Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 8839763, 16 pages
https://doi.org/10.1155/2020/8839763

Research Article

K-Means Genetic Algorithms with Greedy Genetic Operators

Lev Kazakovtsev, Ivan Rozhnov, Guzel Shkaberina, and Viktor Orlov

Reshetnev Siberian State University of Science and Technology, Prosp.Krasnoysrskiy Rabochiy 31, Krasnoyarsk 660037, Russia

Correspondence should be addressed to Lev Kazakovtsev; levk@bk.ru

Received 29 August 2020; Accepted 11 November 2020; Published 30 November 2020

Academic Editor: Piotr Jdrzejowicz

Copyright © 2020 Lev Kazakovtsev et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The *k*-means problem is one of the most popular models of cluster analysis. The problem is NP-hard, and modern literature offers many competing heuristic approaches. Sometimes practical problems require obtaining such a result (albeit notExact), within the framework of the *k*-means model, which would be difficult to improve by known methods without a significant increase in the computation time or computational resources. In such cases, genetic algorithms with greedy agglomerative heuristic crossover operator might be a good choice. However, their computational complexity makes it difficult to use them for large-scale problems. The crossover operator which includes the *k*-means procedure, taking the absolute majority of the computation time, is essential for such algorithms, and other genetic operators such as mutation are usually eliminated or simplified. The importance of maintaining the population diversity, in particular, with the use of a mutation operator, is more significant with an increase in the data volume and available computing resources such as graphical processing units (GPUs). In this article, we propose a new greedy heuristic mutation operator for such algorithms and investigate the influence of new and well-known mutation operators on the objective function value achieved by the genetic algorithms for large-scale *k*-means problems. Our computational experiments demonstrate the ability of the new mutation operator, as well as the mechanism for organizing subpopulations, to improve the result of the algorithm.

1. Introduction

The *k*-means problem is a continuous unconstrained global optimization problem which has become a classic clustering model. This problem is proved to be NP-hard [1, 2], so it is necessary to find a compromise between the computation time and the solution preciseness. The aim of the problem is to find set $S = \{X_1, \ldots, X_k\}$ of k points and $X_1, \ldots, X_k \in \mathbb{R}^d$ called centroids in a d-dimensional space that minimizes the sum of squared distances from N known points (data vectors) $A_1, \ldots, A_N \in \mathbb{R}^d$ to the nearest centroid [3]:

$$F(X_1, \ldots, X_k) = F(S) = \sum_{i=1}^{N} \min_{X \in \{X_1, \ldots, X_k\}} (L(A_i, X))^2 \longrightarrow \min_{X_1, \ldots X_k \in \mathbb{R}^d}, \quad (1)$$

where $L(\cdot)$ is the distance between two points (usuallyEuclidean) and k is given.

Data vector indexes for which the jth centroid is the nearest one form a set (cluster) C_j, $j = \overline{1,k}$. An equivalent problem setting is as follows:

$$F(X_1, \ldots, X_k) = \sum_{j=1}^{k} \sum_{i \in C_j} \left(L\left(A_i, X_j\right)\right)^2 \longrightarrow \min, \quad (2)$$

where X_j is the centroid of the jth cluster.

The simplest and most popular local search algorithm is the *k*-means algorithm [4, 5] also called Alternate Location and Allocation (ALA) procedure [6, 7] or Lloyd algorithm. Similar procedure called EM (Expectation Maximization) [8, 9] and its modifications [10–12] are the most popular algorithms for separating the mix probability distribution. The *k*-means algorithm improves an intermediate solution sequentially, which enables us to find a local minimum.

Technically, this is not a true local search algorithm in terms of continuous optimization, as it searches for a new solution not necessarily in the ε-neighborhood of an existing

solution. Nevertheless, it enables us a solution which is locally optimal in ε-neighborhood.

If we use distances instead of squared distances in (1), we deal with the continuous p-median problem. The similarity of these NP-hard problems [13, 14] allows us to use similar approaches to solving them. However, unlike the p-median problem with Euclidean distances, finding the exact solution of a 1-means problem (k-means problem with $k = 1$ or the centroid search problem) in accordance with Algorithm 1 is trivial, and finding a local minimum of the k-means problem takes less computational resources. This allows the local search to be integrated into various effective global search strategies.

In the early attempts to solve the p-median problem by exact methods (its discrete modifications), the authors used a branch and bound algorithm [15–17] for solving very small problems. In [18–20], the authors reviewed various heuristic solution techniques for k-means and p-median problems. In [21–23], the authors presented local search approaches including the Variable Neighborhood Search (VNS) and concentric search. In [22], Drezner et al. proposed heuristic procedures including the genetic algorithm (GA), for rather small datasets.

Many approaches, based on the data reducing [24], simplify the problem by selection of some part of the initial dataset and then use these results as an initial solution to the k-means algorithm on the complete dataset [25–28]. Such aggregating as well as reducing the number of the data vectors [29] enables us to solve large-scale problems within a reasonable time. However, such approaches lead to a reduction in preciseness. In our research aimed at obtaining the most precise solutions, we consider only the methods which estimate the objective function (1) directly, without aggregation or approximation approaches.

Modern publications offer many heuristic procedures [19, 30] for setting the initial centroids for the k-means algorithm, most of them belong to various evolutionary and random search methods. Local search algorithms and their randomized versions are widely presented. For instance, Variable Neighborhood Search (VNS) algorithms [23, 31, 32] or agglomerative algorithms [33, 34] sometimes show good results. A large number of articles are devoted to the initialization procedures for local search algorithms, such as random seeding and estimating the distribution of the data vectors [30]. The challenge is that, in many cases, even multiple runs of simple local search algorithms from various randomly generated solutions do not lead to a solution that is close to the global optimum. More advanced algorithms enable us to get the objective function (1) values many times better than the local search methods [32].

The use of genetic algorithms and other evolutionary approaches to improve the results of the local search is a widely used idea [35–38]. Such algorithms recombine the local minima obtained by the k-means algorithm. GAs operate with a certain set (population) of candidate solutions and include special genetic operators (algorithms) of initialization, selection, crossover, and mutation. The mutation operator randomly changes the resulting solutions and provides some diversity in the population.

However, in genetic algorithms, as the number of iterations increases, the population degenerates into a certain set of solutions close to each other. Larger populations as well as dynamically growing populations improve this situation. However, simpler algorithms based on the use of the same greedy agglomerative procedures [32, 39] often show better results within the same computation time.

In this research, we do not discuss the adequacy of the k-means clustering model, which is actually questionable. We only focus on the preciseness and stability of the obtained objective function value (1) within the framework of the k-means model.

There are situations when the cost of error is high [9]. In these cases, as well as when comparing the accuracy of an algorithm with a certain standard solution (not necessarily globally optimal), we need to get a result that would be difficult to enhance by other known methods without meaningful increase of the computation time. Evolution of parallel processing systems such as graphics processing units (GPUs) makes multiple runs of local search algorithms very cheap. In this case, large-scale problems (up to several millions of data vectors) can be solved with the use of the most advanced algorithms providing the highest preciseness. As our study shows, for large-scale problems, further improvement in the results of the genetic algorithms with greedy heuristic crossover can be achieved by using a special mutation operator and partially isolated solution subpopulations.

The aim of this paper is to introduce a new k-Means Genetic Algorithm with the greedy agglomerative crossover operator, a special greedy agglomerative mutation operator, and subpopulations.

The rest of this article is organized as follows. In Section 2, we propose a brief overview of known approaches to the development of k-means genetic algorithms. In Section 3, we give an overview of known mutation genetic operators used in k-median genetic algorithms in accordance with various approaches to chromosome encoding as well as other instruments for increasing the population diversity. In Section 4, we propose new modifications to the genetic algorithms with greedy heuristic crossover operator. Such modifications include partially isolated subpopulations and the use of a new mutation operator based on the greedy heuristic procedure. In Section 5, we describe the results of our computational experiments which demonstrate the efficiency of our new modifications on large datasets.

2. *K*-Means Genetic Algorithms

The idea of various genetic algorithms is based on a recombination (interchange) of elements in a set ("population") of candidate solutions ("individuals") encoded by "chromosomes." Such elements of the chromosomes are called "genes" or "alleles." Each chromosome is a vector of genes (bits, integers, or real numbers) representing a solution. The goal of gene recombination is achieving the best value of an objective function called "fitness function." The appearance of the first genetic algorithms for solving the discrete p-median problem [40] preceded the genetic algorithms for the k-means problem (k-

Require: Set of k initial centroids $X_1, \ldots, X_k$.
(1) ForEach centroid X_i, $i = \overline{1,k}$, define its cluster $C_i \subset \{A_1, \ldots, A_N\}$ as the subset of data vectors having closest centroid X_i.
(2) ForEach cluster C_i, $i = \overline{1,k}$, recalculate its centroid as follows: $X_i = \sum_{j \in C_i} A_j \cdot |C_i|^{-1}$.
(3) Repeat from Step 1 if Steps 1, 2 made any changes.

Algorithm 1: *kMeans*().

Means Genetic Algorithms). Alp et al. [41] proposed a rather fast and precise algorithm with a special "greedy" (agglomerative) heuristic procedure used as the crossover genetic operator for the network p-median problem. Such algorithms solve discrete network problems and use a very simple binary chromosome encoding (1 for the network nodes selected as the centers of the clusters, and 0 for those not selected).

In the genetic algorithms for the k-means and similar problems with binary-encoded chromosomes, many mutation techniques can be used. For example, in [42], the authors represent the chromosome with binary strings composed from binary-encoded features (coordinates) of the centroids. The mutation operator arbitrarily alters one or more components (binary substrings) of a selected chromosome.

If the centers or centroids are searched for in a continuous space, some genetic algorithms still use the binary encoding [38, 43, 44]. In the k-means algorithm, its initial solutions are usually subsets of the dataset $\{A_1, \ldots, A_N\}$. In such chromosome code, 1 means that the corresponding data vector must be selected as an initial centroid and 0 for those not selected. In this case, some local search algorithm (k-means algorithm or similar) is used at each iteration of the GA to estimate the final value (local minimum) of the objective function (1).

In [45], the authors refer to their algorithm as "Evolutionary k-Means." However, they actually solved an alternative problem which aimed to increase the clustering stability instead of minimizing (1). Their algorithm operates with binary consensus matrices and uses two types of mutation genetic operators: cluster split (dissociative) and cluster merge (agglomerative) mutation. In [46], the chromosomes are strings of integers representing the cluster number for each of the clustered objects, and the authors solve the k-means problem with simultaneous determining the number of clusters based on the silhouette [47] and Davies and Bouldin criteria [48], which are used as the fitness functions. Thus, in [46], the authors also solve a problem with the mathematical statement other than (1). Similar encoding is used in [37] where the authors propose a mutation operator which changes the assignment of individual data vectors to the clusters.

In [49], the authors described the mutation operator as a procedure that guarantees population diversity (variability). Usually, for the k-means and p-median problems, the mutation randomly changes one or many chromosomes, replacing some centroids [36, 37]. Mutation and crossover are the most important genetic operators playing different roles: the crossover seeks to preserve the features of parent solutions, while the mutation tries to cause small local changes in the solutions. Compared to a crossover, a mutation is usually regarded as a secondary operator with a low probability μ [50]. High frequency of mutations makes a genetic algorithm to search randomly and chaotically. Nevertheless, many studies have shown that evolutionary algorithms without a crossover can work better than a standard genetic algorithm if the mutation is combined with an effective selection operator [51–54]. Mutation is performed on a single parent solution.

In [36], the authors encode the solutions (chromosomes) in their GA as sets of centroids represented by their coordinates (real vectors or arrays). The genetic algorithms with the greedy heuristic crossover operator use the same principle [55].

Thus, various genetic algorithms for the k-means and similar problems can be classified into three categories in accordance with the chromosome encoding method:

(a) Integer encoding each gene represents a data vector $A_1, \ldots, A_N$, and the value is its cluster number. Such algorithms are declared for solving the k-means or p-median problem; however, they may use other objective function than (1). A local search for the minimum of (1) is sometimes declared their mutation operator.

(b) Integer or binary encoding each gene corresponds to a centroid (cluster) and describes the data vector index $\overline{1,N}$ selected as the initial centroid for the local search method. Such algorithms may use a wide variety of crossover and mutation operators.

(c) Real (direct) encoding each gene is a centroid encoded by its coordinates. Such algorithms are able to demonstrate the most precise results. However, the modern literature offers a very limited variety of mutation operators for such algorithms. Usually, they do not use any mutation [38, 41, 43].

The greedy heuristic crossover operator can be described as a two step algorithm. The first step combines two known ("parent") solutions (chromosomes) into one intermediate invalid solution with an excessive number of centroids (clusters). At the second step (the greedy agglomerative procedure), the algorithm removes excessive centroids in each iteration so that the removal of the centroid results in the least significant growth of the objective function (1) [41, 43], see Algorithm 2.

Algorithms 3 and 4 are known heuristic procedures [32, 41, 43], which implement the first step of the greedy heuristic crossover operator and run *Greedy* heuristic procedure.

Require: Final number of centroids k, initial solution $S = \{X_1, \ldots, X_K\}$, $K > k$
(1) Improve S with the local search algorithm if possible: $S \longleftarrow kMeans(S)$
(2) **while** $K > k$
(3) **for all** $i' \in \overline{1, K}$
(4) Assign $S' \longleftarrow S \backslash \{X_{i'}\}$
(5) Calculate $F_{i'}' \longleftarrow F(S')$ //where $F(\cdot)$ is the objective function (1)
(6) **end for**
(7) Select a subset S_{elim} of n_{elim} centroids, $S_{\text{elim}} \subset S$, $|S_{\text{elim}}| = n_{\text{elim}}$, with the minimal values of the corresponding variables $F_{i'}'$.Here, $n_{\text{elim}} = \max\{1, 0.2 \cdot (|S| - k)\}$
(8) Obtain the new solution $S \longleftarrow S \backslash S_{\text{elim}}$, $K \longleftarrow |S|$, and improve this new solution with the local search algorithm: $S \longleftarrow kMeans(S)$
(9) **End while**
(10) **return** Solution S

Algorithm 2: *Greedy*(): greedy (agglomerative) heuristic procedure.

Require: Two solutions (sets of centers) $S' = \{X_1', \ldots, X_k'\}$ and $S'' = \{X_1'', \ldots, X_k''\}$
return $Greedy(S' \cup S'')$

Algorithm 3: *Greedy FULL*.

Require: Two solutions (sets of centers) $S' = \{X_1', \ldots, X_k'\}$ and $S'' = \{X_1'', \ldots, X_k''\}$
for all $i' \in \overline{1, p}$
Merge S' and one item of S'': $S \leftarrow S' \cup \{X_{i'}''\}$; $S_{i'} \leftarrow Greedy(S)$
end for
return the best of solutions $S_1, \ldots, S_p$

Algorithm 4: *Greedy ONE*.

These algorithms can be included in various global search strategies. Combining items (centroids) of solution S' with the items of the other solution S'' and running Algorithm 1, we get a set of "child" solutions. These solutions are used as the neighborhoods, in which a better solution is sought for. Thus, the second solution S'' is a parameter of the neighborhood [32].

The general framework of the GA for the k-means and similar location problems can be described as Algorithm 5.

The objective function F_{fitness} is (1). We used the tournament selection (tournament replacement, see Algorithm 6) for Step 10 of Algorithm 5:

Such algorithms usually operate with a very small population, and other selection procedures do not improve the results significantly [41, 43, 46].

In the GAs with greedy heuristic crossover [43, 44], Algorithms 2 and 3 are used as the crossover genetic operator. These operators are computationally expensive due to multiple runs of the *KMeans* algorithm. In the case of large-scale problems and very strict time limitation, GAs with greedy heuristic crossover operator performs only few iterations for large-scale problems. The population size is usually small, 10–25 chromosomes. Dynamically growing populations [43, 44] are able to improve the results. In this case, Step 7 of Algorithm 5 is replaced by the following procedure (see Algorithm 7).

Thus, in this paper, we intend to improve the GAs with greedy heuristic crossover operator which can be described as follows [43, 46]:

k-GA-ONE: GA framework (Algorithm 5) with *Greedy ONE* as the crossover operator, *Tournament* selection (Algorithm 6), dynamic population size adjustment (Algorithm 7), and empty mutation operator.

k-GA-FULL: the same but *Greedy FULL* crossover operator.

k-GA-RND: the same but the crossover operator *Greedy FULL* or *Greedy ONE* is selected randomly with equal probability.

The empty mutation operator can be replaced with a known or new procedure described in Section 3.

3. Known Methods of Increasing Population Diversity in the Genetic Algorithms

Despite the widespread use of various genetic algorithms for the k-means problems in the modern literature, there is practically no systematization of the approaches used [56–59]. For various methods of chromosome encoding, various mutation operators have been developed: bit inversion for binary encoding [50], exchange, insert, inverse, and offset permutation

Require: Initial population size N_{POP} (in ourExperiments, $N_{\text{POP}} = 10$).
(1) Assign $N_{\text{iter}} \longleftarrow 0$. Generate N_{POP} initial solutions $S_1, \ldots, S_{N_{\text{POP}}} \subset \{\{A_1, \ldots, A_N\}\}$ where $|S_i| = k \forall i = \overline{1, N_{\text{POP}}}$. ForEach initial solution, run the *kMeans* algorithm: $S_l \longleftarrow kMeans(S_l)$; $f_k \longleftarrow F_{\text{fitness}}(S_l), l = \overline{1, N_{\text{POP}}}$
(2) **loop**
(3) $N_{\text{iter}} \longleftarrow N_{\text{iter}} + 1$
(4) **if** stop condition is satisfied
(5) **return** solution S_{i^*} from the population with minimal value of f_{i^*}
(6) **end if**
(7) Selection: Randomly choose two indexes $i_1, i_2 \in \{\overline{1, N_{\text{POP}}}\}, i_1 \neq i_2$
(8) Run chosen crossover operator: $S_C \longleftarrow Crossover(S_{i_1}, S_{i_2})$
(9) Run chosen mutation operator: $S_C \longleftarrow Mutation(S_C)$
(10) Run chosen procedure to replace a solution in the population
(11) E**nd loop**

Algorithm 5: GA with real chromosome encoding for the k-means and p-median problems.

Randomly choose two indexes $i_4, i_5 \in \{\overline{1, N_{\text{POP}}}\}$
if $f_{i_4} > f_{i_5}$
$i_3 \longleftarrow i_4$
E**lse**
$i_3 \longleftarrow i_5$
E**nd if**
$S_{k_3} \longleftarrow S_C; f_{k_3} \longleftarrow F_{\text{fitness}}(S_C)$

Algorithm 6: *Tournament* (Step 10 of Algorithm 4).

$N_{\text{iter}} \leftarrow N_{\text{iter}} + 1$; $N_{\text{POP}} \leftarrow \max\{N_{\text{POP}}, \lceil \sqrt{1 + N_{\text{iter}}} \rceil$
if N_{POP} has changed
initialize the new individual $S_{N_{\text{POP}}}$: generate randomly, $|S| = p$; $S_{N_{POP}} \leftarrow kMeans(S)$
end if

Algorithm 7: Dynamic population size adjustment (replacement for Step 7 of Algorithm 4).

[60] for variable length chromosomes, Gaussian mutation [61], and polynomial mutation for real coding [62, 63]. Some studies suggest a combination of the mutation operators [64] or the self-adaptive mutation operators [65–67]. The efficiency of various mutation operators depends on the GA parameters [53, 68, 69] and problem type [70, 71]. However, the number of various mutation operators with real encoding for continuous problems is very limited.

The GA for the network p-median problem described in [72] includes the hypermutation operator, which consists in an attempt to replace each gene in the chromosome with each gene from the set of genes that were not originally part of the processed chromosome. After each replacement, the algorithm checks for the improvement of the objective function value. The operator is computationally expensive due to numerous checks of the objective function and actually similar to the local search principle embedded in the j-means algorithm [73]. In [74], the hypermutation algorithm was further developed as the nearest four neighbors' algorithm. The idea is to reduce computational costs by reducing the set of genes used for the replacement to the nearest neighbors of the gene being replaced. In several works [37, 75, 76], the authors propose using the *kMeans* algorithm as a mutation operator.

Each of these algorithms declares a local search as a mutation operator. The GA framework allows us to use a wide variety of genetic operator options. However, the local search is designed to improve an arbitrary solution by transforming it into a local optimum and thereby reducing, rather than increasing, the variety of chromosomes (solutions).

In [36, 42], the mutation operator is as follows (uniform random mutation). Randomly generate $r \sim U[0, 1)$. If $r < \mu$ (where μ is mutation probability), then the chromosome will mutate. Randomly generate $b \sim U[0, 1)$. If the current position of a centroid is $X_j = (x_{j,1}, \ldots, x_{x,d})$, the mutation operator modifies it as follows:

$$x_{j,l} \longleftarrow \begin{cases} x_{j,l} \pm 2bx_{j,l}, & x_{j,l} \neq 0, \\ \pm 2b, & x_{j,l} = 0. \end{cases} \tag{3}$$

Signs "+" and "–" are used with the same probability [42]. This mutation operator shifts the centroid coordinates

randomly. A similar technique with an "amplification factor" was used in [44, 77]. However, the local minima distribution among the search space is not uniform [49]: the new local minima of (1) can be found with higher probability in some neighborhood of a known local minimum than in a neighborhood of a randomly chosen point (here, by a neighborhood, we do not necessarily mean an ε-neighborhood but any subset of solutions which can be obtained by application of some defined procedure to the current solution). Combining local minima (subsets of centroids from two locally minimal solutions) must usually outperform the random shift of the centroid coordinates. The idea of combining local minima is the basic idea of the greedy heuristic crossover operator in genetic algorithms [38, 43] and other algorithms [21]. The greedy heuristic crossover operator for the discrete p-median problem proposed in [41] and adapted for continuous p-median and k-means problems in [38, 43] was used in the GAs without any mutation operator. Such algorithms demonstrate more accurate results in comparison with many other algorithms for practically important middle-size problems.

The other common approach to increasing the diversity in a population is to create subpopulations that develop more or less autonomously. Algorithms that produce subpopulations containing individuals gathered around optima are a wide class of such methods. The fitness sharing method [78] allows the evolutionary algorithm to search simultaneously in different areas (niches) corresponding to different local (or global) optima, i.e., this method allows one to identify and localize multiple optima in search space. The group of crowding methods [79–81] also uses a niche approach. The general concept of crowding is for individuals to fight for survival with similar offspring and apply tournament selection to a high-likeness parent-child pair. The main idea of the genetic chromodynamics [82] is to force the formation and maintenance of stable subpopulations. The proposed scheme of local interaction provides stabilization of the subpopulation in the early stages of the search. Subpopulations co-develop and converge to several optimal solutions.

In [83], the authors present the roaming optimization method. By using subpopulations developing in isolation, multiple optima are found. This method uses the tendency of evolutionary algorithms to premature convergence, turning this disadvantage into an advantage in the process of detecting local optima.

4. New Modifications to the Genetic Algorithms

The essence of our new mutation operator (greedy heuristic mutation, GHM) is as follows. We perform the crossover operator to the single parent chromosome and a randomly generated chromosome improved by the *kMeans* algorithm. In Step 9 of Algorithm 5, the *mutation* operator is replaced with Algorithm 8.

Despite small populations in the genetic algorithms with the greedy agglomerative crossover, the application of a simple approach with two subpopulations allows us to improve the result of the algorithm. In our research, within the population, we organized two subpopulations of equal volume. For the crossover and tournament, both chromosomes are mainly selected within the same subpopulation. If one of the subpopulations during a certain number of iterations does not provide an improvement in solutions and its record (the best solution) is inferior to the record of the second subpopulation, its individuals are replaced by new ones (reinitialization of the population). We assumed that chromosomes in the same subpopulation tend to develop in a similar way under the influence of crossover. Mutation of a separate chromosome increases the population diversity; however, under the influence of the crossover, the differences are gradually levelled. Reinitialization of a subpopulation is a substitute for a complete restart of the algorithm while maintaining the record. Thus, Step 7 of Algorithm 4 (selection) is transformed as Algorithm 9.

Similarly, Step 10 of Algorithm 5 changes (see Algorithm 10).

An additional Step is added to Algorithm 5 (see Algorithm 11).

The idea of the Variable Neighborhood Search with randomized neighborhoods (see [32]) is also based on applying the greedy heuristic procedures (Algorithms 2 and 3) to a current solution and a randomly generated one transformed into a local minimum by Algorithm 1. Our computational experiments (see Section 5) show that the new genetic algorithms with GHM as the mutation operator outperform both the original genetic algorithms with greedy agglomerative crossover operator (Algorithm 4 with empty mutation) and the Variable Neighborhood Search with randomized neighborhoods.

As mentioned before, the greedy agglomerative crossover operator is a computationally expensive algorithm. In Algorithm 2, the objective function calculation $F_i' \leftarrow F(S')$ is performed more than $(K-k)\cdot k$ times. Therefore, such algorithms are traditionally considered as methods for solving comparatively small problems (hundred thousands of data points and hundreds of centers). However, the rapid development of the massive parallel processing systems (GPUs) allows us to solve the large-scale problems with reasonable time expenses (minutes).

One of the most important issues of the GAs is the convergence of the entire population into some narrow area (population degeneration) around some local minimum. On the first crossover iterations, the "child" solutions usually have significant advantages in the objective function value in comparison with their "parents" due to the ability of the greedy agglomerative crossover operator to choose much better solutions in comparison with the k-means procedure. On a single central processor unit, such GAs manage to perform only few crossover operations due to the computationally expensive $Greedy()$, and the population diversity problem is not important. Our computational experiments show that, with an increase in the computational capacities and increase of the population size (which grows dynamically with the iteration number), the mutation operator plays more important role.

5. Computational Experiments

Parallel (CUDA) implementations of the $kMeans()$ algorithm are known [84, 85], and we used this approach in our experiments. All other algorithms were realized on the central processor unit.

Require: Solution S_C.
Randomly generate new solution S', $|S| = p$; $S' \longleftarrow kMeans(S')$
$S^C \longleftarrow Crossover(S, S')$
if $F(S'_C) < F(S_C)$
$S_C \longleftarrow S'_C$
end if
return S_C

Algorithm 8: *GHM*: version of *Mutation* operator.

Randomly choose $p_{intersubpop} \in [0, 1)$
if $p_{intersubpop} < 0.1$
Randomly choose two indexes $k_1, k_2 \in \{\overline{1, N_{\text{POP}}}\}, k_1 \neq k_2$; $Pop_{\text{Num}} \longleftarrow 0$
else if $p_{intersubpop} < 0.55$
Randomly choose two indexes $k_1, k_2 \in \{\overline{1, [N_{\text{POP}}/2] - 1}\}, k_1 \neq k_2$; $Pop_{\text{Num}} \longleftarrow 1$
Else
Randomly choose two indexes $k_1, k_2 \in \{\overline{[N_{\text{POP}}/2], N_{\text{POP}}}\}, k_1 \neq k_2$; $Pop_{\text{Num}} \longleftarrow 2$
End if

Algorithm 9: Interpopulation selection (replacement for Step 7 of Algorithm 4 with subpopulations).

Randomly choose $p_{\text{intersubpop}} \in [0, 1)$
if $Pop_{\text{Num}} = 0$
Randomly choose two indexes $k_3, k_4 \in \{\overline{1, N_{\text{POP}}}\}, k_3 \neq k_4$
Else if $Pop_{Num} = 1$
Randomly choose two indexes $k_3, k_4 \in \{\overline{1, [N_{POP}/2] - 1}\}, k_3 \neq k_4$
Else
Randomly choose two indexes $k_3, k_4 \in \{\overline{[N_{POP}/2], N_{POP}}\}, k_3 \neq k_4$
End if
if $f_{k_4} > f_{k_5}$
$k_3 \longleftarrow k_4$
Else
$k_3 \longleftarrow k_5$
End if
$S_{k_3} \longleftarrow S_C$; $f_{k_3} \longleftarrow F_{fitness}(S_C)$

Algorithm 10: *Tournament* (Step 10 of Algorithm 4 with subpopulations).

if the algorithm gave no improvement during $2N_{POP}$ iterations
Reinitialize all solutions in the subpopulation with indexes $\{\overline{1, [N_{\text{POP}}/2] - 1}\}$ or $\{\overline{[N_{\text{POP}}/2], N_{\text{POP}}}\}$.
end if

Algorithm 11: Additional step of Algorithm 4 with subpopulations (Step 10a).

For our experiments, we used the classic datasets from the UCI and Clustering basic benchmark repositories:

(a) Individual Household Electric Power Consumption (IHEPC): energy consumption data of households during several years (more than 2 million data vectors, 7 dimensions), 0–1 normalized data; "date" and "time" columns removed.

(b) SUSY ($5 \cdot 10^6$ data vectors, 18 dimensions), 0–1 normalized data. Here, we do not take into account the true labelling provided by the database, and use this dataset to search for internal structure in the data.

(c) Chess (King-Rook vs. King-Pawn, 3196 Boolean data vectors, 36 dimensions).

(d) BIRCH3 [10]: groups of points of random size on a plane (100000 data vectors, 2 dimensions).

Table 1: Comparative results for SUSY dataset, $k = 25$ clusters and time limitation 500 seconds.

Algorithm	Objective function value Summarized after 30 attempts				
	Min (record)	Max (worst)	Avg.	Median Value	Std. dev
(A) *Known algorithms*					
k-Means (multistart)	924115.13	925244.06	924548.63	924317.69	419.9655
j-Means	926428.75	928250.50	927328.31	927256.00	741.0420
***k*-GH-VNS1**	924227.38	924317.56	924276.83	924265.25	41.0833
k-GH-VNS2	924115.13	925141.50	924822.84	924994.22	410.9538
k-GH-VNS3	924712.69	925243.88	924997.23	925141.50	222.6291
k-GA-FULL	925134.50	925243.88	925198.99	925243.81	56.0092
k-GA-RND	924227.38	925141.50	924635.38	924265.31	473.6178
***k*-GA-ONE**	924115.13	924317.50	924251.27	924265.25	63.1124
(B) *Genetic algorithms with new combinations of known genetic operators*					
k-GA-FULL-m1, $\mu = 0.001$	924712.69	925243.81	925109.47	925141.50	181.4919
k-GA-FULL-m1, $\mu = 0.01$	924265.25	925243.88	925032.71	925141.50	344.9659
k-GA-FULL-m1, $\mu = 0.1$	924265.25	925243.88	925015.63	925141.50	341.6232
k-GA-FULL-m2, $\mu = 0.001$	924227.38	925243.81	925021.54	925141.50	352.6434
k-GA-FULL-m2, $\mu = 0.01$	924877.00	925141.56	925096.58	925141.50	98.6034
k-GA-FULL-m2, $\mu = 0.1$	924115.13	925243.88	924864.72	925141.50	485.5295
k-GA-RND-m1, $\mu = 0.001$	924265.25	925243.88	924714.60	924877.00	424.5925
k-GA-RND-m1, $\mu = 0.01$	924115.06	925141.50	924457.88	924227.38	461.0992
k-GA-RND-m1, $\mu = 0.1$	924227.38	925243.81	924539.75	924317.50	448.1161
***k*-GA-RND-m2,** $\mu = 0.001 \Downarrow$	924115.13	925243.81	924516.25	924265.25	467.2019
k-GA-RND-m2, $\mu = 0.01$	924115.06	925051.50	924357.08	924317.50	319.4935
***k*-GA-RND-m2**, $\mu = 0.1 \uparrow$	924115.06	924265.25	924200.90	924265.25	80.2563
***k*-GA-ONE-m1**, $\mu = 0.001 \Updownarrow$	924227.38	924317.63	924282.27	924265.25	35.6455
k-GA-ONE-m1, $\mu = 0.01$	924115.13	925864.13	924626.94	924317.50	661.5964
***k*-GA-ONE-m1**, $\mu = 0.1 \Updownarrow$	924115.06	924317.56	924231.88	924265.25	85.8271
k-GA-ONE-m2, $\mu = 0.001$	924227.38	925925.75	924933.53	924876.94	709.7990
k-GA-ONE-m2, $\mu = 0.01$	924115.06	925243.81	924518.29	924317.50	466.6610
k-GA-ONE-m2, $\mu = 0.1$	924265.25	924891.75	924384.63	924317.50	224.9720
(C) *Genetic algorithms with a new mutation operator*					
k-GA-FULL-GHM	924115.13	924994.00	924381.57	924227.38	383.1728
k-GA-FULL-SUBPOP	924265.25	925243.81	924905.77	925141.50	439.1485
k-GA-RND-GHM	924227.38	925243.88	924804.63	925140.75	486.4890
k-GA-RND-SUBPOP	924115.13	924227.38	924131.16	924115.13	42.4265
k-GA-ONE-GHM	924115.06	924265.25	924184.66	924227.38	66.4470
***k*-GA-ONE-SUBPOP** $\uparrow\Uparrow$	924115.06	924115.13	924115.07	924115.06	0.0236

Note (for all tables): "$\uparrow\Uparrow$" denotes the advantage of the best algorithm in this group over known algorithms (group A) is statistically significant ("$\uparrow$" for t-test and"$\Uparrow$" for Mann–Whitney U test); "$\downarrow\Downarrow$" denotes the disadvantage of the best algorithm in this group over known algorithms is statistically significant; "$\updownarrow\Updownarrow$" denotes the advantage or disadvantage is statistically insignificant. Significance level is 0.99.

(e) Europe (map of Europe, 169308 data vectors, 2 dimensions).

(f) Mopsi-Joensuu: locations of users (6014 data vectors, 2 dimensions).

The test system consisted of Intel Core 2 DuoE8400 CPU, 16GB RAM, NVIDIA GeForce GTX1050ti GPU with 4096 MB RAM, floating-point performance 2,138 g flops. For all datasets, 30 attempts were made to run each of 32 algorithms (Tables 1–6).

For comparison, we used the genetic algorithms with greedy heuristic crossover (k-GA-FULL, k-GA-ONE, and k-GA-RND described in Section 2) as well as the *kMeans* procedure in the multistart mode and *j*-Means algorithm (centers are replaced with the data vectors) [73]. In addition, we ran various Variable Neighborhood Search (VNS) algorithms with randomized neighborhoods formed by greedy heuristic procedure [32], see algorithms k-GH-VNS1 and k-GH-VNS2. For algorithms launched in the multistart mode (*j*-Means and *kMeans*), only the best results achieved in each attempt were recorded. The minimum, maximum, average, and median objective function values and its standard deviation are summarized after 30 runs. For all algorithms, we used the same realization of the *kMeans* procedure which consumes the absolute majority of the computation time. The initial population size for all genetic algorithms consisted of $N_{POP} = 10$ chromosomes.

All algorithms were classified into three groups. The first group of algorithms consists of known algorithms including the genetic algorithms with greedy heuristic crossover. Algorithms of the second group are the genetic algorithms with greedy heuristic crossover and known mutation operators (k-GA-*xxx*-m1 for uniform random mutation and k-GA-*xxx*-m2 for scramble mutation [86] where a gene (centroid)

Table 2: Comparative results for IHEPC dataset, $k = 100$ clusters and time limitation 1000 seconds.

Algorithm	Objective function value Summarized after 30 attempts				
	Min (record)	Max (worst)	Avg.	Median value	Std. dev
(A) *Known algorithms*					
k-Means (multistart)	3609.0322	3767.3699	3682.9703	3702.0381	66.0620
j-Means	3222.0024	3469.1904	3330.8628	3308.4663	95.1764
k-GH-VNS1	3194.3730	3218.3042	3200.5752	3194.9126	10.2875
***k*-GH-VNS2**	3192.3491	3193.9951	3193.4711	3193.2563	0.7125
k-GH-VNS3	3192.9253	3195.6731	3194.7855	3195.6541	1.2618
k-GA-FULL	3194.5671	3196.0510	3195.5192	3195.6592	0.5586
k-GA-RND	3193.8230	3194.9165	3194.6958	3194.9133	0.4879
k-GA-ONE	3194.2986	3194.9141	3194.7907	3194.9136	0.2751
(B) *Genetic algorithms with new combinations of known genetic operators*					
k-GA-FULL-m1, $\mu = 0.001$	3195.1061	3196.8482	3196.0125	3196.2146	0.8129
k-GA-FULL-m1, $\mu = 0.01$	3195.1055	3195.6616	3195.3321	3195.1191	0.2999
k-GA-FULL-m1, $\mu = 0.1$	3195.1050	3197.3811	3196.2253	3196.3418	0.8776
k-GA-FULL-m2, $\mu = 0.001$	3194.2142	3197.0143	3195.4257	3195.5142	0.9382
k-GA-FULL-m2, $\mu = 0.01$	3194.3091	3197.0327	3195.5808	3195.6631	1.0298
k-GA-FULL-m2, $\mu = 0.1$	3194.0044	3196.3257	3195.2396	3195.1074	0.8542
k-GA-RND-m1, $\mu = 0.001$	3194.0125	3196.0512	3195.0221	3194.9287	0.7481
k-GA-RND-m1, $\mu = 0.01$	3193.8232	3196.1799	3194.9487	3194.9133	0.8346
k-GA-RND-m1, $\mu = 0.1$	3193.9868	3195.0732	3194.3911	3194.2983	0.4045
k-GA-RND-m2, $\mu = 0.001$	3194.0418	3195.9175	3194.9819	3194.7105	0.8723
***k*-GA-RND-m2**, $\mu = 0.01$ $\downarrow\Downarrow$	3193.0942	3194.9136	3194.0628	3193.8240	0.8194
k-GA-RND-m2, $\mu = 0.1$	3194.2979	3194.9136	3194.6118	3194.4761	0.2839
k-GA-ONE-m1, $\mu = 0.001$	3194.7247	3196.4271	3195.5692	3195.1289	0.7018
k-GA-ONE-m1, $\mu = 0.01$	3194.5459	3196.2983	3195.3459	3195.0737	0.7263
k-GA-ONE-m1, $\mu = 0.1$	3194.5449	3213.1353	3198.7382	3195.0747	8.0666
k-GA-ONE-m2, $\mu = 0.001$	3194.3081	3195.4285	3194.8905	3194.8857	0.4765
k-GA-ONE-m2, $\mu = 0.01$	3194.2979	3195.0737	3194.7310	3194.9126	0.3341
k-GA-ONE-m2, $\mu = 0.1$	3194.2986	3194.9141	3194.7027	3194.9126	0.2949
(C) *Genetic algorithms with a new mutation operator*					
k-GA-FULL-GHM	3195.0874	3196.7456	3195.8983	3195.6616	0.6483
k-GA-FULL-SUBPOP	3195.0902	3196.5621	3195.6319	3195.5258	0.6157
***k*-GA-RND-GHM** $\updownarrow\Updownarrow$	3193.2725	3194.9143	3193.4996	3193.8240	0.7473
k-GA-RND-SUBPOP	3193.2725	3195.0774	3194.1428	3193.8240	0.6876
k-GA-ONE-GHM	3194.9131	3195.0774	3195.0426	3195.0737	0.0724
***k*-GA-ONE-SUBPOP** $\downarrow\Updownarrow$	3193.2725	3195.0737	3194.6758	3193.8240	0.2959

is replaced with a randomly chosen data point). We performed our experiments with various values of mutation probability μ. Algorithms of the third group are genetic algorithms with greedy agglomerative crossover and new instruments for maintaining the population diversity: k-GA-*xxx*-GHM are algorithms with the new GHM mutation operator, and k-GA-*xxx*-SUBPOP are algorithms with the new GHM mutation operator and two subpopulations.

In each group of algorithms, the best average and median values of the objective function (1) are underlined. We compared the best algorithms in the second and third groups with the best algorithm in the first group (the best of known algorithms) with the use of *t*-test and Mann–Whitney *U* test.

In the comparative analysis of algorithm efficiency, the choice of the unit of time plays an important role. The astronomical time spent by an algorithm strongly depends on the peculiarities of its implementation, the ability of the compiler to optimize the program code, and the fitness of the hardware to execute the code of a specific algorithm. Algorithms are often estimated by comparing the number of iterations performed (for example, the number of population generations for a GA) or the number of evaluations of the objective function. In our case, some of the algorithms are not evolutionary, and in genetic algorithms, the execution time of the crossover operator with the embedded *kMeans* algorithm can differ hundreds of times. Therefore, comparing the number of generations is unacceptable. Comparison of the objective function calculations is also not quite correct. Firstly, the *kMeans* algorithm which consume almost all of the processor time, do not calculate (1) directly. Secondly, during the operation of the greedy agglomerative crossover operator, the number of centroids changes (decreases from $2k$ down to k or from $k + 1$ down to k), and the time spent on computing the objective function also varies. Therefore, we nevertheless chose astronomical time as a scale for comparing algorithms. Moreover, all the algorithms use the same implementation of the *kMeans* algorithm launched under the same conditions.

In our computational experiments, the time limitation was used as the stop condition for all algorithms. As can be

Table 3: Comparative results for Chess dataset, $k = 50$ clusters and time limitation 10 seconds.

Algorithm	Objective function value Summarized after 30 attempts				
	Min (record)	Min (worst)	Avg.	Median value	Std. dev
(A) *Known algorithms*					
k-Means (multistart)	6926.22	6958.36	6941.13	6939.64	11.2781
j-Means	6938.97	6987.53	6962.71	6961.96	18.6573
***k*-GH-VNS1**	6851.11	6855.66	6853.08	6853.21	1.5482
k-GH-VNS2	6851.07	6857.08	6853.96	6854.35	2.4912
k-GH-VNS3	6851.15	6859.06	6854.82	6855.94	3.5286
k-GA-FULL	6864.33	6867.14	6865.68	6865.66	1.2282
k-GA-RND	6851.41	6858.32	6854.64	6854.56	2.9540
k-GA-ONE	6851.44	6860.75	6856.01	6856.16	4.0019
(B) *Genetic algorithms with new combinations of known genetic operators*					
k-GA-FULL-m1, $\mu = 0.001$	6860.04	6870.22	6867.09	6867.79	2.6577
k-GA-FULL-m1, $\mu = 0.01$	6858.87	6870.85	6866.83	6867.13	3.3960
k-GA-FULL-m1, $\mu = 0.1$	6861.89	6871.42	6868.57	6868.84	2.3854
k-GA-FULL-m2, $\mu = 0.001$	6863.43	6872.63	6868.37	6867.97	2.4871
k-GA-FULL-m2, $\mu = 0.01$	6865.95	6871.79	6868.40	6868.34	1.9573
k-GA-FULL-m2, $\mu = 0.1$	6863.01	6869.63	6867.21	6867.82	1.9583
k-GA-RND-m1, $\mu = 0.001$	6851.29	6856.30	6853.51	6853.35	1.6152
k-GA-RND-m1, $\mu = 0.01$	6851.67	6860.16	6854.67	6854.23	2.7174
k-GA-RND-m1, $\mu = 0.1$	6851.31	6859.33	6853.83	6853.31	2.3047
k-GA-RND-m2, $\mu = 0.001$	6852.11	6859.69	6855.23	6854.35	2.4314
k-GA-RND-m2, $\mu = 0.01$	6851.68	6858.65	6854.49	6853.68	1.9113
k-GA-RND-m2, $\mu = 0.1$	6851.23	6860.69	6854.17	6853.34	2.7578
k-GA-ONE-m1, $\mu = 0.001$	6851.35	6859.78	6853.79	6852.97	2.4321
k-GA-ONE-m1, $\mu = 0.01$	6851.18	6857.68	6853.48	6853.41	1.7505
***k*-GA-ONE-m1,** $\mu = 0.1$ ↕⇑	6851.17	6858.90	6852.82	6852.02	2.0071
k-GA-ONE-m2, $\mu = 0.001$	6851.19	6860.02	6854.86	6854.32	3.2739
k-GA-ONE-m2, $\mu = 0.01$	6851.35	6857.28	6854.09	6853.67	1.9327
k-GA-ONE-m2, $\mu = 0.1$	6851.26	6854.55	6852.87	6853.18	1.0231
(C) *Genetic algorithms with a new mutation operator*					
k-GA-FULL-GHM	6852.40	6857.80	6855.43	6855.78	1.7635
k-GA-FULL-SUBPOP	6852.81	6858.47	6855.48	6855.64	1.7971
***k*-GA-RND-GHM** ↑⇑	6851.12	6853.19	6851.61	6851.18	0.8080
k-GA-RND-SUBPOP	6851.11	6855.98	6852.39	6852.12	1.4238
k-GA-ONE-GHM	6851.11	6853.81	6852.04	6851.34	1.0307
k-GA-ONE-SUBPOP	6851.09	6853.89	6851.84	6851.25	1.1297

Table 4: Comparative results for BIRCH3 dataset, $k = 100$ clusters and time limitation 10 seconds.

Algorithm	Objective function value Summarized after 30 attempts				
	Min (record)	Max (worst)	Avg.	Median value	Std. dev
(A) *Known algorithms*					
k-Means (multistart)	$7.83252E+13$	$9.52953E+13$	$8.93284E+13$	$9.02011E+13$	$52.9723E+11$
j-Means	$4.03770E+13$	$6.59148E+13$	$4.78407E+13$	$4.69827E+13$	$58.1576E+11$
k-GH-VNS1	$3.71492E+13$	$3.74385E+13$	$3.72471E+13$	$3.72063E+13$	$0.76473E+11$
k-GH-VNS2	$3.71530E+13$	$3.72899E+13$	$3.72045E+13$	$3.71994E+13$	$0.27407E+11$
k-GH-VNS3	$3.71558E+13$	$3.73439E+13$	$3.72163E+13$	$3.72011E+13$	$0.50354E+11$
k-GA-FULL	$3.72086E+13$	$3.73018E+13$	$3.72411E+13$	$3.72413E+13$	$0.25040E+11$
k-GA-RND	$3.71538E+13$	$3.72083E+13$	$3.71998E+13$	$3.72071E+13$	$0.16590E+11$
k-GA-ONE	$3.71526E+13$	$3.72083E+13$	$3.71925E+13$	$3.72072E+13$	$0.24657E+11$
(B) *Genetic algorithms with new combinations of known genetic operators*					
k-GA-FULL-m1, $\mu = 0.001$	$3.72566E+13$	$3.77255E+13$	$3.74320E+13$	$3.74355E+13$	$1.10195E+11$
k-GA-FULL-m1, $\mu = 0.01$	$3.72566E+13$	$3.79192E+13$	$3.74764E+13$	$3.74532E+13$	$1.61443E+11$
k-GA-FULL-m1, $\mu = 0.1$	$3.72893E+13$	$3.84583E+13$	$3.76484E+13$	$3.74674E+13$	$3.91344E+11$
k-GA-FULL-m2, $\mu = 0.001$	$3.72397E+13$	$3.85824E+13$	$3.76392E+13$	$3.75036E+13$	$3.95201E+11$

Table 4: Continued.

Algorithm	Objective function value Summarized after 30 attempts				
	Min (record)	Max (worst)	Avg.	Median value	Std. dev
k-GA-FULL-m2, $\mu = 0.01$	$3.72859E+13$	$3.75769E+13$	$3.74216E+13$	$3.74359E+13$	$0.78992E+11$
k-GA-FULL-m2, $\mu = 0.1$	$3.72555E+13$	$3.78012E+13$	$3.74676E+13$	$3.74355E+13$	$1.56028E+11$
k-GA-RND-m1, $\mu = 0.001$	$3.71491E+13$	$3.73835E+13$	$3.72213E+13$	$3.72072E+13$	$0.52620E+11$
k-GA-RND-m1, $\mu = 0.01$	$3.71967E+13$	$3.72074E+13$	$3.72039E+13$	$3.72070E+13$	$0.03821E+11$
k-GA-RND-m1, $\mu = 0.1$	$3.71466E+13$	$3.72074E+13$	$3.72007E+13$	$3.72072E+13$	$0.15455E+11$
k-GA-RND-m2, $\mu = 0.001$	$3.71530E+13$	$3.72072E+13$	$3.71979E+13$	$3.72011E+13$	$0.14772E+11$
k-GA-RND-m2, $\mu = 0.01$	$3.71466E+13$	$3.72083E+13$	$3.72015E+13$	$3.72072E+13$	$0.15465E+11$
k-GA-RND-m2, $\mu = 0.1$	$3.71592E+13$	$3.72072E+13$	$3.71980E+13$	$3.72072E+13$	$0.16092E+11$
k-GA-ONE-m1, $\mu = 0.001$	$3.72009E+13$	$3.72102E+13$	$3.72053E+13$	$3.72072E+13$	$0.03291E+11$
k-GA-ONE-m1, $\mu = 0.01$ ↕⇕	$3.71472E+13$	$3.72074E+13$	$3.71938E+13$	$3.72009E+13$	$0.22370E+11$
k-GA-ONE-m1, $\mu = 0.1$	$3.71969E+13$	$3.72869E+13$	$3.72107E+13$	$3.72072E+13$	$0.21359E+11$
k-GA-ONE-m2, $\mu = 0.001$	$3.71681E+13$	$3.72430E+13$	$3.72062E+13$	$3.72072E+13$	$0.14321E+11$
k-GA-ONE-m2, $\mu = 0.01$	$3.71968E+13$	$3.73260E+13$	$3.72133E+13$	$3.72072E+13$	$0.31388E+11$
k-GA-ONE-m2, $\mu = 0.1$	$3.71528E+13$	$3.72077E+13$	$3.71984E+13$	$3.72072E+13$	$0.18669E+11$
(C) *Genetic algorithms with a new mutation operator*					
k-GA-FULL-GHM	$3.71908E+13$	$3.73862E+13$	$3.72381E+13$	$3.72108E+13$	$0.61387E+11$
k-GA-FULL-SUBPOP	$3.71934E+13$	$3.73928E+13$	$3.72527E+13$	$3.72162E+13$	$0.62692E+11$
k-GA-RND-GHM	$3.71977E+13$	$3.72072E+13$	$3.72045E+13$	$3.72071E+13$	$0.03486E+11$
k-GA-RND-SUBPOP ↕⇕	$3.71529E+13$	$3.72072E+13$	$3.71938E+13$	$3.72024E+13$	$0.13139E+11$
k-GA-ONE-GHM	$3.71976E+13$	$3.72074E+13$	$3.72052E+13$	$3.72072E+13$	$0.03436E+11$
k-GA-ONE-SUBPOP	$3.72009E+13$	$3.72083E+13$	$3.72061E+13$	$3.72072E+13$	$0.02444E+11$

Table 5: Comparative results for Europe dataset, $k = 30$ clusters and time limitation 10 seconds.

Algorithm	Objective function value Summarized after 30 attempts				
	Min (record)	Max (worst)	Avg.	Median value	Std. dev
(A) *Known algorithms*					
k-Means (multistart)	$7.54173E+12$	$7.57541E+12$	$7.55414E+12$	$7.54853E+12$	$11.2422E+9$
j-Means	$7.55697E+12$	$7.71002E+12$	$7.64233E+12$	$7.64924E+12$	$47.5626E+9$
k-GH-VNS1	$7.49695E+12$	$7.50850E+12$	$7.50287E+12$	$7.50346E+12$	$3.4668E+9$
k-GH-VNS2	$7.50035E+12$	$7.51500E+12$	$7.50727E+12$	$7.50753E+12$	$4.3317E+9$
k-GH-VNS3	$7.50346E+12$	$7.51458E+12$	$7.50803E+12$	$7.50919E+12$	$3.7463E+9$
k-GA-FULL	$7.50359E+12$	$7.51505E+12$	$7.51020E+12$	$7.50998E+12$	$2.9113E+9$
k-GA-RND	$7.50036E+12$	$7.50752E+12$	$7.50350E+12$	$7.50360E+12$	$1.5822E+9$
k-GA-ONE	$7.49831E+12$	$7.55141E+12$	$7.50696E+12$	$7.50372E+12$	$12.4626E+9$
(B) *Genetic algorithms with new combinations of known genetic operators*					
k-GA-FULL-m1, $\mu = 0.001$	$7.50359E+12$	$7.51488E+12$	$7.51099E+12$	$7.51182E+12$	$2.8380E+9$
k-GA-FULL-m1, $\mu = 0.01$	$7.50359E+12$	$7.51504E+12$	$7.51057E+12$	$7.51065E+12$	$2.7895E+9$
k-GA-FULL-m1, $\mu = 0.1$	$7.50362E+12$	$7.51504E+12$	$7.51020E+12$	$7.51065E+12$	$3.1105E+9$
k-GA-FULL-m2, $\mu = 0.001$	$7.50359E+12$	$7.51505E+12$	$7.51003E+12$	$7.50919E+12$	$3.4939E+9$
k-GA-FULL-m2, $\mu = 0.01$	$7.50077E+12$	$7.51503E+12$	$7.50926E+12$	$7.51108E+12$	$4.6759E+9$
k-GA-FULL-m2, $\mu = 0.1$	$7.49595E+12$	$7.51417E+12$	$7.50831E+12$	$7.50919E+12$	$4.5349E+9$
k-GA-RND-m1, $\mu = 0.001$	$7.50102E+12$	$7.50723E+12$	$7.50468E+12$	$7.50472E+12$	$2.0215E+9$
k-GA-RND-m1, $\mu = 0.01$	$7.50066E+12$	$7.50652E+12$	$7.50321E+12$	$7.50361E+12$	$1.9696E+9$
k-GA-RND-m1, $\mu = 0.1$	$7.50091E+12$	$7.50704E+12$	$7.50382E+12$	$7.50375E+12$	$1.9825E+9$
k-GA-RND-m2, $\mu = 0.001$	$7.49812E+12$	$7.50851E+12$	$7.50478E+12$	$7.50528E+12$	$2.7402E+9$
k-GA-RND-m2, $\mu = 0.01$	$7.49506E+12$	$7.50520E+12$	$7.50257E+12$	$7.50353E+12$	$2.5055E+9$
k-GA-RND-m2, $\mu = 0.1$↕⇕	$7.49573E+12$	$7.50677E+12$	$7.50256E+12$	$7.50251E+12$	$3.4842E+9$
k-GA-ONE-m1, $\mu = 0.001$	$7.49840E+12$	$7.55417E+12$	$7.50577E+12$	$7.50353E+12$	$13.5436E+9$
k-GA-ONE-m1, $\mu = 0.01$	$7.49234E+12$	$7.54085E+12$	$7.50941E+12$	$7.50363E+12$	$15.8707E+9$
k-GA-ONE-m1, $\mu = 0.1$	$7.50089E+12$	$7.55332E+12$	$7.51161E+12$	$7.50372E+12$	$17.5433E+9$
k-GA-ONE-m2, $\mu = 0.001$	$7.49731E+12$	$7.55152E+12$	$7.50997E+12$	$7.50270E+12$	$16.7714E+9$
k-GA-ONE-m2, $\mu = 0.01$	$7.50075E+12$	$7.55416E+12$	$7.50721E+12$	$7.50374E+12$	$13.1445E+9$
k-GA-ONE-m2, $\mu = 0.1$	$7.49523E+12$	$7.55026E+12$	$7.51145E+12$	$7.50363E+12$	$18.1598E+9$

Table 5: Continued.

Algorithm	Objective function value Summarized after 30 attempts				
	Min (record)	Max (worst)	Avg.	Median value	Std. dev
(C) *Genetic algorithms with a new mutation operator*					
k-GA-FULL-GHM	$7.50034E+12$	$7.50777E+12$	$7.50361E+12$	$7.50376E+12$	$2.4491E+9$
k-GA-FULL-SUBPOP ↑⇑	$7.49812E+12$	$7.50376E+12$	$7.50103E+12$	$7.50074E+12$	$1.4074E+9$
k-GA-RND-GHM	$7.49859E+12$	$7.50399E+12$	$7.50211E+12$	$7.50213E+12$	$1.6438E+9$
k-GA-RND-SUBPOP	$7.49957E+12$	$7.50364E+12$	$7.50124E+12$	$7.50081E+12$	$1.4788E+9$
k-GA-ONE-GHM	$7.49966E+12$	$7.50373E+12$	$7.50222E+12$	$7.50233E+12$	$1.4000E+9$
k-GA-ONE-SUBPOP	$7.50042E+12$	$7.50730E+12$	$7.50304E+12$	$7.50346E+12$	$1.9860E+9$

Table 6: Comparative results for Mopsi-Joensuu dataset, k = 300 clusters and time limitation 10 seconds.

Algorithm	Objective function value Summarized after 30 attempts				
	Min (record)	Max (worst)	Avg.	Median value	Std. dev
(A) *Known algorithms*					
k-Means (multistart)	5.3229	7.5711	6.6616	6.8050	0.6727
j-Means	2.7867	6.0623	4.2377	3.8380	1.1752
k-GH-VNS1	1.9960	3.4027	2.4989	2.3928	0.5308
k-GH-VNS2	3.1070	9.1468	6.9344	7.4990	2.2980
***k*-GH-VNS3**	0.1432	0.2974	0.1836	0.1592	0.0582
k-GA-FULL	0.1912	1.0615	0.4290	0.3564	0.2299
k-GA-RND	0.7039	2.5733	1.4348	1.2930	0.6968
k-GA-ONE	2.1265	7.3622	4.4011	4.3184	1.3787
(B) *Genetic algorithms with new combinations of known genetic operators*					
k-GA-FULL-m1, $\mu = 0.001$	0.1499	1.0124	0.6256	0.5817	0.2681
k-GA-FULL-m1, $\mu = 0.01$	0.1492	0.9951	0.3741	0.2949	0.2330
k-GA-FULL-m1, $\mu = 0.1$	0.1440	0.4354	0.2632	0.2699	0.0807
k-GA-FULL-m2, $\mu = 0.001$	0.1302	0.3872	0.2017	0.1589	0.0811
k-GA-FULL-m2, $\mu = 0.01$	0.1239	0.3441	0.1969	0.1552	0.0726
GA-FULL-m2, $\mu = 0.1$↕⇕	0.1123	0.2823	0.1622	0.1384	0.0535
k-GA-RND-m1, $\mu = 0.001$	0.4157	15.8679	7.1372	5.9456	5.6937
k-GA-RND-m1, $\mu = 0.01$	0.4061	20.5874	8.7251	6.3025	7.1967
k-GA-RND-m1, $\mu = 0.1$	0.2678	10.2416	3.4529	3.3582	2.7349
k-GA-RND-m2, $\mu = 0.001$	0.6825	7.2932	4.8789	3.9325	2.1523
k-GA-RND-m2, $\mu = 0.01$	0.6373	6.7522	3.2538	3.3473	1.5760
k-GA-RND-m2, $\mu = 0.1$	0.1366	15.2759	3.2085	1.5904	4.4927
k-GA-ONE-m1, $\mu = 0.001$	2.1527	8.2427	3.5267	3.2028	1.7428
k-GA-ONE-m1, $\mu = 0.01$	2.1336	9.1164	3.9120	3.6403	1.9144
k-GA-ONE-m1, $\mu = 0.1$	2.1199	5.9004	3.5209	3.4819	0.9954
k-GA-ONE-m2, $\mu = 0.001$	2.5137	14.8735	7.4719	3.8462	4.8952
k-GA-ONE-m2, $\mu = 0.01$	2.3010	20.5685	8.3449	4.9896	6.2828
k-GA-ONE-m2, $\mu = 0.1$	2.8133	8.4065	4.0472	3.5393	1.5569
(C) *Genetic algorithms with a new mutation operator*					
k-GA-FULL-GHM	0.1457	1.0108	0.3529	0.3077	0.2326
***k*-GA-FULL-SUBPOP** ↕⇕	0.1331	0.3186	0.1642	0.1384	0.0679
k-GA-RND-GHM	0.2647	3.8119	2.3451	2.6905	1.1910
k-GA-RND-SUBPOP	0.4014	3.8415	2.3430	2.8902	1.2284
k-GA-ONE-GHM	2.0543	8.2719	4.2152	4.0904	1.6552
k-GA-ONE-SUBPOP	2.2634	6.8275	4.0534	3.5353	1.6508

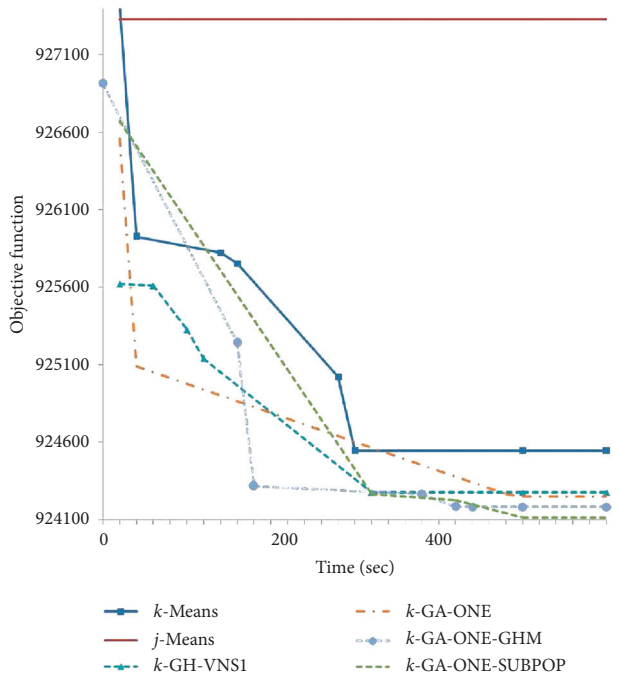

Figure 1: Comparative results of various algorithms for SUSY dataset ($N = 5 \cdot 10^6$ data vectors in $\mathbb{R}^{18}$ and $k = 25$ clusters).

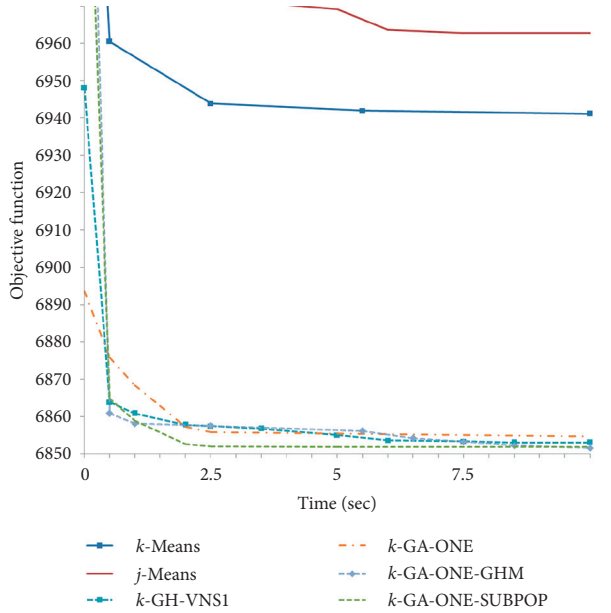

Figure 2: Comparative results of various algorithms for Chess dataset ($N = 3196$ in $\mathbb{R}^{36}$ and $k = 50$ clusters).

seen from Figures 1 and 2, the result of each algorithm depends on the elapsed time. Nevertheless, an advantage of the new algorithms remains regardless of the chosen time limit.

The range of values in all tables is small; nevertheless, the differences are statistically significant in several cases. In all cases, new algorithms with the greedy heuristic mutation outperform known ones or demonstrate approximately the same efficiency (difference in the results is statistically insignificant). Moreover, new algorithms demonstrate the stability of results (narrow range of objective function values). In most cases, the best results were achieved by the genetic algorithms with nonempty mutation operators.

6. Conclusions

When solving some large-scale clustering problems, traditional local search algorithms often give a result very far from the optimal solution. In this research we aimed at developing not only fast but also the most accurate algorithm, based on genetic algorithms with greedy heuristic crossover operator, for solving related optimization problems. Methods for obtaining solutions in a fixed time, which would be difficult to improve by known methods without a significant increase in computational costs, include genetic algorithms with a greedy agglomerative crossover operator. As the computational results presented in this article show, further improvement in the achieved result of such algorithms is possible by increasing the diversity in their populations.

Computational experiments show that the population diversity maintaining mechanisms such as mutation genetic operator and subpopulations improve the features of genetic algorithms with greedy heuristic crossover for the large-scale k-means problem. Moreover, the best results can be shown by algorithms with a mutation operator based on greedy heuristic crossover operator with a randomly generated chromosome (new greedy heuristic mutation).

The similarity in mathematical formulations of k-means, k-medoids, and p-median problems, as well as the problem of a mixture probability distribution separation, gives us a reasonable hope for the applicability of similar approaches to improving the results of solving those problems which determine possible directions for further research.

Data Availability

In our work, we used only data from the UCI Machine Learning and Clustering Basic Benchmark repositories which are available at https://archive.ics.uci.edu/ml/index.php and http://cs.joensuu.fi/sipu/datasets.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Ministry of Science and HigherEducation of the Russian Federation (State Contract no. FEFE-2020-0013).

References

[1] M. Garey, D. Johnson, and H. Witsenhausen, "The complexity of the generalized lloyd - max problem (Corresp.)," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 255-256, 1982.

[2] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "NP-hardness ofEuclidean sum-of-squares clustering," *Machine Learning*, vol. 75, no. 2, pp. 245–248, 2009.

[3] Z. Drezner and H. Hamacher, *Facility Location: Applications and Theory*, Springer-Verlag, Berlin, Germany, 2004.

[4] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[5] J. B. MacQueen, "Some methods of classification and analysis of multivariate observations,"vol. 1, pp. 281–297, in *Proceedings of the 5th Berkley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281–297, University of California Press, Berkeley, CA, USA, January 1967.

[6] L. Cooper, "Heuristic methods for location-allocation problems," *SIAM Review*, vol. 6, no. 1, pp. 37–53, 1964.

[7] J.-L. Jiang and X.-M. Yuan, "A heuristic algorithm for constrained multi-source weber problem - the variational inequality approach," *uropean Journal of Operational Research*, vol. 187, no. 2, pp. 357–370, 2008.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via theEMAlgorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[9] L. Kazakovtsev, D. Stashkov, M. Gudyma, and V. Kazakovtsev, "Algorithms with greedy heuristic procedures for mixture probability distribution separation," *Yugoslav Journal of Operations Research*, vol. 29, no. 1, pp. 51–67, 2019.

[10] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: anEffcient data clustering method for very large databases," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD'96)*, pp. 103–114, ACM, New York, NY, USA, June 1996.

[11] L. O'Callaghan, A. Meyerson, R. Motwani, N. Mishra, and S. Guha, "Streaming-data algorithms for high-quality clustering, dataEngineering," in *Proceedings 18th International Conference on DataEngineering*, pp. 685–694, IEEE, San Jose, CA, USA, March 2002.

[12] M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler, "StreamKM++," *ACM Journal ofExperimental Algorithmics*, vol. 17, no. 2, 4 pages, 2012.

[13] S. Masuyama, T. Ibaraki, and T. Hasegawa, "The computational complexity of the m-center problems on the plane," *The Transactions of the Institute ofElectronics and CommunicationEngineers of Japan*, vol. 64E, pp. 57–64, 1981.

[14] O. Kariv and S. L. Hakimi, "An algorithmic approach to network location problems. I: thep-centers," *SIAM Journal on Applied Mathematics*, vol. 37, no. 3, pp. 513–538, 1979.

[15] R.E. Kuenne and R. M. Soland, "Exact and approximate solutions to the multisource weber problem," *Mathematical Programming*, vol. 3-3, no. 1, pp. 193–209, 1972.

[16] L. M. Ostresh Jr, "The stepwise location-allocation problem: Exact solutions in continuous and discrete spaces," *Geographical Analysis*, vol. 10, no. 2, pp. 174–185, 1978.

[17] K.E. Rosing, "An optimal method for solving the (generalized) multi-weber problem," *uropean Journal of Operational Research*, vol. 58, no. 3, pp. 414–426, 1992.

[18] R. Z. Farahani and M. Hekmatfar, *Facility Location Concepts, Models, Algorithms and Case Studies*, SpringerVerlag, Berlin Heidelberg, Germany, 2009.

[19] N. Mladenovic, J. Brimberg, P. Hansen, and J. A. Moreno-Perez, "The p-median problem: a survey of metaheuristic approaches," *uropean Journal of Operational Research*, vol. 179, pp. 927–939, 2007.

[20] J. Reese, "Solution methods for thep-median problem: an annotated bibliography," *Networks*, vol. 48, no. 3, pp. 125–142, 2006.

[21] J. Brimberg, Z. Drezner, N. Mladenović, and S. Salhi, "A new local search for continuous location problems," *uropean Journal of Operational Research*, vol. 232, no. 2, pp. 256–265, 2014.

[22] Z. Drezner, J. Brimberg, N. Mladenović, and S. Salhi, "New heuristic algorithms for solving the planar p-median problem," *Computers & Operations Research*, vol. 62, pp. 296–304, 2015.

[23] Z. Drezner, J. Brimberg, N. Mladenović, and S. Salhi, "Solving the planar p-median problem by variable neighborhood and concentric searches," *Journal of Global Optimization*, vol. 63, no. 3, pp. 501–514, 2015.

[24] N. Mishra, D. Oblinger, and L. Pitt, *Sublinear Time Approximate Clustering*, Hewlett-Packard Labs, Palo Alto, CA, USA, 2001.

[25] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley, New York, NY, USA, 1990.

[26] F. Eisenbrand, F. Grandoni, T. Rothvosz, and G. Schafer, "Approximating connected facility location problems via random facility sampling and core detouring," in *Proceedings of SODA'2008*, pp. 1174–1183, ACM, New York, NY, USA, January 2008.

[27] R. Jaiswal, A. Kumar, and S. Sen, "A simple D 2-sampling based PTAS for k-means and other clustering problems," *Algorithmica*, vol. 70, no. 1, pp. 22–46, 2014.

[28] P. Avella, M. Boccia, S. Salerno, and I. Vasilyev, "An aggregation heuristic for large scale p-median problem," *Computers & Operations Research*, vol. 39, no. 7, pp. 1625–1632, 2012.

[29] R. L. Francis, T. J. Lowe, M. B. Rayco, and A. Tamir, "AggregationError for location models: survey and analysis," *Annals of Operations Research*, vol. 167, no. 1, pp. 171–208, 2009.

[30] D. Arthur and S. Vassilvitskii, "k-Means++: the advantages of careful seeding," in *Proceedings of SODA'07*, pp. 1027–1035, SIAM, Diego, CA, USA, January 2007.

[31] P. Hansen and N. Mladenovic, "Variable neighborhood search," in *Search Methodologies*, . K. Burke and G. Kendall, Eds., Springer, Boston, MA, USA, 2005.

[32] I. P. Rozhnov, V. I. Orlov, and L. A. Kazakovtsev, "VNS-based algorithms for the centroid-based clustering problem," *Facta Universitatis Series: Mathematics and Informatics*, vol. 34, no. 5, pp. 957–972, 2019.

[33] S. Still, W. Bialek, and L. Bottou, "Geometric clustering using the information bottleneck method," in *Proceedings of the Advances In Neural Information Processing Systems*, vol. 16MIT Press, Cambridge, UK, December 2004.

[34] Z. Sun, G. Fox, W. Gu, and Z. Li, "A parallel clustering method combined information bottleneck theory and centroid-based clustering," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 452–467, 2014.

[35] C. R. Houck, J. A. Joines, and M. G. Kay, "Comparison of genetic algorithms, random restart and two-opt switching for solving large location-allocation problems," *Computers & Operations Research*, vol. 23, no. 6, pp. 587–596, 1996.

[36] U. Maulik and S. Bandyopadhyay, "Genetic algorithm-based clustering technique," *Pattern Recognition*, vol. 33, no. 9, pp. 1455–1465, 2000.

[37] K. Krishna and M. Narasimha Murty, "Genetic K-means algorithm," *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, vol. 29, no. 3, pp. 433–439, 1999.

[38] M. N. Neema, K. M. Maniruzzaman, and A. Ohgai, "New genetic algorithms based approaches to continuous p-median problem," *Networks and SpatialEconomics*, vol. 11, no. 1, pp. 83–99, 2011.

[39] L. A. Kazakovtsev and I. Rozhnov, "Application of algorithms with variable greedy heuristics for k-medoids problems," *Informatica*, vol. 44, no. 1, pp. 55–61, 2020.

[40] C. M. Hosage and M. F. Goodchild, "Discrete space location-allocation solutions from genetic algorithms," *Annals of Operations Research*, vol. 6, no. 2, pp. 35–46, 1986.

[41] O. Alp, . Erkut, and Z. Drezner, "AnEfficient genetic algorithm for the p-median problem," *Annals of Operations Research*, vol. 122, no. 1/4, pp. 21–42, 2003.

[42] K. Kim and H. Ahn, "A recommender system using GA K-means clustering in an online shopping market," *xpert Systems with Applications*, vol. 34, no. 2, pp. 1200–1209, 2008.

[43] L. A. Kazakovtsev and A. N. Antamoshkin, "Genetic algorithm with fast greedy heuristic for clustering and location problems," *Informatica*, vol. 38, no. 3, pp. 229–240, 2014.

[44] W. Kwedlo and P. Iwanowicz, *Using Genetic Algorithm for Selection of Initial Cluster Centers for the K-Means Method, ICAISC 2010: Artifical Intelligence and Soft Computing*, Springer-Verlag, Berlin, Heidelberg, Germany, 2010.

[45] Z. He and C. Yu, "Clustering stability-basedEvolutionary K-means," *Soft Computing*, vol. 23, no. 1, pp. 305–321, 2019.

[46] C. Pizzuti and N. Procopio, "A K-means based genetic algorithm for data clustering," in *Proceedings of the International Joint Conference SOCO'16-CISIS'16-ICEUTE'16*, vol. 527, pp. 211–222, San Sebastián, Spain, October 2016.

[47] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[48] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, no. 2, pp. 224–227, 1979.

[49] A. V. remeev, "Genetic algorithm with tournament selection as a local search method," *Discrete Analysis and Operations Research*, vol. 19, no. 2, pp. 41–53, 2012.

[50] J. H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, UK, 1992.

[51] D. B. Fogel and J. W. Atmar, "Comparing genetic operators with Gaussian mutations in simulatedEvolutionary processes using linear systems," *Biological Cybernetics*, vol. 63, no. 2, pp. 111–114, 1990.

[52] C. Liu and A. Kroll, "On designing genetic algorithms for solving small- and medium-scale traveling salesman problems," *Swarm andEvolutionary Computation*, vol. 7269, pp. 283–291, 2012.

[53] E. Osaba, R. Carballedo, F. Diaz, E. Onieva, I. de la Iglesia, and A. Perallos, "Crossover versus mutation: a comparative analysis of theEvolutionary strategy of genetic algorithms applied to combinatorial optimization problems," *The Scientific World Journal*, vol. 2014, Article ID 154676, 22 pages, 2014.

[54] J. Walkenhorst and T. Bertram, "Multi-kriterielleoptimierungsverfahren Fur pickup-and-delivery-probleme," in *Proceedings of 21. Workshop Computational Intelligence*, pp. 61–76, Dortmund, Germany, December 2011.

[55] L. A. Kazakovtsev and A. N. Antamoshkin, "Greedy heuristic method for location problems," *Vestnik SibGAU*, vol. 16, no. 2, pp. 317–325, 2015.

[56] D. Q. Zeebaree, H. Haron, A. M. Abdulazeez, and S. R. M. Zeebaree, "Combination of K-means clustering with genetic algorithm: a review," *International Journal of AppliedEngineering Research*, vol. 12, no. 24, pp. 14238–14245, 2017.

[57] . R. Hruschka, R. J. G. B. Campello, A. A. Freitas, and A. C. P. L. F. de Carvalho, "A survey ofEvolutionary algorithms for clustering," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 39, no. 2, pp. 133–155, 2009.

[58] A. A. Freitas, *A Review ofEvolutionary Algorithms for Data Mining, Data Mining and Knowledge Discovery Handbook*, Oxford University, Oxford, UK, 2009.

[59] S. Bandyopadhyay, "Genetic algorithms for clustering and fuzzy clustering," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 6, pp. 524–531, 2011.

[60] P. Larrañaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: a review of representations and operators," *Artificial Intelligence Review*, vol. 13, no. 2, pp. 129–170, 1999.

[61] A. Sarangi, R. Lenka, and S. K. Sarangi, "Design of linear phase firhigh pass filter using PSO with gaussian mutation," in *Proceedings of the Swarm,Evolutionalry, and Memetic Computing*, pp. 471–479, Bhubaneswar, India, July 2015.

[62] D. Deb and K. Deb, "Investigation of mutation schemes in real-parameter genetic algorithms," *Swarm,Evolutionary, and Memetic Computing*, vol. 7677, pp. 1–8, 2012.

[63] K. Deep and M. Thakur, "A new mutation operator for real coded genetic algorithms," *Applied Mathematics and Computation*, vol. 193, no. 1, pp. 211–230, 2007.

[64] K. Deep and H. Mebrahtu, "Combined mutation operators of genetic algorithm for the travelling salesman problem," *International Journal of Combinatorial Optimization Problems and Informatics*, vol. 2, no. 3, pp. 1–23, 2011.

[65] T.-P. Hong, H.-S. Wang, and W.-C. Chen, "Simultaneously applying multiple mutation operators in genetic algorithms," *Journal of Heuristics*, vol. 6, no. 4, pp. 439–455, 2000.

[66] B. McGinley, J. Maher, C. O'Riordan, and F. Morgan, "Maintaining healthy population diversity using adaptive crossover, mutation, and selection," *IEEE Transactions onEvolutionary Computation*, vol. 15, no. 5, pp. 692–714, 2011.

[67] M. Serpell and J.E. Smith, "Self-adaptation of mutation operator and probability for permutation representations in genetic algorithms," *volutionary Computation*, vol. 18, no. 3, pp. 491–514, 2010.

[68] C. A. Brizuela and R. Aceves, *xperimental Genetic Operators Analysis for the Multi-Objective Permutation Flowshop*, Springer-Verlag, Berlin, Heidelberg, Garmany, 2003.

[69] L. Wang and L. Zhang, "Determining optimal combination of genetic operators for flow shop scheduling," *The International Journal of Advanced Manufacturing Technology*, vol. 30, no. 3-4, pp. 302–308, 2006.

[70] B. H. F. Hasan and M. S. M. Saleh, "valuating theEffectiveness of mutation operators on the behavior of genetic algorithms applied to non-deterministic polynomial problems," *Informatica*, vol. 35, no. 4, pp. 513–518, 2011.

[71] P. Karthikeyan, S. Baskar, and A. Alphones, "Improved genetic algorithm using different genetic operator combinations (GOCs) for multicast routing in ad hoc networks," *Soft Computing*, vol. 17, no. 9, pp. 1563–1572, 2013.

[72] . S. Correa, M. T. A. Steiner, A. A. Freitas, and C. Carnieri, "A genetic algorithm for the p-median problem," in *Proceedings of the GECCO-2001*, pp. 1268–1275, Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, 2001.

[73] P. Hansen and N. Mladenović, "J-Means: a new local search heuristic for minimum sum of squares clustering," *Pattern Recognition*, vol. 34, no. 2, pp. 405–413, 2001.

[74] Y. Alkhalifah and R. L. Wainwright, "A genetic algorithm applied to graph problems involving subsets of vertices,"vol. 1, pp. 303–308, in *Proceedings of the 2004 Congress onEvolutionary Computation, Portland*, vol. 1, IEEE, Portland, OR, USA, June 2004.

[75] Y. Lu, S. Lu, F. Fotouhi, Y. Deng, and S. J. Brown, "FGKA: a fast genetic k-means clustering algorithm," in *Proceedings of the 2004 ACM Symposium on Applied Computing-SAC'04*, Nicosia, Cyprus, March 2004.

[76] S. S. Cheng, Y. H. Chao, H. M. Wang, and H. C. Fu, "A prototypes-embedded genetic k-means algorithm," in *Proceedings of the 18th International Conference on Pattern Recognition (ICPR'06)*, pp. 724–727, IEEE, Hong Kong, China, August 2006.

[77] D.-X. Chang, X.-D. Zhang, and C.-W. Zheng, "A genetic algorithm with gene rearrangement for K-means clustering," *Pattern Recognition*, vol. 42, no. 7, pp. 1210–1222, 2009.

[78] D.E. Goldberg and J. Richarson, "Genetic algorithms with sharing for multimodal function optimization," in *Proceedings 2nd International Conference on Genetic Algorithms*, pp. 41–49, Cambridge, UK, October 1987.

[79] K. A. D. Jong, "An analysis of the behaviour of a class of genetic adaptive systems," Ph.D. thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[80] O. Mengshoel and D. Goldberg, *Probabilistic Crowding: Deterministic Crowding with Probabilistic Replacement*, Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, 1999.

[81] I. Ono and S. Kobayashi, *A Real-coded Genetic Algorithm Using the Unimodal Normal Distribution Crossover: Natural Computing Series*, Springer-Verlag, Berlin, Heidelberg, Germany, 2003.

[82] D. Dumitrescu and C. Stoean, "The genetic chromodynamics metaheuristic," in *Proceedings of TELE-INFO'06, WSEAS, Stevens Point*, pp. 92–97, Wisconsin, USA, August 2006.

[83] R. I. Lung and D. Dumitrescu, "Roaming optimization: a newEvolutionary technique for multimodal optimization," *Studia Universitatis Babes-Bolyai - Informatica*, vol. XLIX, no. 1, pp. 99–109, 2004.

[84] M. Zechner and M. Granitzer, "Accelerating K-means on the graphics processor via CUDA," in *Proceedings of the International Conference on Intensive Applications and Services*, pp. 7–15, Valencia, Spain, April 2009.

[85] D. Luebke and G. Humphreys, "How GPUs work," *Computer*, vol. 40, no. 2, pp. 96–100, 2007.

[86] S. N. Sivanandam and S. N. Deepa, *Introduction to Genetic Algorithms*, Springer, Berlin, Germany, 2007.