | 481857.69 | 483155.50 | 481991.90 | **483331.88** | 482672.49 | 483126.07 | 482576.77 | 482258.38 |
| RanReal960_01.30 | 1337569.34 | 1337842.45 | **1339017.66** | 1338194.50 | 1338687.01 | 1338343.70 | 1337824.90 | 1338577.17 | 1338652.58 | 1338060.94 | 1336941.32 |
| RanReal960_02.30 | 1432803.15 | 1433493.70 | **1433680.68** | 1432631.01 | 1433645.38 | 1432468.31 | 1433099.30 | 1432679.45 | 1433276.29 | 1432789.55 | 1432231.37 |
| RanReal960_03.30 | 1395898.38 | 1396241.28 | **1396638.38** | 1396257.53 | 1396167.43 | 1396393.31 | 1395966.77 | 1395882.92 | 1395808.30 | 1395536.20 | 1395671.53 |
| RanReal960_04.30 | 1411870.15 | 1412050.58 | 1412634.09 | 1412594.34 | 1412380.22 | 1412519.69 | 1412628.23 | **1412763.69** | 1412155.07 | 1412600.49 | 1412079.18 |
| RanReal960_05.30 | 1369934.30 | 1370255.07 | 1370720.53 | 1370500.10 | 1370209.04 | 1370550.79 | **1370847.70** | 1370019.32 | 1370254.66 | 1369782.24 | 1370264.08 |
| RanReal960_01.40 | 1040223.81 | 1040308.50 | 1040612.23 | **1040846.80** | 1040519.46 | 1040752.52 | 1040666.79 | 1041303.23 | 1040481.92 | 1040468.49 | 1040441.10 |
| RanReal960_02.40 | 1114481.64 | 1114711.79 | 1115041.84 | **1115539.96** | 1114725.16 | 1114994.71 | 1115392.71 | 1115516.04 | 1115246.03 | 1115388.05 | 1115521.83 |
| RanReal960_03.40 | 1086787.02 | 1086898.10 | **1087541.32** | 1087521.73 | 1087244.42 | 1087384.09 | 1086639.57 | 1087026.45 | 1087096.05 | 1087181.95 | 1087410.84 |
| RanReal960_04.40 | 1099410.32 | 1099834.55 | 1099838.12 | 1099552.96 | 1100290.91 | 1100238.65 | 1100121.08 | **1100829.11** | 1100142.72 | 1099788.37 | 1099941.40 |
| RanReal960_05.40 | 1064123.23 | **1065382.72** | 1064691.02 | 1064245.36 | 1064582.61 | 1064564.68 | 1064804.55 | 1064344.32 | 1064295.92 | 1064106.80 | 1064622.74 |
| RanReal960_01.60 | 731015.35 | 730765.15 | 731521.55 | 731508.34 | 731171.48 | 731617.51 | 731456.04 | 731777.19 | 731905.48 | **732096.98** | 731063.23 |
| RanReal960_02.60 | 773986.01 | 773757.64 | 774281.72 | 774161.40 | 774351.84 | 774780.23 | 774413.99 | **774818.84** | 774642.05 | 774787.10 | 774659.53 |
| RanReal960_03.60 | 758534.27 | 758378.13 | 758902.29 | 758945.84 | 758733.51 | 758980.24 | 759250.55 | **759672.30** | 759330.76 | 759307.84 | 759182.32 |
| RanReal960_04.60 | 767654.76 | 767583.41 | 767926.66 | 767674.49 | 768306.38 | 767794.52 | 768262.45 | 768393.12 | 768444.39 | **768770.44** | 768577.77 |
| RanReal960_05.60 | 746765.97 | 746814.23 | 746688.21 | 746883.25 | 747074.31 | 747162.63 | 747247.68 | **747606.74** | 747256.53 | 747518.81 | 747157.78 |
| MDG-a_21 | 390845.20 | **390877.65** | 390793.55 | 390330.05 | 390365.65 | 390573.65 | 390238.25 | 390208.45 | 390313.70 | 390055.10 | 390199.60 |
| MDG-a_22 | 387934.80 | **387777.40** | 387738.40 | 387172.90 | 387482.25 | 387260.35 | 387011.45 | 387335.20 | 387115.45 | 386981.65 | 387042.25 |
| MDG-a_23 | 389011.70 | **389193.30** | 388927.60 | 388550.65 | 388436.40 | 388526.85 | 388447.90 | 388437.70 | 388266.50 | 388173.50 | 388151.95 |
| MDG-a_24 | 389894.35 | **389798.70** | 389652.95 | 389392.25 | 389286.80 | 389384.25 | 389305.05 | 389672.20 | 389127.95 | 389290.15 | 389247.20 |
| MDG-a_25 | 399195.90 | **399274.85** | 399070.60 | 398747.95 | 398736.70 | 398733.35 | 398607.90 | 398586.40 | 398456.20 | 398533.65 | 398444.50 |
| MDG-a_26 | 402801.90 | **403002.30** | 402898.30 | 402623.35 | 402615.20 | 402519.25 | 402386.45 | 402596.35 | 402459.85 | 402234.10 | 402280.00 |
| MDG-a_27 | **383144.45** | 383090.70 | 383014.00 | 382614.45 | 382703.60 | 382633.20 | 382535.90 | 382532.60 | 382371.60 | 382540.60 | 382267.85 |
| MDG-a_28 | **388703.10** | 388671.70 | 388637.05 | 388058.40 | 388238.45 | 388182.70 | 387864.95 | 388275.35 | 388000.75 | 387985.20 | 387952.00 |
| MDG-a_29 | 384892.50 | **384904.45** | 384683.65 | 384375.10 | 384371.55 | 384431.00 | 384355.30 | 384324.55 | 384221.10 | 384061.50 | 384045.25 |
| MDG-a_30 | 397754.10 | 397771.45 | **397782.80** | 397396.45 | 397385.05 | 397418.65 | 397265.05 | 397223.50 | 397460.85 | 397001.90 | 397037.50 |
| MDG-a_21.40 | 460675.20 | **460815.90** | 460621.80 | 460610.30 | 460068.05 | 460210.95 | 459967.85 | 460363.45 | 460128.60 | 460120.80 | 460216.90 |
| MDG-a_22.40 | 457043.00 | 457017.65 | **457114.85** | 456616.60 | 456628.65 | 456581.00 | 456529.10 | 456592.20 | 456315.80 | 456284.35 | 456369.40 |
| MDG-a_23.40 | 458172.35 | **458232.55** | 458015.00 | 457775.00 | 457551.35 | 457608.50 | 457646.30 | 457566.30 | 457507.30 | 457381.20 | 457329.50 |
| MDG-a_24.40 | **459430.95** | 459280.60 | 459387.65 | 459069.70 | 459079.35 | 458993.40 | 458777.10 | 459156.90 | 458783.55 | 458689.90 | 458612.10 |
| MDG-a_25.40 | **470116.80** | 470106.30 | 469779.70 | 469572.55 | 469536.25 | 469575.85 | 469255.90 | 469473.80 | 469445.25 | 469172.65 | 469174.25 |
| MDG-a_26.40 | 474544.70 | **474584.75** | 474268.20 | 474076.55 | 474033.15 | 474205.25 | 473871.40 | 473962.50 | 473877.00 | 473953.10 | 473769.90 |
| MDG-a_27.40 | **451695.65** | 451694.40 | 451648.40 | 451118.85 | 451344.70 | 451275.10 | 451099.30 | 451331.55 | 451039.80 | 450926.20 | 451049.30 |
| MDG-a_28.40 | **457849.15** | 457818.20 | 457661.10 | 457496.50 | 457377.05 | 457392.00 | 457167.30 | 457417.00 | 457383.10 | 457227.55 | 457155.50 |
| MDG-a_29.40 | **453639.05** | 453485.70 | 453410.50 | 453133.50 | 452892.45 | 453107.40 | 452880.20 | 452982.95 | 452754.20 | 452824.70 | 452756.60 |
| MDG-a_30.40 | 468319.80 | **468379.90** | 468285.80 | 467917.50 | 467989.25 | 467788.70 | 467777.20 | 467887.55 | 467802.45 | 467626.05 | 467617.65 |
| Avg. | 680437.16 | 680610.23 | **680696.55** | 680401.47 | 680487.54 | 680492.24 | 680406.56 | 680531.96 | 680415.44 | 680317.01 | 680262.82 |
| #best | 7 | **13** | 7 | 2 | 0 | 1 | 2 | 5 | 0 | 2 | 1 |

to the setting of parameter $Q$ and the impact of the setting of parameter $Q$ depends on the instances to be solved. Specifically, for large instances with $N = 2000$, small $Q$ values ($\leq 0.2$) led to better results. Moreover, for smaller instances with $N \leq 960$, the effectiveness of $Q$ varied largely according to the instances. For example, for RanReal480_1, $Q = 0.2$ produced the best result in terms of $f_{avg}$, while for another instance named RanReal480_2, $Q = 0.5$ is the best setting. Finally, one observes from Table 6 that $Q = 0.2$ led to the best result in terms of "Avg.", and thus this setting was used as the default value of $Q$ in the present work.

### *5.1.2 Sensitivity Analysis of the Parameter $\alpha$*

To enhance its diversification ability, the NDVNS algorithm employs the parameter $\alpha$ to determine whether a newly generated solution should be accepted as the current solution (line 8 of Algorithm 1), where a larger value of $\alpha$ means that the algorithm emphasizes more on the distance of the offspring solution from the current solution and the best solution found so far. To show the impact of this parameter, we carried out an additional experiment based on 10 representative instances. We ran the algorithm with $\alpha$ in the range of $\{0.002, 0.004, 0.006, 0.008, 0.01, 0.02, 0.03, 0.04\}$ to solve each instance 20 times. The computational results are summarized in Table 7. The first column and the first row give respectively the names of instances and the settings of parameter $\alpha$. The average objective values ($f_{avg}$) over 20 runs are reported in columns 2–9 respectively for each $\alpha$ value. The last row "#Best" shows the number of instances for which the corresponding $\alpha$ value led to the best result in terms of $f_{avg}$ among all the tested $\alpha$ values.

Table 7 shows that the performance of the algorithm depends on the setting of $\alpha$. A too small or too large value of $\alpha$ deteriorates the performance. Specifically, for $\alpha = 0.002$ that is the smallest value tested, the algorithm performed the worst. Moreover, for $\alpha = 0.04$ that is the largest value tested, the algorithm failed to obtain the best result in terms of $f_{avg}$ for any instance. On the other hand, one observes that the setting of $\alpha = 0.01$ led to the best result for 4 out of 10 instances, which implies the best performance of the algorithm. Hence, the default value of $\alpha$ was set to 0.01 in this study.

## *5.2 Effectiveness of the Neighborhood Decomposition Strategy*

The neighborhood decomposition technique described in Section 3.2.2 is the most essential component of the NDVNS algorithm. In order to analyze its effectiveness, we carried out an additional experiment to compare the NDVNS algorithm and a NDVNS variant denoted by NDVNS-D, which was created

Table 7
Influence of parameter $\alpha$ on the average objective values ($f_{avg}$). The NDVNS algorithm was run 20 times for each instance and each $\alpha$ value, and the best results are indicated in bold among the tested parameter settings.

| Instance/$\alpha$ | 0.002 | 0.004 | 0.006 | 0.008 | 0.01 | 0.02 | 0.03 | 0.04 |
|---|---|---|---|---|---|---|---|---|
| RanReal480_01 | 551972.05 | 553334.08 | 554664.70 | 554943.00 | **555519.77** | 555222.15 | 554532.40 | 555212.73 |
| RanReal480_02 | 506587.35 | 507526.65 | 509512.47 | 509651.18 | 509766.19 | **509771.40** | 509395.45 | 509391.92 |
| RanReal480_03 | 492032.45 | 493338.74 | 495741.40 | **496315.30** | 495276.96 | 495539.21 | 495714.79 | 495542.26 |
| RanReal480_04 | 517084.98 | 518199.11 | 520179.98 | 520144.21 | **520456.70** | 519745.04 | 520140.74 | 520306.09 |
| RanReal480_05 | 479475.83 | 481015.04 | 482583.06 | **482837.64** | 482337.86 | 482508.11 | 481984.18 | 482389.91 |
| MDG-a_21 | 388733.50 | 390682.95 | 390544.65 | **390762**.65 | 390630.90 | 390746.05 | 390672.10 | 390611.90 |
| MDG-a_22 | 385473.20 | 387601.00 | **387741.65** | 387631.65 | 387583.40 | 387500.95 | 387561.90 | 387676.05 |
| MDG-a_23 | 386416.55 | 388803.75 | 388831.65 | 388880.25 | **388969.10** | 388788.70 | 388827.70 | 388735.15 |
| MDG-a_24 | 387404.90 | 389537.45 | **389855.90** | 389800.25 | 389804.70 | 389715.60 | 389719.45 | 389697.50 |
| MDG-a_25 | 396814.05 | 399019.30 | 399135.10 | 399106.85 | **399150.95** | 398981.35 | 399115.50 | 399033.00 |
| #Best | 0 | 0 | 2 | 3 | **4** | 1 | 0 | 0 |

by disabling the neighborhood decomposition strategy of NDVNS (i.e., all the entries of state matrices $M_1$, $M_2$, and $M_3$ are always set to the value of 1 during the search process). In this experiment, we ran both algorithms 20 times for each of 40 representative instances, and the results are summarized in Table 8. Column 1 of the table gives the names of instances, columns 2–3 report the best objective values ($f_{best}$) over 20 runs for each algorithm, columns 4–5 indicate the average objective value ($f_{avg}$), columns 6–7 give the worst objective value ($f_{worst}$), and the last two columns show the standard deviation ($\sigma$) of the objective values obtained. The rows "#better", "#equal", and "#worse" indicate the number of instances for which the corresponding algorithm obtained a better, equal, and worse result compared to the other algorithm. In addition, the $p$-values from the non-parametric Friedman tests are provided for each performance indicator in the last row of the table.

Table 8 shows that NDVNS dominates the NDVNS-D variant in terms of $f_{best}$, $f_{avg}$, and $f_{worst}$. Specifically, for each tested instance, NDVNS obtained a better result than NDVNS-D in terms of $f_{best}$, $f_{avg}$, and $f_{worst}$. Moreover, the standard deviations ($\sigma$) are smaller with NDVNS than with NDVNS-D on most instances. The statistical differences of the compared algorithms are confirmed by small *p-values* ($< 0.05$). This experiment demonstrates the effectiveness of the neighborhood decomposition strategy of the NDVNS algorithm.

## *5.3 Spatial distribution of high-quality solutions*

To understand intuitively the spatial distribution of high-quality local optimum solutions in the search space and the rationality of the underlying strategies of the proposed algorithm, we conducted an additional computational experiment to show a rough picture as to how high-quality solutions could be distributed in the search space. In this experiment, the NDVNS al-

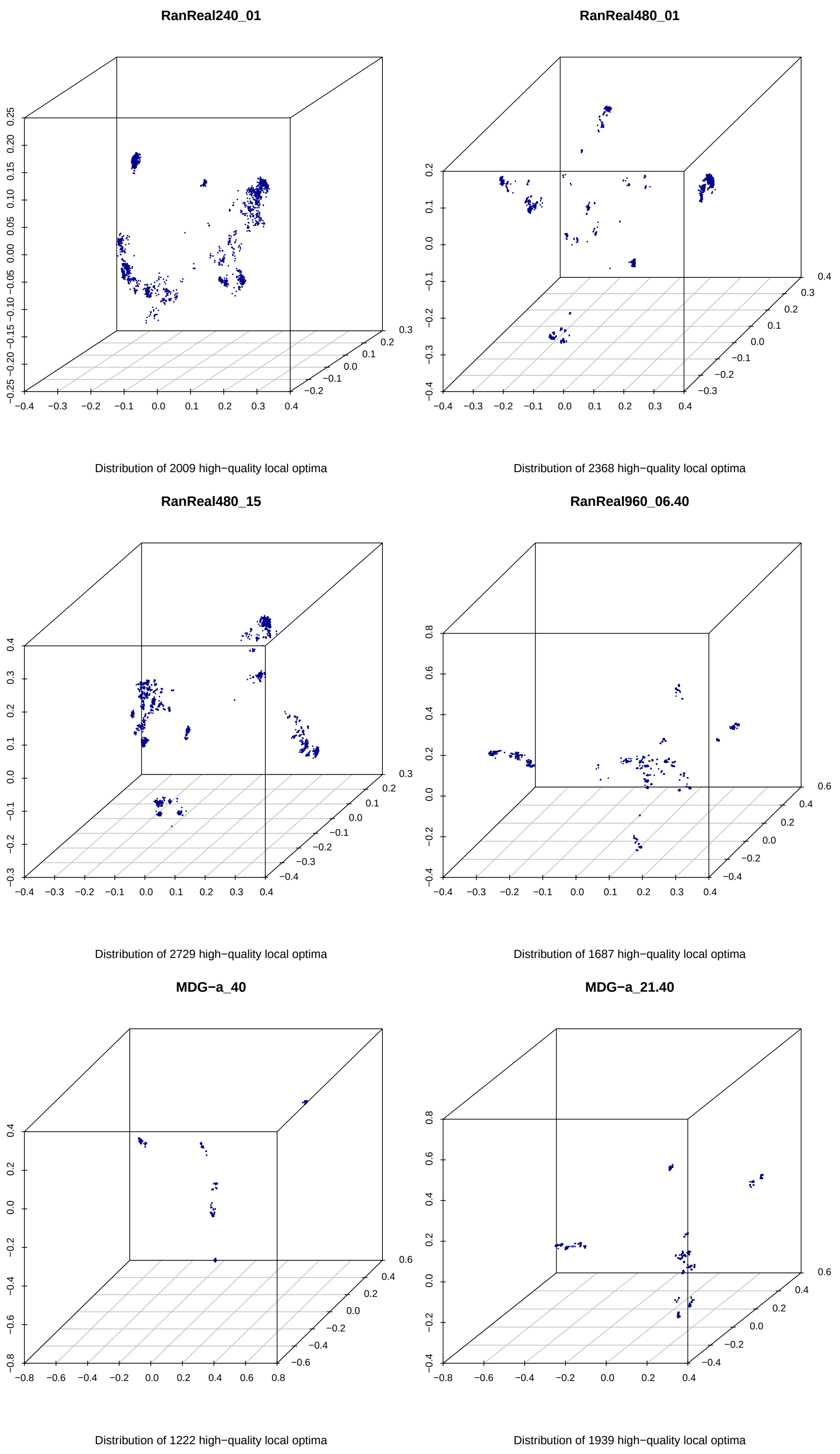


Fig. 4. Spatial distribution of high-quality local optimum solutions for six representative instances.

Table 8
Comparison between the proposed NDVNS algorithm and its variant denoted by the NDVNS-D algorithm in which the neighborhood neighborhood strategy is disabled. Dominating values between the two compared algorithms are indicated in bold for each instance and each performance indicator.

| | $f_{best}$ | | $f_{avg}$ | | $f_{worst}$ | | $\sigma$ | |
|---|---|---|---|---|---|---|---|---|
| Instance | NDVNS | NDVNS-D | NDVNS | NDVNS-D | NDVNS | NDVNS-D | NDVNS | NDVNS-D |
| RanReal480_01 | **556477.17** | 556280.14 | **555549.33** | 555188.88 | **552936.00** | 549586.97 | **823.69** | 1441.65 |
| RanReal480_02 | **511381.67** | 511156.11 | **509625.83** | 508838.48 | **505101.17** | 504309.24 | **1504.78** | 1953.92 |
| RanReal480_03 | **497586.82** | 497492.28 | **495715.20** | 495595.49 | **493672.18** | 493368.03 | **1449.09** | 1499.75 |
| RanReal480_04 | **523258.76** | 522697.56 | **520445.75** | 520276.45 | **517178.17** | 516555.37 | 1743.75 | **1505.43** |
| RanReal480_05 | **484541.10** | 484217.10 | **482670.41** | 482420.78 | **480870.94** | 479795.19 | **1129.91** | 1258.78 |
| RanReal960_01.30 | **1340580.00** | 1339721.61 | **1338735.21** | 1337128.18 | **1335372.26** | 1333572.21 | **1079.28** | 1327.26 |
| RanReal960_02.30 | **1435894.10** | 1432847.02 | **1432816.25** | 1431412.37 | **1431006.80** | 1429735.56 | 1258.08 | **866.59** |
| RanReal960_03.30 | **1399371.79** | 1396425.01 | **1396676.76** | 1394971.21 | **1392729.52** | 1392159.71 | 1560.16 | **1252.32** |
| RanReal960_04.30 | **1414472.27** | 1413280.79 | **1412766.74** | 1410858.71 | **1410140.56** | 1406130.61 | **1106.39** | 1749.05 |
| RanReal960_05.30 | **1372173.99** | 1370546.90 | **1370903.70** | 1368766.98 | **1369593.86** | 1366537.32 | **752.46** | 1222.22 |
| RanReal960_01.40 | **1042160.11** | 1040891.98 | **1040859.48** | 1038960.64 | **1039445.10** | 1035496.19 | **729.39** | 1441.51 |
| RanReal960_02.40 | **1117362.73** | 1115627.50 | **1115766.18** | 1113797.06 | **1113129.71** | 1110776.04 | **1102.45** | 1288.23 |
| RanReal960_03.40 | **1088482.73** | 1087850.94 | **1087361.36** | 1085841.46 | **1084553.32** | 1084119.65 | 857.30 | **824.45** |
| RanReal960_04.40 | **1102093.11** | 1100375.06 | **1100356.79** | 1098719.99 | **1098057.48** | 1096684.10 | **1053.52** | 1105.54 |
| RanReal960_05.40 | **1066512.35** | 1064464.01 | **1064946.28** | 1062779.69 | **1063548.18** | 1061008.97 | **901.98** | 999.41 |
| RanReal960_01.60 | **733456.08** | 731350.74 | **731642.79** | 729716.28 | **730397.52** | 728080.80 | **833.78** | 837.64 |
| RanReal960_02.60 | **776818.81** | 775654.32 | **774895.27** | 772275.93 | **772821.57** | 769968.23 | **907.74** | 1182.85 |
| RanReal960_03.60 | **760549.17** | 759597.79 | **758954.36** | 757088.50 | **757406.97** | 755836.32 | 858.37 | **850.66** |
| RanReal960_04.60 | **769889.49** | 767863.97 | **768146.77** | 766096.19 | **766488.28** | 764262.17 | 916.02 | **889.35** |
| RanReal960_05.60 | **749585.55** | 747151.07 | **747026.39** | 745440.24 | **745099.96** | 742728.88 | **1017.47** | 1078.15 |
| MDG-a_21 | **391375.00** | 390547.00 | **390724.55** | 389740.95 | **389947.00** | 388851.00 | **362.89** | 396.02 |
| MDG-a_22 | **388315.00** | 387383.00 | **387513.50** | 386596.25 | **387003.00** | 385926.00 | **346.58** | 406.09 |
| MDG-a_23 | **389437.00** | 388321.00 | **388879.50** | 387909.05 | **388218.00** | 387210.00 | 320.46 | **317.16** |
| MDG-a_24 | **390204.00** | 389533.00 | **389726.40** | 388761.40 | **389141.00** | 387878.00 | **292.65** | 409.48 |
| MDG-a_25 | **399482.00** | 398824.00 | **398931.90** | 398227.00 | **398315.00** | 397773.00 | **318.32** | 329.20 |
| MDG-a_26 | **403151.00** | 402656.00 | **402653.30** | 402100.95 | **401907.00** | 401471.00 | 316.88 | **294.16** |
| MDG-a_27 | **383599.00** | 382724.00 | **382984.05** | 381960.65 | **382307.00** | 381385.00 | **283.87** | 321.03 |
| MDG-a_28 | **388917.00** | 388106.00 | **388480.90** | 387551.10 | **387746.00** | 386965.00 | **309.98** | 325.49 |
| MDG-a_29 | **385599.00** | 384607.00 | **384860.65** | 383847.75 | **384148.00** | 383189.00 | **384.72** | 391.42 |
| MDG-a_30 | **398146.00** | 397380.00 | **397687.00** | 396764.20 | **397191.00** | 395634.00 | **294.11** | 363.92 |
| MDG-a_21.40 | **461467.00** | 460399.00 | **460667.70** | 459906.10 | **459840.00** | 458802.00 | **358.25** | 405.02 |
| MDG-a_22.40 | **457376.00** | 456526.00 | **456782.20** | 456084.20 | **455514.00** | 455358.00 | 416.19 | **315.13** |
| MDG-a_23.40 | **458711.00** | 458054.00 | **457912.05** | 456990.45 | **457138.00** | 455904.00 | **364.49** | 462.72 |
| MDG-a_24.40 | **459778.00** | 458703.00 | **459164.10** | 458127.35 | **458404.00** | 457108.00 | **331.25** | 448.82 |
| MDG-a_25.40 | **470605.00** | 469873.00 | **469607.40** | 469119.20 | **468781.00** | 468034.00 | **357.85** | 424.64 |
| MDG-a_26.40 | **475036.00** | 474078.00 | **474313.50** | 473349.90 | **473408.00** | 471753.00 | **415.44** | 575.59 |
| MDG-a_27.40 | **451983.00** | 451363.00 | **451253.05** | 450589.65 | **450777.00** | 449563.00 | **334.28** | 450.01 |
| MDG-a_28.40 | **458496.00** | 457561.00 | **457585.55** | 456946.50 | **456221.00** | 456069.00 | 482.50 | **322.54** |
| MDG-a_29.40 | **453877.00** | 453216.00 | **453416.60** | 452515.05 | **452757.00** | 451725.00 | **334.95** | 425.75 |
| MDG-a_30.40 | **468597.00** | 468312.00 | **468128.40** | 467430.10 | **467418.00** | 466729.00 | **368.32** | 434.44 |
| Avg. | **681919.97** | 680841.45 | **680678.33** | 679517.28 | **679143.26** | 677700.96 | **714.49** | 809.83 |
| #Better | 40 | 0 | 40 | 0 | 40 | 0 | 30 | 10 |
| #Equal | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| #Worse | 0 | 40 | 0 | 40 | 0 | 40 | 10 | 30 |
| *p-value* | 2.54E-10 | | 2.54E-10 | | 2.54E-10 | | 2.04E-3 | |

gorithm was performed 20 times for each instance and all the high-quality local optimum solutions returned by the local search method $\text{NDVND}_2$ were collected respectively for each instance, where a solution $s$ is considered to be of high-quality if its objective value $f(s)$ is superior to the average objective value $f_{avg}$ reported in Section 4, i.e., $f(s) > f_{avg}$.

Following [23,34], we visualize the spatial distribution of the collected high-quality solutions in Euclidean space $R^3$ by using a multidimensional scaling (MDS) procedure as follows. First, we generate a distance matrix $D_{l\times l}$ between solutions in the original search space $\Omega$, where $l$ is the number of the collected solutions and the partition distances $d_{ij} \in D_{l\times l}$ between solutions are calculated. Then, from the distance matrix $D_{l\times l}$, the $l$ coordinate points are generated in Euclidean space $R^3$ by using the classic *cmdscale* method whose goal is to minimize the distance distortion caused between the original space and Euclidean space. Finally, the scatter graph of these $l$ coordinate points is plotted in $R^3$.

Fig. 4 shows the scatter graphs for six selected instances. One clearly observes that high-quality local optimum solutions attained by the proposed algorithm are grouped in clusters in the search space. This finding implies that when the search process reaches a high-quality solution, it is very useful to make a sufficient exploitation around this solution by performing limited perturbations and subsequent local searches. On the other hand, when a search region has been sufficiently exploited and the search process is trapped into a deep local optimum, it is necessary to jump out of the trap by performing some large perturbation operations. In the NDVNS algorithm, this is jointly achieved by the intensification-oriented NDVND$_2$ procedure and the probabilistic perturbation of the shake procedure.

## 6 Conclusion and Future work

The neighborhood decomposition-driven variable neighborhood search algorithm proposed in this work for solving the capacitated clustering problem integrates the ideas of neighborhood decomposition and probability-based perturbation. By isolating promising candidate solutions to be considered at each search iteration, neighborhood decomposition constitutes a powerful technique to speed up neighborhood examination and enable more focused searches.

We have performed extensive experiments on 110 instances commonly used in the literature to show the competitiveness of the proposed algorithm. The computational results demonstrate that the algorithm performs extremely well on the test suite compared to the state-of-the-art methods in the literature. The advantage of the algorithm is even more evident in terms of computational efficiency and search effectiveness when large-scale problem instances are considered. This work thus advances the state-of-the-art of solving the challenging capacitated clustering problem. Moreover, given that the considered problem is a general model to formulate a variety of practical applications, the publicly available code of the algorithm can help to solve these practical problems.

The ideas of neighborhood decomposition-driven local search and probability-based perturbation are of general nature. Thus, it would be interesting to check the usefulness of these ideas on other clustering or grouping problems such as those mentioned in the introduction.

## Acknowledgments

We are grateful to the reviewers for their valuable comments and suggestions which helped us to improve the paper. This work was partially supported by the six talent peaks project in Jiangsu Province of China (Grant No. RJFW-011), the National Natural Science Foundation of China (Grant No. 61703213), the Natural Science Foundation of Jiangsu Province of China (Grant No. BK20170904), and Shenzhen Science and Technology Innovation Commission under Grant Nos. JCYJ20180508162601910 and 2019-INT003.

## References

[1] U. Benlic and J.K. Hao, A multilevel memetic approach for improving graph k-partitions, *IEEE Transactions on Evolutionary Computation*, 15(5): 624–642, 2011.

[2] J.L. Bentley, Fast algorithms for geometric traveling salesman problems, *ORSA Journal on Computing*, 4(4): 387–411, 1992.

[3] J. Brimberg, N. Mladenović, and D. Urošević, Solving the maximally diverse grouping problem by skewed general variable neighborhood search, *Information Sciences*, 295: 650–675, 2015.

[4] J. Brimberg, N. Mladenović, R. Todosijević, and D. Urošević, Solving the capacitated clustering problem with variable neighborhood search, *Annals of Operations Research*, 272(1): 289–321, 2019.

[5] J. Chang, L. Wang, J.K. Hao, and Y. Wang, Parallel iterative solution-based tabu search for the obnoxious $p$-median problem. *Computers & Operations Research*, 127: 105155, March 2021.

[6] Y. Chen, Z.P. Fan, J. Ma, and S. Zeng, A hybrid grouping genetic algorithm for reviewier group construction problem. *Expert Systems with Applications*, 38(3): 2401–2411, 2011.

[7] R.L. Church and S. Wang, Solving the $p$-median problem on regular and lattice networks, *Computers & Operations Research*, 123: 105057, November 2020.

[8] C. Contardo, M. Iori, and R. Kramer, A scalable exact algorithm for the vertex $p$-center problem, *Computers & Operations Research*, 103: 211–220, 2019.

[9] Y.M. Deng and J.F. Bard, A reactive GRASP with path relinking for capacitated clustering, *Journal of Heuristics*, 17(2): 119–152, 2011.

[10] C.N. Fiechter, A parallel tabu search algorithm for large traveling salesman problems, *Discrete Applied Mathematics*, 51: 243-267, 1994.

[11] C.E. Ferreira, A. Martin, C.C. de Souza, R. Weismantel, and L.A. Wolsey, The node capacitated graph partitioning problem: A computational study, *Mathematical Programming*, 81: 229–256, 1998.

[12] M. Gallego, M. Laguna, R. Martí, and A. Duarte, Tabu search with strategic oscillation for the maximally diverse grouping problem, *Journal of the Operational Research Society*, 64: 724–734, 2013.

[13] P. Galinier, Z. Boujbel, and M.C. Fernandes, An efficient memetic algorithm for the graph partitioning problem, *Annals of Operations Research*, 191(1): 1–22, 2011.

[14] F. Glover and M. Laguna, Tabu search. Springer, Kluwer Academic Publishers, Boston, MA, 1997.

[15] M. Gnägi and P. Baumann, A matheuristic for large-scale capacitated clustering, *Computers & Operations Research*, 132: 105304, August 2021.

[16] G. González-Almagro, J. Luengo, J. Cano and S. García, DILS: Constrained clustering through dual iterative local search, *Computers & Operations Research*, 121: 104979, September 2020.

[17] D. Gusfield, Partition-distance: A problem and class of perfect graphs arising in clustering, *Information Processing Letters*, 82(3): 159–164, 2002.

[18] P. Hansen and N. Mladenović, Variable neighborhood search: Principles and applications, *European Journal of Operational Research*, 130(3): 449–467, 2001.

[19] B. Hendrickson and T.G. Kolda, Graph partitioning models for parallel computing, *Parallel Computing*, 26(12):1519–1534, 2010.

[20] H. H. Hoos and T. Stützle, Stochastic local search: foundations and applications, The Morgan Kaufmann Series in Artificial Intelligence, Morgan Kaufmann, 2004

[21] M. Laguna, J.W. Barnes and F. Glover, Tahu search methods for a single machine scheduling problem. *Journal of Intelligent Manufacturing*, 2: 63-74, 1991.

[22] X.J. Lai and J.K. Hao, Iterated variable neighborhood search for the capacitated clustering problem, *Engineering Applications of Artificial Intelligence*, 56: 102–120, 2016.

[23] X.J. Lai and J.K. Hao, Iterated maxima search for the maximally diverse grouping problem, *European Journal of Operational Research*, 254(3): 780–800, 2016.

[24] X.J. Lai, J.K. Hao, Z.H. Fu, D. Yue, Neighborhood decomposition based variable neighborhood search and tabu search for maximally diverse grouping, *European Journal of Operational Research*, 289: 1067–1086, 2021.

[25] M. Lewis, H.B. Wang, and G. Kochenberger, Exact solutions to the capacitated clustering problem: A comparison of two models, *Annals of Data Science*, 1(1): 15–23, 2014.

[26] Z. Lu, J.K. Hao, and Y. Zhou, Stagnation-aware breakout tabu search for the minimum conductance graph partitioning problem. *Computers & Operations Research*, 111: 43–57, 2019.

[27] J. M. Mulvey and M. P. Beck, Solving capacitated clustering problems, *European Journal of Operational Research*, 18(3), 339–348, 1984.

[28] A. Martínez-Gavara, V. Campos, M. Gallego, M. Laguna, and R. Martí, Tabu search and GRASP for the capacitated clustering problem. *Computational Optimization and Applications*, 62(2): 589–607, 2015.

[29] A. Martínez-Gavara, D. Landa-Silva, V. Campos, and R. Martí, Randomized heuristics for the capacitated clustering problem, *Information Sciences*, 417: 154–168, 2017.

[30] H. Meyerhenke, P. Sanders, and C. Schulz, Parallel graph partitioning for complex networks, *IEEE Transactions on Parallel and Distributed Systems*, 28(9): 2625–2638, 2017.

[31] N. Mladenović and P. Hansen, Variable neighborhood search, *Computers & Operations Research*, 24(1): 1097–1100, 1997.

[32] L. F. Morán-Mirabal, J. L., González-Velarde, M. G. C. Resende, R. M. A. Silva, Randomized heuristics for handover minimization in mobility networks. *Journal of Heuristics*, 19:845–880, 2013.

[33] G. Palubeckis, A. Ostreika, and D. Rubliauskas, Maximally diverse grouping: an iterated tabu search approach, *Journal of the Operational Research Society*, 65(6): 1–14, 2014.

[34] D.C. Porumbel, J.K. Hao, and P. Kuntz, A search space cartography for guiding graph coloring heuristics. *Computers & Operations Research* 37(4): 769–778, 2010.

[35] D.C. Porumbel, J.K. Hao, and P. Kuntz, An efficient algorithm for computing the distance between close partitions, *Discrete Applied Mathematics*, 159(1): 53–59, 2011.

[36] J. Puerto, M. Rodríguez-Madrena, and A. Scozzari, Clustering and portfolio selection problems: A unified framework, *Computers & Operations Research*, 117: 104891, May 2020.

[37] M.G.C. Resende and C.C. Ribeiro, Greedy randomized adaptive search procedures, In F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics, p. 219–249, Kluwer Academic Publishers, 2003.

[38] F. Semet and E. Taillard, Solving real-life vehicle routing problems efficiently using tabu search, *Annals of Operations Research*, 41: 469-488, 1993.

[39] T. Stützle, Iterated local search for the quadratic assignment problem, *European Journal of Operational Research*, 174: 1519–1539, 2006.

[40] D.H. Tran, B. Babaki, D.V. Daele, P. Leyman, and P. De Causmaecker, Local search for constrained graph clustering in biological networks, *Computers & Operations Research*, 132: 105299, August 2021.

[41] R. Weitz and S. Lakshminarayan, An empirical comparison of heuristic and graph theoretic methods for creating maximally diverse groups, VLSI design, and exam scheduling, *Omega*, 25(4): 473–482, 1997.

[42] Q. Zhou, U. Benlic, Q.H. Wu, and J.K. Hao, Heuristic search to the capacitated clustering problem, *European Journal of Operational Research*, 273(2): 464–487, 2019.

[43] Y. Zhou, J.K. Hao, and A. Goöffon, A three-phased local search approach for the clique partitioning problem. *Journal of Combinatorial Optimization*, 32(2): 469–491, 2016.