algorithms

MDPI

Article

Self-Configuring (1 + 1)-Evolutionary Algorithm for the Continuous p-Median Problem with Agglomerative Mutation †

Lev Kazakovtsev *, Ivan Rozhnov and Guzel Shkaberina

Reshetnev Siberian State, University of Science and Technology, Institute of Informatics and Telecommunications, 660037 Krasnoyarsk, Russia; ris2005@mail.ru (I.R.); z_guzel@mail.ru (G.S.)

* Correspondence: levk@bk.ru

† Comparative study of local search in SWAP and agglomerative neighbourhoods for the continuous p-median problem. In Proceedings of the 9th International Workshop on Mathematical Models and their Applications, Krasnoyarsk, Russia, 16–18 November 2020.

Abstract: The continuous p-median problem (CPMP) is one of the most popular and widely used models in location theory that minimizes the sum of distances from known demand points to the sought points called centers or medians. This NP-hard location problem is also useful for clustering (automatic grouping). In this case, sought points are considered as cluster centers. Unlike similar k-means model, p-median clustering is less sensitive to noisy data and appearance of the outliers (separately located demand points that do not belong to any cluster). Local search algorithms including Variable Neighborhood Search as well as evolutionary algorithms demonstrate rather precise results. Various algorithms based on the use of greedy agglomerative procedures are capable of obtaining very accurate results that are difficult to improve on with other methods. The computational complexity of such procedures limits their use for large problems, although computations on massively parallel systems significantly expand their capabilities. In addition, the efficiency of agglomerative procedures is highly dependent on the setting of their parameters. For the majority of practically important p-median problems, one can choose a very efficient algorithm based on the agglomerative procedures. However, the parameters of such algorithms, which ensure their high efficiency, are difficult to predict. We introduce the concept of the $AGGL_r$ neighborhood based on the application of the agglomerative procedure, and investigate the search efficiency in such a neighborhood depending on its parameter r. Using the similarities between local search algorithms and (1 + 1)-evolutionary algorithms, as well as the ability of the latter to adapt their search parameters, we propose a new algorithm based on a greedy agglomerative procedure with the automatically tuned parameter r. Our new algorithm does not require preliminary tuning of the parameter r of the agglomerative procedure, adjusting this parameter online, thus representing a more versatile computational tool. The advantages of the new algorithm are shown experimentally on problems with a data volume of up to 2,000,000 demand points.

Keywords: p-median; (1 + 1)-EA; location problems; evolutionary computation

Citation: Kazakovtsev, L.; Rozhnov, I.; Shkaberina, G. Self-Configuring (1 + 1)-Evolutionary Algorithm for the Continuous p-Median Problem with Agglomerative Mutation. *Algorithms* **2021**, *14*, 130. https://doi.org/10.3390/a14050130

Academic Editor: Eugene Semenkin

Received: 28 March 2021
Accepted: 18 April 2021
Published: 22 April 2021

1. Introduction

1.1. Problem Statement

One of the central problems of location theory is the p-median problem. The goal of the continuous p-median problem is to find p points (centers, medians) such that the sum of the distances from N known points (called demand points or data vectors) to the nearest of the p centers reaches its minimum.

Let us assume that we have N known demand points $A_1, \ldots, A_N$ in a continuous space, $A_i = (a_{i,1}, \ldots, a_{i,d})$, $A_i \in \mathbb{R}^d$, and $S = \{X_1, \ldots, X_p\} \subset \mathbb{R}^d$ is the set of sought points (medians, centers). The objective function (sum of distances) of the p-median problem is [1]:

$$F(X_1, \ldots, X_p) = F(S) = \sum_{i=1}^{N} min_{j=\overline{1,p}} L(A_i, X_j) \rightarrow min_{X_1,\ldots,X_p \in \mathbb{R}^d}. \quad (1)$$

Here, integer p must be known in advance.

The distance metric L $(\cdot,\cdot)$ between objects is the key concept in the location theory. As a rule, a metric is understood as a function or an equation that determines the distance between any points and classes in a metric space [2]. A metric space is a set of points with a defined distance function. The distance of order l between two points is determined by the Minkowski function [3]:

$$L_l(X_j, A_i) = \left(\sum_{k=1}^{d} \left| x_{j,k} - a_{i,k} \right|^l \right)^{\frac{1}{l}}. \quad (2)$$

The parameter l is determined by the researcher. We can use it to progressively increase or decrease the weight of the ith variable. Special cases of the Minkowski function are distinguished by the value of l. For $l = 2$ the function calculates Euclidean metric between two points. By default, location problems use Euclidean metric:

$$L_2(X_j, A_i) = \left(\sum_{k=1}^{d} \left(x_{j,k} - a_{i,k} \right)^2 \right)^{1/2} \quad (3)$$

Here, $X_j = \left(x_{j,1}, \ldots, x_{j,k} \right) \; \forall \; j = \overline{1,p}, k = \overline{1,d}$ are sought centers, also called medians, $A_i = (a_{i,1}, \ldots, a_{i,k}) \; \forall \; i = \overline{1,N}$ are known points called demand points or data vectors.

For $l = 1$, the function calculates Manhattan distance [3]. For $l = +\infty$, the function calculates Tschebychev distance (L_∞ metric), and for $l = 0$, the function calculates Hamming distance [4].

Weber, in his work [5], investigated the problem of finding the center for a set of weighted points (the Weber problem or 1-median problem [5]) which is a special (simplest) case of the problem (1) for $p = 1$. At the same time, the Weber problem is a generalization of a similar Fermat problem [6] with three demand points ($N = 3$, $p = 1$). Weiszfeld, in his work [7], proved a theorem formulated by Sturm [8] and derived a sequence that converged to the optimal solution of the Weber problem. This was a version of the gradient descent algorithm [9] for the Weber problem.

The clustering problem is to divide a given set (sample) of N objects (data vectors) into p disjoint subsets, called clusters, so that each cluster consists of similar objects, and the objects of different clusters differ significantly. In the process of grouping a set of objects into certain groups (subsets), the general features of the object and the applied algorithms play an important role. Some of the automatic grouping problems can be considered from the point of view of location problems. The most popular k-means clustering model [10] can be described by Equation (1) where $L(\cdot,\cdot)$ is the squared Euclidean distance between two points. The existence of a trivial non-iterative solution of the corresponding Weber problem with the squared Euclidean distances [11] makes the k-means problem one of the most popular optimization models for clustering. A disadvantage of this model is its high sensitivity to the outliers (separately located data points), and the p-median model is free from this drawback.

The network p-median problem [12] is to find the p nodes on a network that minimize the sum of the weighted distances from all nodes to the nearest of the p nodes selected as centers (medians). The network and continuous p-median problems are proved to be NP-hard [13,14].

1.2. State-of-the-Art

Weiszfeld proposed an iterative procedure for solving the Weber problem (1-median problem), one of the simplest location problems, based on the iterative weighted least squares method [7]. This algorithm determines a set of weights that are inversely proportional to the distances from the current estimate to the sample points, and creates a new estimate that is the weighted average of the sample according to those weights. The Weiszfeld procedure is very slow if the solution coincides with one of the demand points. Many researchers have developed modifications of the Weiszfeld algorithm [6,15–20] to avoid this disadvantage. The Weiszfeld algorithm can be embedded into more complex algorithms such as the Alternate Location-Allocation (ALA) procedure also called Lloyd's algorithm. This algorithm alternates the allocation step which divides the demand points into p groups in accordance with the number of the closest center and the location step which recalculates the center of each group (i.e., solves the Weber problem for each of groups). The idea of the ALA algorithm is applicable for both network and continuous p-median problems.

Reviews of various solution techniques for p-median problems can be found in [11,21,22].

Hakimi considered the problem of finding the median of a graph (the Weber problem on a network) [12] and generalized this problem to finding the p-medians [23]. The authors of [24–26] proposed a branch-and-bounds algorithm solving the network p-median problem, and in [27,28] the authors considered algorithms based on graph theory for rather small problems. Lagrangian relaxations enables us to obtain an approximate solution to medium-sized problems [29,30].

Many heuristic approaches have been developed for large-scale problems. The simplest approaches consist in local search, in which a set of network nodes adjacent to the nodes of the current intermediate solution is considered as a search neighborhood [31,32]. Drezner et al. [33–36] presented local search approaches for solving continuous p-median location problems including the Variable Neighborhood Search (VNS). Bernabe-Loranca et al. [37] introduce the use of a Hybrid VNS/TABU Algorithm to determine the correct location of p centers. In addition, Drezner et al. [34] proposed heuristic procedures including the genetic algorithm (GA), for rather small datasets.

Modern literature on location methods offers many heuristic approaches [21,38] to setting the initial centers of the ALA procedure, which are mainly various evolutionary and random search methods. A standard local search algorithm starts with some initial solution S and goes to a neighboring solution if this solution turns out to be superior. Moreover, finding the set of neighbor solutions $n(S)$ is the key issue. Elements of this set are formed by applying a certain procedure to a solution S. At each local search step, the neighborhood function $n(S)$ specifies the set of possible search directions [39,40] and determines the efficiency of the algorithm [41].

Local search methods have been developed in metaheuristics. Mladenovich and Hansen [41–44] proposed a search algorithm with alternating neighborhoods (Variable Neighborhood Search, VNS). In works of Kochetov, Mladenovich, and Hansen [41,42,45–48], the authors provide an overview of local search methods based on the idea of alternating neighborhoods, including methods of combining these methods with other metaheuristics which reduce the dependence of the result on the choice of the neighborhood. Flexibility and high efficiency explain the VNS competitiveness in solving NP-hard problems, in particular, for solving automatic grouping and location problems [49,50], multiple Weber problem [51], p-median problem [50,51], and many others. For example, VNS algorithms [35,52] or agglomerative algorithms [53,54] demonstrate good results for the p-median problem. Initialization procedures for local search algorithms may include random seeding or an accurate seeding based on the estimation of the demand points density [38]. However, multiple launches of simple local search algorithms from various randomly generated solutions do not provide a solution to a problem which is close to the global optimum. More advanced algorithms allow us to obtain the objective function values many times better than the local search methods [52].

Evolutionary algorithms are a class of search algorithms that are often used as function optimizers for static objective functions. Using specific evolutionary operators, the evolutionary algorithms recombine and modify a set (population) of candidate solutions to a problem. Various efficient evolutionary algorithms were developed for almost all types of optimization problems including machine learning problems. The use of the genetic and other evolutionary algorithms for solving discrete and continuous location problems [55–60] is also a popular approach. For example, in [61], the authors proposed the online-learning-based reference vector evolutionary many-objective algorithm guided by the reference-vector-based decomposition strategy which employs a learning-based technology to enhance its generalization capability. This strategy employs a learning automaton to obtain the feature of the problem and the searching state according to the feedback information from the environment to adjust the mutation strategy for each sub-problem in different status situations and enhance the optimization performance. The authors use clustering to form groups of reference vectors with the same mutation strategy. The work [62] focuses on a discrete optimization problem of improving the seaside operations at marine container terminals. The authors proposed a new Adaptive Island Evolutionary Algorithm for the berth scheduling problem, aiming to minimize the total weighted service cost of vessels. The developed algorithm simultaneously executes separate evolutionary algorithms in parallel on its "islands" (subpopulations of solutions) and exchanges individuals between the "islands" based on an adaptive mechanism, which enables more efficient exploration of the problem search space. The authors of [63] proposed an alternative many-objective EA, called AnD. In evolutionary many-objective optimization, there exists a problem of selecting some promising individuals from the population. The AnD uses two strategies: angle-based selection and shift-based density estimation, which are employed to remove poor individuals one by one. In a pair of individuals with the minimum vector angle (most similar search directions), one of the individuals is considered as a candidate for elimination. In addition, the EA is a popular solution methodology, showing a promising performance in solving different types of vehicle routing problems [64,65]. In [66], the authors presented an efficient EA developed to solve the mathematical model, which addresses the vehicle routing problem with a "factory-in-a-box". In [67], the authors proposed an approach to distinguish between bacterial and viral meningitis using genetic programming and decision trees able to determine the typology of meningitis for any combination of clinical parameters by achieving 100% of sensitivity.

The EAs are popular for training in various the machine learning methods. In [68], the Salp Swam Algorithm (SSA) was deployed in training the Multilayer Perceptron (MLP) for data classification. The proposed method shows supremacy in results as compared with other evolutionary algorithm-based machine learning problems.

In the case of p-median and similar location problems, evolutionary algorithms recombine the initial solution obtained by the ALA procedure or a local search algorithm. In genetic algorithms, a set (population) of solutions is successively improved with the use special genetic operators (algorithms) of initialization, selection, crossover, and mutation. The crossover operator recombines the elements ("genes") of two parent solution for generating a new ("offspring") solution. One-point and two-point crossover which are standard for many other genetic algorithms have been proved to present drawbacks when applied to problems such as k-means and p-median problems [69] due to generating the offspring solutions too far from their parents. The GAs with the greedy agglomerative crossover operator demonstrates better results [58,70,71]. In this crossover operator, as well as in the IBC algorithms [53,54], the number of centers is successively reduced down to the desired number p. Being originally developed for the network p-median problems, they were adapted for continuous p-median and k-means problems [72]. Metaheuristic approaches, such as genetic algorithms [73], are aimed at finding the global optimum. However, in large-scale instances, such approaches require very significant computational costs, especially if they are adapted to solving continuous problems [58]. The greedy agglomerative procedure is a computationally expensive algorithm, especially in the case

of continuous problems when it includes multiple execution of a local search algorithm. However, this procedure allows us to find solutions that are difficult to improve by other methods without significantly increasing the calculation time.

The concept of the usage of greedy agglomerative procedures as the crossover operator in genetic algorithms is as follows. First, for two selected parent solutions (sets of medians) S_1 and S_2, the algorithm constructs an intermediate solution S' adding r randomly selected medians (centers) from the second parent solution S_2 to the first solution S_1: $S' = S_1 \cup S_2''$. Here, S_2'' is a randomly chosen subset of S_2, $|S_2| = r$. Integer parameter r can be given in advance (often, $r = 1$ or $r = p$). After improving this new intermediate solution with the ALA or local search algorithm, one or more excessive medians are removed until $|S'| = p$. At each iteration, the greedy agglomerative procedure removes such a median that its elimination results in the least significant increase of the objective Function (1).

In [74], the authors systematized approaches to constructing algorithms for searching in neighborhoods (denoted as GREEDY_r) formed using greedy algorithmic procedures for the k-means problem. Searching in SWAP and GREEDY_r neighborhoods has advantages over the simplest Lloyd's procedure [75,76]. However, the results strongly depend on the parameter of the greedy agglomerative procedures, and the optimal values of these parameters differ significantly for test problems. However, the GREEDY_r search outperforms the SWAP search in terms of accuracy. Moreover, such algorithms often outperform more complex genetic algorithms.

Embedding the computationally expensive greedy agglomerative procedures into metaheuristics is even more computationally expensive, which limits the application of such a combination of algorithms to large problems. Nevertheless, the development of massively parallel systems expands the capabilities of such algorithmic combinations [74,77].

The (1 + 1)-evolutionary algorithms or (1 + 1)-EAs are similar with the local search and VNS algorithms but they have no clearly defined neighborhood; therefore, they can reach in one single step any point in the search space [78]. Unlike genetic algorithms, in (1 + 1)-EAs, there is no true population of solutions, and usually, they successively improve a single current solution with the application of the mutation operator to it.

In an (1 + 1)-evolutionary algorithm [78] for pseudo-Boolean optimization problems, the bitwise mutation operator flips each bit independently of the others with some probability p_m that depends on the length of the bit string. The current bit string is replaced by the new one if the fitness of the current bit string is not superior to the fitness of the new string.

In their work, Borisovsky and Eremeev made the study on performance of the (1 + 1)–EA [79] and compared other evolutionary algorithms to the (1 + 1)–EA [80]. In [80], the authors studied the conditions under which (1 + 1)-EA is competitive with other evolutionary algorithms (EAs) in terms of the distribution of the fitness function at a given iteration and relative to the average optimization time. Their approach is applicable when the (1 + 1)-EA mutation operator prevails in the reproduction operator of the evolutionary algorithm. In this case, the lower bounds obtained for the expected optimization time of (1 + 1)-EA can be extended to other EAs based on the dominant operator. They proved that under the domination condition it is an optimal search technique with respect to the probability of finding solutions of sufficient quality after a given number of iterations. In the case of domination, the (1 + 1)-EA is also preferable with respect to the expected fitness at any iteration and the expected optimization time.

Reference [81] considers the scenario of the (1 + 1)-EA and randomized local search (RLS) with memory. The authors present two new algorithms: (1 + 1)-EA-m (with a raw list and hashtable option) and RLS-m+ (and RLS-m if the function is known in advance to be unimodal). These algorithms can be regarded as very simple forms of tabu search. Empirical results, with a reasonable fitness evaluation time assumption, verify that (1 + 1)-EA-m and RLS-m+ are superior to their conventional counterparts. In paper [82], the authors investigate the (1 + 1)-EA for optimizing functions over the space $\{0, ..., r\}^n$, where r is a positive integer. They show that for linear functions over $\{0, 1, 2\}^n$, the ex-

pected runtime of this algorithm is $O(n \log n)$. This result generalizes an existing result on pseudo-Boolean functions and is derived using drift analysis. They also show that for large values of r, no upper bound for the runtime of the (1 + 1)-EA for linear function on $\{0, ..., r\}^n$ can be obtained with this approach nor with any other approach based on drift analysis with weight-independent linear potential functions.

In [83], the authors investigate the performance of the (1 + 1)-EA, on the maximum independent set problem (MISP) from a theoretical point of view. They showed that the (1 + 1)-EA can obtain an approximation ratio of $(\Delta + 1)/2$ on this problem in expected time $O(n^4)$, where Δ and n denote the maximum vertex degree and the number of nodes in a graph, respectively. They reveal that the (1 + 1)-EA has better a performance than the local search algorithm on an instance of MISP and demonstrate that the local search algorithm with 3-flip neighborhood will be trapped in local optimum while the (1 + 1)-EA can find the global optimum in expected running time $O(n^4)$.

In [84], the authors theoretically investigate the approximation performance of the (1 + 1)-EA, on the minimum degree spanning tree (MDST) problem which is a classical NP-hard optimization problem and show its capability of obtaining an approximate solution for the MDST problem with a limited maximum degree in expected polynomial time. In [85], the authors analyze the expected runtime of the (1 + 1)-EA solving robust linear optimization problems (i.e., linear problems under robust scenarios) with a cardinality constraint. They consider two common robust scenarios, i.e., deletion-robust and worst-case, and disclose the potential of (1 + 1)-EAs for robust optimization.

An interesting approach was proposed in [86]. The authors propose a (1 + 1)-fast evolutionary algorithm ((1 + 1)-FEA) with an original approach to the adjustment of the mutation rate. This algorithm uses a random variable $\lambda = \in \left\{\overline{1, n}\right\}$ which takes its values in accordance with some distribution D. The mutation operator in this (1 + 1)-FEA generates an instance λ' of λ in accordance with this distribution, applies the mutation with the mutation rate λ' to the current solution (replacing λ' bits with random values), and, if the new obtained solution improves the objective function value, then it replaces the current solution with the new one and corrects the probability distribution D so that the probability of generating the current value λ' increases. The worst case estimation of (1 + 1)-FEA for any black box function is significantly smaller than that for the original (1 + 1)-EA with a random mutation rate.

1.3. Research Gap and Our Contribution

Many important practical problems require obtaining a solution to a problem with a minimum error. For the p-median problem, such cases may include optimal location problems with a high cost of error, determined, for example, by the cost of transportation between sites. If the p-median problem is considered as a mathematical statement for clustering problems, it may be necessary to obtain a solution that would be difficult to improve by known methods without a multiple increase in the computation time. When comparing the accuracy of various algorithms, we need a reference solution which is not necessarily the exact global optimum of the problem, but at the same time is the best known solution. In this work, aimed at obtaining the most accurate solutions, we understand the accuracy of the algorithm exclusively as the achieved value of the objective function.

Algorithms based on the use of greedy agglomerative procedures, despite their computational costs without guaranteeing an exact result or even a result with a predetermined accuracy, enable one to obtain solutions to practical p-median problems that are difficult to improve by other known methods in comparable time [71,77]. Such algorithms are extremely sensitive to the parameter r of these procedures which determines the number of centers (medians) added to the intermediate solution [77]. In the field of (1 + 1)-evolutionary algorithms for pseudo-Boolean optimization problems, there are known approaches that can adjust a similar mutation parameter that determines the number of replaced bits in the current solution.

The work hypotheses of this article are as follows:

1. The efficiency of the greedy agglomerative procedure applied to successive improvement of the p-median problem solution embedded into a more complex algorithm, such as evolutionary algorithm, highly depends on its parameter r (a number of excessive centers to be eliminated), and this dependence is hardly predictable.
2. The principle of adjusting the numerical parameter of the mutation operator by changing the probability distribution of its values, which is used in (1 + 1)-evolutionary algorithms with 0–1 coding of solutions for pseudo-Boolean optimization problems, can also be effectively applied to adjust the numerical parameter of the agglomerative mutation operator based on the use of agglomerative procedures in an evolutionary algorithm with real coding of solutions when solving the p-median problems.

We propose a new algorithm based on the ideas of known search algorithms with greedy agglomerative procedures and (1 + 1)-evolutionary algorithms which successively improves the current solution with the use of greedy agglomerative procedures with the randomly chosen value of parameter r and simultaneously adjusts the probability distribution of parameter r in such procedures.

This article is an extended version of the paper presented at the International Workshop on Mathematical Models and their Applications (IWMMA'2020, 16–18 November 2020, Krasnoyarsk, Russia) [79].

The rest of this article is organized as follows. In Section 2, we describe known algorithms for the p-median problem including the algorithms with greedy agglomerative procedures. We investigate the dependence of their efficiency on parameter r. We propose the a massive parallel (CUDA) version for the greedy agglomerative procedures. In addition, using the similarities between local search algorithms and (1 + 1)-evolutionary algorithms, as well as the ability of the latter to adapt their search parameters, we propose a new algorithm based on the use of greedy agglomerative procedures with the automatically tuned parameter r. In Section 3, we describe the results of our computational experiments on various datasets up to 2 million demand points. Experiments demonstrate the advantage of our new algorithm in comparison with known ones. In Section 4, we discuss the applicability of the new algorithm to practical problems and propose possible directions for the further development of such algorithms. In Section 5, we give a short conclusion.

2. Methods

2.1. The Basic Algorithmic Approaches

Based on the algorithm proposed by Lloyd [75] for discrete p-median problems on a network, Alternate Location-Allocation (ALA) procedure [87,88] also known as the standard k-means procedure [76] the most popular algorithm for the k-means and p-median problems. This simple algorithm can be considered as a special case of similar Expectation Maximization (EM) procedure [89–92]. The ALA procedure sequentially improves an intermediate solution, enabling us to find a local minimum of the objective Function (1).

In terms of continuous optimization, the ALA procedure is not a true local search algorithm since it searches for a new solution not necessarily in the ε-neighborhood of the existing solution.

However, it usually outperforms simpler local optimization methods: gradient or coordinate descent, etc.

In the case of a p-median problem, Algorithm 1 solves the Weber problem (i.e., center search problem or a 1-median problem) for each cluster. The iterative Weiszfeld procedure gives an approximate solution to the Weber problem with given accuracy ε_1. At each subsequent iteration, the coordinates of the solution (center) X' of a subset $S' = \{A'_1, \ldots, A'_{N'}\} \subset \{A_1, \ldots, A_N\}$ of N' demand points are calculated from the previous solution X as follows:

$$X' \leftarrow \sum_{i=1}^{N'} \frac{A'_i}{L(X, A'_i)} \Big/ \sum_{i=1}^{N'} \frac{1}{L(X, A'_i)} \tag{4}$$

Algorithm 1 ALA (): Alternate Location-Allocation (Lloyd's procedure)

Require: Set S of p initial centers $S = \{X_1, \ldots, X_p\}$.
1. For each center X_i, $i = \overline{1,p}$, define its cluster $C_i \subset \{A_1, \ldots, A_N\}$ as a subset of demand points having the same closest center X_i.
2. For each cluster C_i, $i = \overline{1,p}$, recalculate its center X_i, i.e., solve the Weber problem:

$$X_i \leftarrow \text{argmin}_X \sum_{Y \in C_i} L(X,Y)$$

3. Repeat from Step 1 if Steps 1, 2 made any changes.

These calculations are repeated until $L(X, X') < \varepsilon_1$. Usually, the Weiszfeld procedure converges quickly. For the majority of problems, no more than 50 iterations are required in order to reach the accuracy limits of the double data type except include the cases where the solution coincides with one of the demand points. Such situations are easily bypassed:

$$X' \leftarrow \frac{\sum_{i \in \{\overline{1,N'}\}, L(X,A'_i) < \varepsilon_2} \left[\frac{A'_i}{L(X,A'_i)} \right]}{\sum_{i \in \{\overline{1,N'}\}, L(X,A'_i) < \varepsilon_2} \left[\frac{1}{L(X,A'_i)} \right]} \tag{5}$$

Here, $\varepsilon_2 < \varepsilon_1$ is a minimum distance at which two points are considered different.

Embedding the Weiszfeld iterative procedure in an iterative ALA algorithm, which, in turn, is embedded in more complex algorithms, makes such algorithmic constructions difficult to use effectively with large-scale problems. Steps 1 and 2 of the ALA algorithm are aimed at the reduction of the objective function. Knowing the coordinates of the centers X_i, without changing their coordinates, Step 1 redistributes the demand points $A_i, \ldots, A_N$ among the centers (i.e., forms clusters of demand points around the centers) so that the sum of the distances from each of the demand points to the nearest center becomes minimal. Step 1 solves a simplest simple discrete optimization problem. Step 2 solves a series of continuous optimization problems (Weber problems) which minimize the total distance from the center X_i to the demand points $A'_1, \ldots, A'_{N'}$ of its cluster. Each iteration of the Weiszfeld procedure (4) is also aimed at gradual improvement of the current solution X. For a gradual improvement of solutions, the Weber problem does not have to be solved to a given accuracy ε_1 at each iteration of the ALA algorithm. Thus, Steps 2 and 3 can take the following form:

Step 2: For each cluster C_i, $i = \overline{1,p}$, correct its center X_i with an iteration of Weiszfeld procedure:

$$X_i \leftarrow \frac{\sum_{Y \in C_i, L(X,Y) < \varepsilon_2} \left[\frac{Y}{L(X_i,Y)} \right]}{\sum_{Y \in C_i, L(X,Y) < \varepsilon_2} \left[\frac{1}{L(X_i,Y)} \right]} \tag{6}$$

Step 3: Repeat from Step 1 if Steps 1 and 2 moved at least one of centers X_i by a distance exceeding ε_1.

The essence of the VNS [42] is that for some intermediate solution, we determine a set of neighborhoods of this solution. From this set, the next type of neighborhood is selected, and we apply the corresponding local search algorithm for searching in this solution. If this algorithm finds an improved solution, the intermediate solution is replaced by this new solution, and the search continues in the same neighborhood. If the next local search algorithm cannot improve the solution, a new search neighborhood is selected from the set of neighborhoods. The stop criterion is the time limitation.

Several algorithms use only two types of neighborhoods: the first is ε-neighborhood (i.e., at this step, the problem of continuous optimization is solved). For example, the search in SWAP neighborhood is a popular method for solving p-median and k-means problems. The j-means [93] algorithm using these neighborhoods is one of the most accurate methods for solving such problems. The essence of the search in SWAP neighborhoods is as follows.

Let $S = \{X_1, \ldots, X_p\}$ be the local optimum of the p-median problem in ε-neighborhood (such a solution can be obtained by the ALA algorithm or other algorithms such as gradient descent). If the value of the objective function has improved due to this replacement, then the search continues in the ε-neighborhood of the new obtained solution (the problem of continuous optimization is solved again).

The search algorithm makes an attempt to replace some of the medians $X_1, \ldots, X_p$ with one of the data vectors $A_1, \ldots A_N$. Search in the SWAP neighborhoods can be regular (all possible replacements are enumerated) or randomized (the medians and demand points are randomly picked for the replacement).

Let us denote by SWAP$_r$ a neighborhood (set of solutions) of a current solution obtained by the replacement of r medians by demand points.

In our computational experiments (described in detail in Section 3) on various problems from repositories [94,95], we investigated the dependence of the efficiency (ability to reach the minimum values of the objective functions) of the search in SWAP$_r$ neighborhood on the parameter r. The results obtained after the fixed runtime are given in Figure 1. Obviously, these results are highly dependent on r.

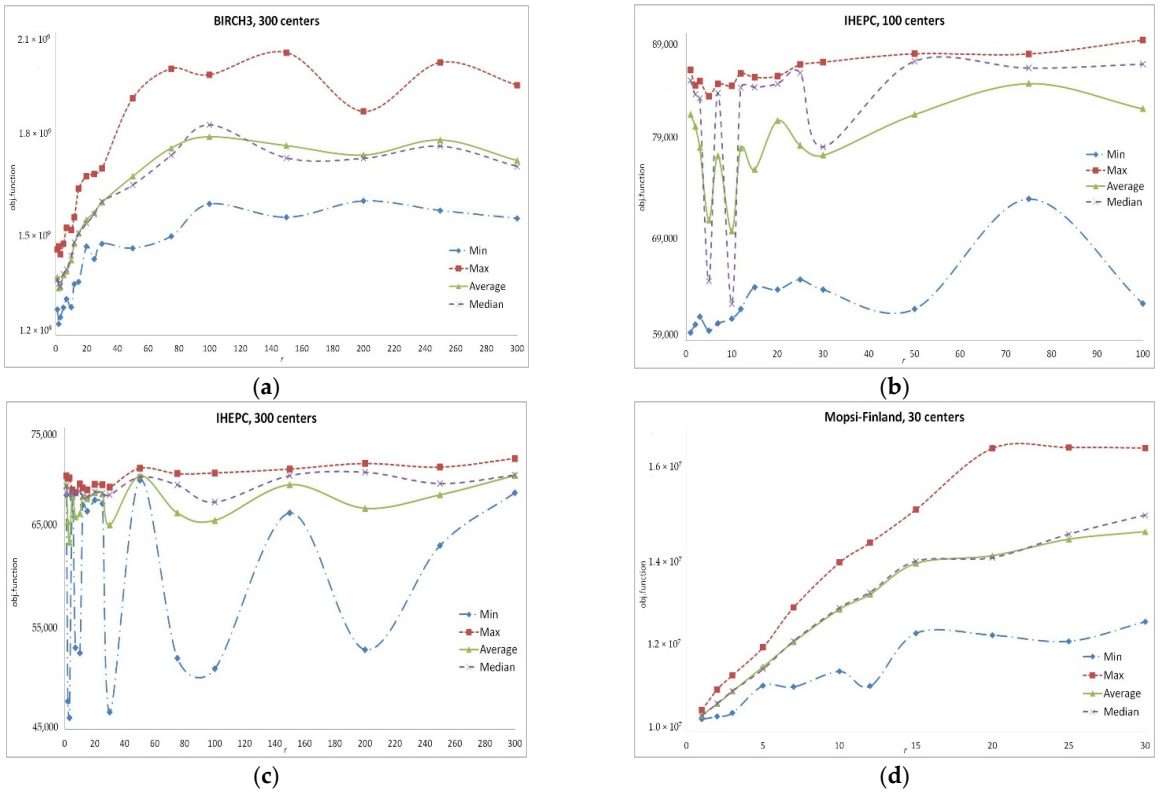

Figure 1. Comparative results of the search in SWAPr neighborhoods. Dependence of the result (1) on r: (**a**) BIRCH3 dataset, search for 300 centers, 100,000 data vectors, time limitation 10 s; (**b**,**c**) IHEPC dataset, search for 100 and 300 centers, respectively, 2,075,259 data vectors, time limitation 300 s; (**d**) Mopsi-Finland dataset, search for 30 centers, 13,467 data vectors, time limitation 5 s.

2.2. Greedy Agglomerative Procedures

The agglomerative approach to solving the p-median problem is often successful [70]. To solve the p-median problem in a continuous space, the authors of [60] use genetic algorithms with various crossover operators based on greedy agglomerative procedures. Alp, Erkut, and Drezner in [70] presented a genetic algorithm for facility location problems, where evolution is facilitated by a greedy agglomerative heuristic procedure. A genetic

algorithm with a faster greedy heuristic procedure for clustering and location problems was also proposed in [71].

Greedy agglomerative procedures can be used as independent algorithms or embedded in VNS or evolutionary algorithms [52,96]. Such procedures can be described as follows in Algorithm 2:

Algorithm 2 BasicAggl (S)

Require: Set of initial centroids $S = \{X1, \ldots, X_K\}$, $|S| = K > k$, required final number of centroids k.
Improve S with the two-step local search algorithm if possible;
while $|S| > k$ **do**
  for $i = \overline{1, |S|}$ **do**
    $F_i \leftarrow SSE(S \backslash \{X_i\})$;
  end for;
  Select a subset $S' \subset S$ of to remove centroids with the minimum values of the corresponding variables F_i; // By default, $r_{toremove} = 1$;
  Improve this new solution $S \leftarrow (S \backslash S')$; with the two-step local search algorithm;
end while.

To improve the performance of such a procedure, the number of simultaneously eliminated centers can be calculated as $r_{toremove} = \max\left\{1, \left\lceil (|S| - p) \cdot r_{coef} \right\rceil\right\}$. In [52,97], the authors used the elimination coefficient value $r_{coef} = 0.2$. This means that at each iteration, up to 20% of the excessive centers are eliminated, and such values are proved to make the algorithm faster.

The agglomerative procedure for obtaining an $AGGL_r$ neighborhood can be defined as follows in Algorithm 3:

Algorithm 3 AGGLr (S, S_2)

Require: Two sets of centers S, S_2, $|S| = |S_2| = p$, the number of centers r of the solution S_2 which are used to obtain the resulting solution, $r \in \{\overline{1, p}\}$.
for $i = \overline{1, n_{repeats}}$ **do**
  1. Select a subset $S' \subset S_2 : |S'| = r$;
  2. $S' \leftarrow BasicGreedy(S \cup S')$;
  3. **if** $F(S') < F(S)$ **then**
    $S \leftarrow S'$
  end if;
end for;
return S.

Such procedures use various values of $r \in \{\overline{1, p}\}$, and $n_{repeats}$ depends on r: $n_{repeats} = \max\{1, [p/r]\}$.

If the solution S_2 is fixed, then all possible results of applying the $AGGL_r$ (S, S_2) procedure form a neighborhood of the solution S, and S_2 as well as r are parameters of such a neighborhood. If S_2 is a randomly chosen locally optimal solution obtained by ALA ($S_{2'}$) procedure applied to a randomly chosen subset $S_2' \subset \{A_1, \ldots, A_N\}$, $|S_2'| = p$, then we deal with a randomized neighborhood.

Let us denote such a neighborhood by $AGGL_r$ (S). Our experiments demonstrate that the obtained result of the local search in $AGGL_r$ neighborhoods strongly depends on r (see Figure 2).

As mentioned above, the greedy agglomerative procedure $AGGL_r$ can be used as the crossover operator of the genetic algorithms. Algorithms proposed in [52,71] use such procedures with $r = 1$ and $r = p$ for solving the k-means and p-median problems.

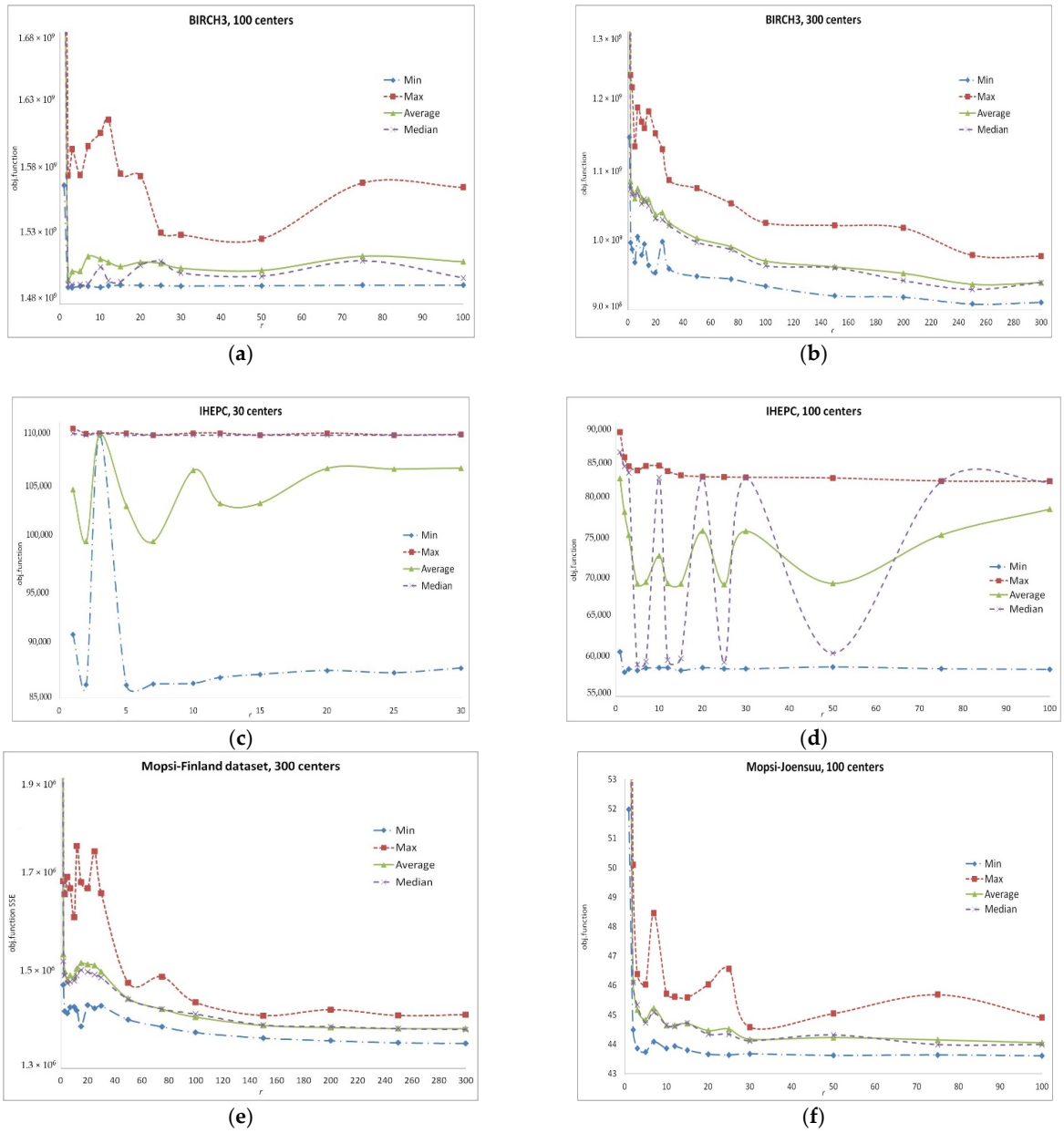

Figure 2. Comparative results of the search in AGGL*r* neighborhoods. Dependence of the result (1) on *r*: (**a**,**b**) BIRCH3 dataset, search for 100 and 300 centers, respectively, 100,000 data vectors, time limitation 10 s; (**c**,**d**) IHEPC dataset, search for 30 and 100 centers, respectively, 2,075,259 data vectors, time limitation 300 s; (**e**) Mopsi-Finland dataset, search for 300 centers, 13,467 data vectors, time limitation 5 s; (**f**) Mopsi-Joensuu dataset, search for 100 centers, 6014 data vectors, time limitation 5 s.

As a rule, the VNS algorithms move from neighborhoods of lower cardinality to wider neighborhoods. For instance, in [52], the authors propose a sequential search in the neighborhoods $\mathrm{AGGL}_1 \rightarrow \mathrm{AGGL}_{\mathrm{random}} \rightarrow \mathrm{AGGL}_p \rightarrow \mathrm{AGGL}_1 \rightarrow \ldots$ Here, $\mathrm{AGGL}_{\mathrm{random}}$ is a neighborhood with randomly selected $r \in \{\overline{2, p-1}\}$. In this case, the initial neighborhood type has a strong influence on the result [52]. Figure 1 shows that the dependence of the obtained objective function value on r is complex. For the specific problems, there may exist specific values of the parameter r which enable us to obtain better results which are hardly predictable. Nevertheless, a search algorithm may collect the information of the most efficient values of r during its runtime.

2.3. CUDA Implementation of Greedy Agglomerative Procedures

Compute Unified Device Architecture (CUDA) is a hardware-software parallel computing architecture that can significantly increase computational performance [98] designed for the effective use of non-graphical computing on the graphics processing units (GPUs). A GPU is a set of multiprocessors that simultaneously execute parallel threads grouped into data blocks. The threads execute the same instructions on different data in parallel. Communication and synchronization in blocks is impossible.

A thread is assigned an identifier (*threadIdx.x*) depending on its position in the block, and a block identifier (*blockIdx.x*) is also assigned to a thread depending on its position in the grid. The thread and block identifiers are available at run time, which allows us to set specific memory access patterns. Each thread in the GPU performs the same procedure, known as the kernel [98].

CUDA versions of the algorithms for solving discrete p-median problems [99–101] (which actually solve rather small discrete problems from the OR (Operation Research) Library) as well as parallel implementations of the ALA algorithm for the k-means problems are described in the modern literature [74,98,102]. In our research, we take into account the specificity of the continuous problems.

For large amounts of data, calculating the distances $L(X_i, A_j)$ from demand points to cluster centers is the most computationally expensive part of the ALA algorithm. The distances should be calculated to all centers, since the algorithm does not know in advance which center is the nearest one. Traditionally used in solving the k-means problem, this approach can be accelerated by using the triangle inequality [103,104] in the case of p-median problems: if the displacement of the ith center relative to the previous iteration was $\Delta X_i = L(X_i, X_i')$, and its distance to the jth demand point was $L_{ij} = L(X_i, A_j)$, then the new distance lies within $L_{ij} \pm \Delta X_i$. If $\exists i^* \in \{\overline{1, p}\}$:

$$L_{i^*j} + \Delta X_{i^*} < L_{ij} - \Delta X_i, \tag{7}$$

such that for new distances, $L(X_{i^*}', A_j) < L(X_i', A_j)$. Knowing all distances L_{ij} from the previous iteration and the shift of the centers X_i , we can avoid calculating a significant part of the distances at each iteration, which is especially important when processing multidimensional data. Nevertheless, for large-scale problems, matrix $\{L_{ij}\}$ requires a significant additional amount of memory (pN values). Moreover, checking the condition (7) for each pair (A_j, X_i) which requires significantly less computational resources than distance calculation

$$L_{ij} = L(A_j, X_i) = \sqrt{\sum_{l=1}^{d} \left(a_{j,l} - x_{i,l}\right)^2} \tag{8}$$

will not accelerate the execution of the algorithm in the case of its CUDA implementation, which assumes strictly simultaneous execution of all instructions by all threads. Therefore, in this work, we do not use of the triangle inequality to accelerate the CUDA version of the ALA algorithm, although more complex implementations for CUDA could probably take advantage of the triangle inequality.

Note that the distances (8) are calculated both in Step 1 of Algorithm 1, and in its Step 2 which includes the iteration of the Weiszfeld procedure (4).

We used the following CUDA implementation of Step 1 of Algorithm 1. The step is divided into two parts. First, variables common for all threads are initialized Algorithm 4:

Algorithm 4 CUDA implementation of Step 1 in the ALA algorithm, part 1 (initialization)

$X'_j \leftarrow 0 \forall j = \overline{1,p}$ // Here, X'_j are vectors used for recalculation of centers.
$counter_j \leftarrow 0 \forall j = \overline{1,p}$ // Counters of objects assigned to each of centers.
$D_j \leftarrow 0 \forall j = \overline{1,p}$ // D_j are used to calculate $\sum_{j \in C_j, L(A_j,X_i) > \varepsilon_2} \frac{1}{L(X_i,A_j)}$.
$D'_j \leftarrow 0 \forall j = \overline{1,p}$ // d-dimensional vectors to calculate $\sum_{j \in C_j, L(A_j,X_i) > \varepsilon_2} \frac{A_j}{L(X_i,A_j)}$.

For the rest of the CUDA implementation of Step 1 of the ALA algorithm, we used the number of threads $N_{trreads} = 512$ for each CUDA block. The number of blocks is calculated as

$$N_{blocks} = (N + N_{threads} - 1)/N_{threads}. \quad (9)$$

Thus, each thread processes a single demand point in Algorithm 5:

Algorithm 5 CUDA implementation of Step 1 in the ALA algorithm

$i \leftarrow blockIdx.x\ timesblockDim.x + threadIdx.x$
if $i > N$ **then**
return
end if
$j' \leftarrow \text{argmin}_j \parallel A_j - X_i \parallel^2$ // number of a center for the ith data point
$d' \leftarrow \min_j \parallel A_j - X_i \parallel^2$; $C_i \leftarrow j'$ // Assign A_i to center j'.
$D \leftarrow (\sqrt{d'})^{-1}$; $D_j \leftarrow D_j + D$; $D'_j \leftarrow D'_j + A_j D$; $counter_{j'} \leftarrow counter_{j'} + 1$;
Synchronize threads.

The parallel implementation of Step 2 of the ALA algorithm (see Algorithm 6) uses $N_{threads} = 512$ threads for each CUDA block. The number of blocks is $N_{blocks2} = (p + N_{threads} - 1)/N_{threads}$. The changed variable, common to all threads, is initialized to 0.

Algorithm 6 CUDA implementation of Step 2 in the ALA algorithm

$j \leftarrow blockIdx.x \times blockDim.x + threadIdx.x$
if $j > k$ **then**
return;
end if;
$X'_j \leftarrow \frac{D'_j}{A_j}$;
if $L\left(X'_j, X_j\right) \geqslant \varepsilon_2$ **then**
$changed \leftarrow 1$;
end if;
$X_j \leftarrow X'_j$;
Synchronize threads.

Variable changed is used at the last step of the ALA algorithm and signals the need to continue the iterations.

In addition, we implemented Step 3 of BasicAggl algorithm on the GPU. At this step, Algorithm 2 calculates the sum of the distances after removing a single center with index i': $F'_{i'} = F(S \backslash \{X_{i'}\})$. Knowing the objective function value $F(S)$ for set S of centers, we can calculate its new value

$$F'_{i'} \leftarrow F(S \backslash \{X_{i'}\}) = F(S) + \sum_{l=1}^{N} \Delta D_l \quad (10)$$

where

$$\Delta D_l = \begin{cases} 0, & C_{i'} \neq l, \\ \left(\min\limits_{j \in \{\overline{1,p}\},\, j \neq i'} L\left(A_j, X_{C_j}\right) \right) - L\left(A_j, X_l\right), & C_{i'} = l. \end{cases} \quad (11)$$

Here, $C_{i'}$ is the center number which $A_{i'}$ is assigned to (nearest to $X_{i'}$). We also used 512 threads per block, the number of blocks is calculated according to (9).

After initialization of common variable $Dsum \leftarrow 0$, the Algorithm 7 is started in a separate thread for each data point:

Algorithm 7 CUDA implementation of Step 3 in Algorithm 2

$l \leftarrow blockIdx.x\ timesblockDim.x + threadIdx.x$;
if $l > k$ **then**
return;
end if;
Calculate ΔD_l in accordance with (2.8);
if $\Delta D_l > 0$ **then**
$D_{sum} \leftarrow D_{sum} + \Delta D_l$;
end if;
Synchronize threads.

After parallel running this algorithm, F_i is calculated in accordance with (10) on Step 3 of Algorithm 2: $F'_{i'} \leftarrow F(S) + D_{sum}$.

The remaining parts of our GA are implemented by the central processor unit. The remaining parts performing the selection do not have any significant effect on the calculation speed. This CUDA implementation of the ALA algorithm enables us to use both the ALA algorithm and more complex algorithms that include it, on large amounts of data, up to millions of demand points.

2.4. New Algorithm

As in local search algorithms, in accordance with the general idea of (1 + 1)-EAs, our new algorithm successively improves the single intermediate solution. To improve it, the AGGL*r* procedure is applied to it (see Algorithm 3) with an additional intermediate solution randomly generated as the second parameter S_2, which is improved by the ALA procedure. A similar idea was applied for the k-means problem in VNS algorithms with randomized neighborhoods [54], which we included in the list of algorithms for comparison. The key feature of the new algorithm is that it does not fix values of the r parameter nor does it generate r values with equal probability, but it adjusts the values of the generation probabilities for each of the possible r values in accordance with the results obtained. If the use of the AGGL*r*—procedure has resulted in an improvement in the value of the objective function, our algorithm increases the probability P_r for the used r and for values close to it.

The new algorithm can be described as follows in Algorithm 8.

Algorithm 8 Aggl-EA ()

Randomly select a subset $S \subset \{A_1, \ldots, A_N\}$; $S \leftarrow ALA(S)$; assign $P_i \leftarrow 1/p\ \forall i = \overline{1,p}$;
repeat
randomly select a subset $S_2 \subset \{A_1, \ldots, A_N\}$; $S_2 \leftarrow ALA(S_2)$;
in proportion to the values of the probabilities Pi, $i = \overline{1,p}$, choose the value of r $\subset \{\overline{1,p}\}$;
$S_3 \leftarrow \mathrm{AGGL}_r(S, S_2)$;
if $F(S_3) < F(S)$ **then**
$P_i \leftarrow k_1 \cdot P_i\ \forall i = \overline{\left\lceil \frac{r}{k_2} \right\rceil, \min\{p, k_2 \cdot r\}}$; // *we used* $k_1 = 1.1$ *and* $k_2 = 1.5$.
$P_i \leftarrow \frac{P_i}{\sum_{i=1}^{p} P_i} \forall i = \overline{1,p}$; *// probability normalization*
$S \leftarrow S_3$;
end if;
until time limitation is reached.
return S.

3. Results of Computational Experiments

In all our experiments, we used the classic datasets from the UCI Machine Learning and Clustering basic benchmark repositories [94,95,105]: (a) Individual Household Electric Power Consumption (IHEPC)—energy consumption data of households during several years, more than 2 million demand points (data vectors) in $\mathbb{R}^2$, 0–1 normalized data, "date" and "time" columns removed; (b) BIRCH3: 100 groups of points of random size, 100,000 demand points in $\mathbb{R}^2$; (c,d) S1 and S4 datasets, respectively: Gaussian clusters with cluster overlap (5000 demand points in $\mathbb{R}^2$); (e) Mopsi-Joensuu: geographic locations of users (6014 data vectors, 2 dimensions) in Joensuu city; (f) Mopsi-Finland: geographic locations of users (13,467 data vectors, 2 dimensions) in Finland.

For our computational experiments, we used the following test system: Intel Core 2 Duo E8400 CPU, 16GB RAM, NVIDIA GeForce GTX1050ti GPU with 4096 MB RAM, floating-point performance 2138 GFLOPS. This choice of the GPU hardware was made due to its prevalence, and also one of the best values of the price/performance ratio. The program code was written in C++. We used Visual C++ 2017 compiler embedded into Visual Studio v.15.9.5, NVIDIA CUDA 10.0 Wizards, and NVIDIA Nsight Visual Studio Edition CUDA Support v.6.0.0. For all datasets, 30 attempts were made to run each of algorithms (Algorithms 1–6).

We examined the following algorithms: (a) Lloyd: the ALA algorithm in the multi-start mode; (b) j-means: j-means algorithm (regular search in $SWAP_1$ neighborhood in combination with the ALA algorithm) in the multi-start mode; (c) $AGGL_r$: randomized search in the $AGGL_r$ neighborhood, $r = \overline{1, p}$; (d) $SWAP_r$: local search in $SWAP_r$ neighborhoods, $r = \overline{1, p}$ (only the best result for $r = \overline{1, p}$ is given); (e–g) GH-VNS1, GH-VNS2, GH-VNS3: Variable Neighborhood Search algorithms with neighborhoods formed by application of $AGGL_r()$ procedure, see [52]; (h) GA-1POINT: genetic algorithm with a standard 1-point crossover; (i) GA-UNIFORM: genetic algorithm with a standard 1-point crossover and uniform random mutation [56]; (j) Aggl-EA: our new algorithm.

For all datasets, 30 attempts were made to run each of the algorithms (see Tables 1 and A1, Tables A2–A15 in Appendix A). In Table 1, we present the results of our new algorithm Aggl-EA() and the best of listed known algorithms, i.e., known algorithms which provided the best average or median values of the objective function (1) after 30 runs.

For the smallest and simplest test problems (see Tables A1 and A4 in Appendix A), several algorithms including Aggl-EA() and search in the *AGGL* neighborhoods resulted in the same objective function value (standard deviation of the result is 0.0) which is probably the global minimum. Nevertheless, the Aggl-EA() algorithm outperforms both genetic algorithms and simplest local search algorithms such as Lloyd() or j-means which do not reach this minimum value in all attempts. For more complex test problems, the comparative efficiency varies. However, there were no test problems for which our new algorithm demonstrated a statistically significant disadvantage in comparison with the best of known algorithms.

The best (minimum), worst (maximum), average, and median values of the achieved objective function were averaged after 30 runs of each algorithm with fixed time limitation. The best average and median values of the objective Function (1) are underlined. We compared our new Aggl-EA () algorithm with known examined algorithm having the best median or average results. The significance of advantage/disadvantage of Aggl-EA() algorithm was estimated with the *t*-test [106,107] and non-parametric Wilcoxon rank sum test [108,109].

In the analysis of algorithm efficiency, the researchers often estimate the computational expenses as the number of the objective function calculations. The ALA procedure is the most computationally expensive part of all examined algorithms. In the ALA algorithm, the objective Function (1) is never calculated directly. The program code profiling results show that Step 2 of Algorithm 1 and Step 3 of Algorithm 2 occupy more than 99% of computational resources (processor time). These steps were implemented on GPUs. Moreover, we use the same implementation of the ALA procedure and BasicAggl() algorithm for all examined algorithms. Thus, the consumed time could be proportional with the number of iterations of the ALA algorithm (its Step 2) and the most time-consuming iterations of Algorithm 2 (its Step 3). However, the time consumption of these iterations depends on the number of centers which successively decreases during the work of the BasicAggl () algorithm. Thus, for the p-median problems, we used the astronomical time as the most informative unit for the computational expenses.

Table 1. Comparative results for all datasets (best of known algorithms vs. new algorithm).

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	**Max (Worst)**	**Average**	**Median**	**Std. Dev**
BIRCH3 dataset. 100,000 data vectors in $\mathbb{R}^2$, k = 30 clusters, time limitation 10 s					
GH-VNS1	3.45057×10^9	3.45057×10^9	3.45057×10^9	3.45057×10^9	0.00000
Aggl-EA↔⇔	3.45057×10^9	3.45057×10^9	3.45057×10^9	3.45057×10^9	0.00000
BIRCH3 dataset. 100.000 data vectors in $\mathbb{R}^2$, k = 100 clusters, time limitation 10 s					
$AGGL_2$	1.49271×10^9	1.57474×10^9	1.49822×10^9	1.49445×10^9	1.51721×10^7
$AGGL_5$	1.49341×10^9	1.57503×10^9	1.50433×10^9	1.49439×10^9	2.44200×10^7
Aggl-EA↔⇔	1.49199×10^9	1.57449×10^9	1.50670×10^9	1.49495×10^9	2.46374×10^7
BIRCH3 dataset. 100.000 data vectors in $\mathbb{R}^2$, k = 300 clusters, time limitation 10 s					
$AGGL_{250}$	9.08532×10^8	9.78792×10^8	9.36947×10^8	9.29497×10^8	1.97893×10^7
GH-VNS3	9.12455×10^8	9.44414×10^8	9.31225×10^8	9.32001×10^8	8.25653×10^6
Aggl-EA↔⇔	9.14179×10^8	9.71905×10^8	9.34535×10^8	9.33403×10^8	1.51920×10^7
Mopsi-Joensuu dataset. 6014 data vectors in $\mathbb{R}^2$, k = 30 clusters, time limitation 5 s					
$AGGL_{20}$	145.7784	145.8113	145.7869	145.7847	0.0084
GH-VNS3	145.7721	146.1273	145.7932	145.7761	0.0664
Aggl-EA↑⇑	145.7721	145.7752	145.7738	145.7739	0.0008
Mopsi-Joensuu dataset. 6014 data vectors in $\mathbb{R}^2$, k = 100 clusters, time limitation 5 s					
$AGGL_{75}$	43.6375	45.6856	44.1450	43.9860	0.4683
$AGGL_{100}$	43.6054	44.9035	44.0472	43.9875	0.3061
Aggl-EA↑⇑	43.5826	44.4452	43.7486	43.7560	0.1624
Mopsi-Joensuu dataset. 6014 data vectors in $\mathbb{R}^2$, k = 300 clusters, time limitation 5 s					
$AGGL_{250}$	15.0975	16.8500	15.7103	15.6757	0.3640
Aggl-EA↔⇔	14.8354	19.4531	15.4576	15.3310	0.8131
IHEPC dataset. 2,075,259 data vectors in $\mathbb{R}^7$, k = 30 clusters, time limitation 5 min					
j-means	87907.7813	95,055.2422	89,657.8895	88,702.9297	2442.1302
Aggl-EA↔⇔	86147.6953	10,9817.7969	91,515.1658	86,393.3750	10,377.0956
IHEPC dataset. 2,075,259 data vectors in $\mathbb{R}^7$, k = 100 clusters, time limitation 5 min s					
GA-1POINT	62,192.1719	64,413.2578	63,051.7500	63,028.9922	733.3966
Aggl-EA↔⇑	57,594.0703	72,909.5000	65,782.6094	58,506.6484	3208.6879
IHEPC dataset. 2,075,259 data vectors in $\mathbb{R}^7$, k = 300 clusters, time limitation 5 min s					
j-means	44,074.6445	47,562.4922	45,375.1657	44,608.1953	1505.4071
Aggl-EA↔⇔	41,795.5078	65,057.9375	51,888.0642	42,525.5000	12,122.2332
Mopsi- Finland dataset.13,467 data vectors in $\mathbb{R}^2$, k = 30 clusters, time limitation 5 s					
$AGGL_7$	1.03013×10^7	1.03013×10^7	1.03013×10^7	1.03013×10^7	3.35641
Aggl-EA↔⇔	1.03013×10^7	1.03013×10^7	1.03013×10^7	1.03013×10^7	2.42022
Mopsi-Finland dataset. 13,467 data vectors in $\mathbb{R}^2$, k = 100 clusters, time limitation 5 s					
$AGGL_{25}$	3.64354×10^6	3.66605×10^6	3.64913×10^6	3.64784×10^6	5184.35
$AGGL_{30}$	3.64290×10^6	3.67201×10^6	3.65106×10^6	3.64702×10^6	7780.79
Aggl-EA↔⇔	3.64285×10^6	3.65052×10^6	3.64473×10^6	3.64502×10^6	1467.87
Mopsi-Finland dataset. 13,467 data vectors in $\mathbb{R}^2$, k = 300 clusters, time limitation 5 s					
GH-VNS3	1.36493×10^6	1.43665×10^6	1.38037×10^6	1.37633×10^6	14603.0
Aggl-EA↔⇔	1.36496×10^6	1.40537×10^6	1.38311×10^6	1.38149×10^6	12302.2
S1 dataset. 5000 data vectors in $\mathbb{R}^2$, k = 15 clusters, time limitation 1 s					
GH-VNS1	1.69034×10^8	1.69034×10^8	1.69034×10^8	1.69034×10^8	0.00000
Aggl-EA↔⇔	1.69034×10^8	1.69034×10^8	1.69034×10^8	1.69034×10^8	0.00000
S1 dataset. 5000 data vectors in $\mathbb{R}^2$, k = 50 clusters, time limitation 1 s					
$AGGL_3$	1.12426×10^8	1.12548×10^8	1.12465×10^8	1.12457×10^8	2.92460×10^4
GH-VNS1	1.12419×10^8	1.12796×10^8	1.12467×10^8	1.12446×10^8	7.47785×10^4
Aggl-EA↔⇔	1.12476×10^8	1.15978×10^8	1.14049×10^8	1.13985×10^8	1.05017×10^6
S4 dataset. 5000 data vectors in $\mathbb{R}^2$, k = 15 clusters, time limitation 1 s					
GH-VNS1	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	0.00000
Aggl-EA↔⇔	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	78.6011
S4 dataset. 5000 data vectors in $\mathbb{R}^2$, k = 50 clusters, time limitation 1 s					
$AGGL_7$	1.35232×10^8	1.35449×10^8	1.35306×10^8	1.35294×10^8	5.64815×10^4
Aggl-EA↔⇔	1.35241×10^8	1.35438×10^8	1.35313×10^8	1.35304×10^8	5.26194×10^4

Note: "↑", "⇑": the advantage of the new algorithms over known algorithms is statistically significant ("↑" for *t*-test and "⇑" for Mann–Whitney U test), "↓", "⇓": the disadvantage of the new algorithm over known algorithms is statistically significant; "↔", "⇔": the advantage or disadvantage is statistically insignificant. Significance level is 0.05. The best results are underlined.

The time limitation plays an important role: the fastest algorithms may stop their convergence after several iterations and, vice versa, the slowest algorithms may continue their slow convergence and outperform the fastest algorithms if we enlarge their time limits. However, our new algorithms demonstrate their comparative advantage within wide time intervals (see Figure 3).

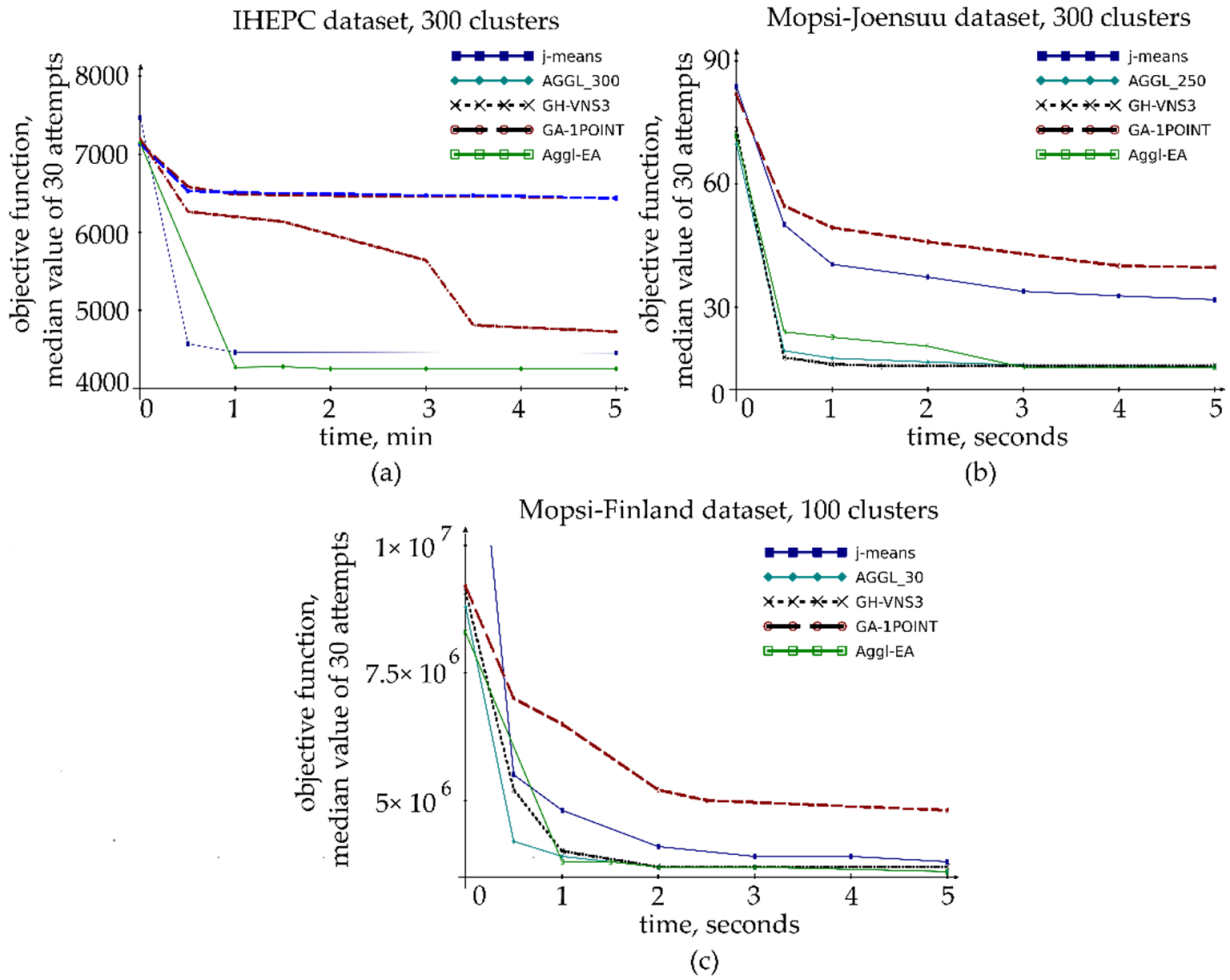

Figure 3. Comparative analysis of the convergence speed. Dependence of the median result on computation time for: (**a**) Individual Household Electric Power Consumption (IHEPC) dataset, 50 centers, 2,075,259 data vectors, time limitation 300 s; (**b**) Mopsi-Joensuu dataset, search for 300 centers, 6014 data vectors, time limitation 5 s. (**c**) Mopsi-Finland dataset, search for 300 centers, 13,467 data vectors, time limitation 5 s.

4. Discussion

The genetic algorithms and Variable Neighborhood Search are algorithmic frameworks useful for creating efficient solvers of various problems including location problems such as p-median. The greedy agglomerative approach can be efficiently used as the crossover genetic operator [58,70,71,102] as well as in the VNS [52]. However, as our experiments show (Appendix A), the correct choice of parameter r in such procedures plays the most important role and simple search algorithms with constant value of r may outperform more complex VNS algorithms if r is tuned correctly.

Unlike the majority of evolutionary algorithms, the (1 + 1)-evolutionary algorithms as well as local search algorithms focus on the successive improvement of a single intermediate solution. Successful application of various local search algorithms such as SWAP search, j-means or various VNS algorithms including to the p-median problem shows that the correct choice of a neighborhood or an algorithmic procedure for the successive improvement

of a single current solution can be a more profitable strategy for solving practical problems than operating with a large population of solutions. In this work, we have increased the efficiency of one of these procedures by adjusting its parameter. Nevertheless, adjusting the parameters of greedy agglomerative procedures can also enhance the capabilities of genetic algorithms with a greedy agglomerative crossing operator, which, in our opinion, is an urgent area of further research. In addition, similar algorithms can be developed for a wider range of problems, including k-means, k-medoid, and the problem of separating a mixture of probability distributions.

Our new algorithm demands significant computational resources. Nevertheless, obtaining the most accurate results for solving problems, in addition to being of immediate practical importance in the case of a high cost of error, solves the problem of obtaining reference solutions with which the results of other, less computationally expensive algorithms can be compared.

As in the case of the k-means problem [74], the most complex dependence of the greedy procedure efficiency on its parameter r and important advantage of our new algorithm is detected for the largest tested problems (IHEPC dataset) as well as for the problems of "geographic" location (Mopsi datasets) with the sets of demand points formed under the influence of natural factors, as well as factors associated with the development of urban infrastructure.

For the majority of test problems, our computational experiments demonstrate the advantage of our new algorithm or its approximately equal effectiveness in comparison with known algorithms. For large-scale problem, the effect of our new algorithm is more significant. Nevertheless, even with equal results in comparison with known algorithms, our new Aggl-EA algorithm is a more versatile tool due to its ability of adjusting its parameter in accordance with its behavior. Sometimes, our new algorithm demonstrates less stable results (higher standard deviation of the objective function) which may limit its scope or demands running in multi-start mode. Thus, further study of the causes and factors leading to the instability of the result is required.

We considered the use of the self-adjusted agglomerative procedures as the mutation operator of the simplest evolutionary algorithm with no true population of solutions. The efficiency of embedding similar mutation operators (without any self-adjustment) was shown in the paper [96]. Thus, the investigation of the self-adjustment features of other evolutionary operators (first of all, crossover operator of genetic algorithms) is a promising direction for the further research.

5. Conclusions

In this article, we introduced the concept of the $AGGL_r$ neighborhood based on the application of the agglomerative procedure, and investigate the search efficiency in such a neighborhood depending on the parameter r. Using the similarities between local search algorithms and (1 + 1)-evolutionary algorithms, as well as the ability of the latter to adapt their search parameters, we introduced our new Aggl-EA () algorithm with an embedded greedy agglomerative procedure with the automatically tuned parameter r. Our computational experiments on large-scale and medium-scale problems demonstrate the advantages of the new algorithm in comparison with known algorithms including the algorithms based on greedy agglomerative procedures.

Our computational experiments confirmed the proposed working hypotheses and led to the following conclusions:

The agglomerative mutation operator, when used as part of an evolutionary algorithm, is not only able to improve its solutions, moreover, it can also be efficiently used as the only evolutionary operator in the (1 + 1)-evolutionary algorithm with results outperforming more complex genetic algorithms. Therefore, self-adjustment capability of other evolutionary operators based on agglomerative procedures is a promising direction for the further research.

The algorithm with the adjustment of the parameter r (the number of excess centers to be removed) of the agglomerative procedure shows the most impressive results in the case of solving large problems, as well as problems of a "geographic" nature, where many points of demand have a complex structure, formed under the influence of natural factors.

Such algorithms, implemented for massively parallel systems, despite the high computational complexity of the embedded agglomerative procedure, ALA-algorithm and Weiszfeld procedure, are capable of solving problems with several million demand points in a reasonable time.

Author Contributions: Conceptualization, L.K.; methodology, L.K. and I.R.; software, L.K.; validation, I.R.; formal analysis, L.K. and G.S.; investigation, I.R.; resources, L.K.; data curation, I.R.; writing—original draft preparation, L.K. and G.S.; writing—review and editing, L.K.; visualization, I.R.; supervision, L.K.; project administration, L.K.; funding acquisition, L.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by The Ministry of Science and Higher Education of the Russian Federation, project No. FEFE-2020-0013.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: http://cs.joensuu.fi/sipu/datasets/ (accessed on 21 April 2020), https://archive.ics.uci.edu/ml/index.php (accessed on 21 April 2020).

Conflicts of Interest: The authors declare no conflict of interest. The funder had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Appendix A. Detailed Results of Computational Experiments

Table A1. Comparative results for BIRCH3 dataset. 10^5 data vectors in $\mathbb{R}^2$, $p = 30$ centers, time limitation 10 s.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	4.38183×10^9	4.67540×10^9	4.52387×10^9	4.51929×10^9	7.82433×10^7
j-means ($SWAP_1$ + Lloyd)	3.56107×10^9	5.25877×10^9	3.92248×10^9	3.88872×10^9	3.09203×10^8
$AGGL_1$	3.45692×10^9	3.56599×10^9	3.46697×10^9	3.45692×10^9	2.40890×10^7
$AGGL_2$	$\underline{3.45057 \times 10^9}$	3.45692×10^9	3.45650×10^9	3.45692×10^9	1.61204×10^6
$AGGL_3$	$\underline{3.45057 \times 10^9}$	3.45692×10^9	3.45650×10^9	3.45692×10^9	1.61204×10^6
$AGGL_{5–20}$ (equal results)	3.45692×10^9	3.45692×10^9	3.45692×10^9	3.45692×10^9	$\underline{0.00000}$
$AGGL_{25}$	$\underline{3.45057 \times 10^9}$	3.45057×10^9	$\underline{3.45057 \times 10^9}$	$\underline{3.45057 \times 10^9}$	46.7390
$AGGL_{30}$	$\underline{3.45057 \times 10^9}$	3.45057×10^9	$\underline{3.45057 \times 10^9}$	$\underline{3.45057 \times 10^9}$	127.558
GH-VNS1-3 (equal results)	$\underline{3.45057 \times 10^9}$	3.45057×10^9	$\underline{3.45057 \times 10^9}$	$\underline{3.45057 \times 10^9}$	$\underline{0.00000}$
$SWAP_1$ (the best of SWAP)	3.70386×10^9	4.43612×10^9	3.89665×10^9	3.89201×10^9	1.24246×10^8
GA-1POINT	3.51505×10^9	3.63646×10^9	3.56546×10^9	3.56848×10^9	3.24395×10^7
GA-UNIFORM	$\underline{3.45057 \times 10^9}$	3.59155×10^9	3.49515×10^9	3.49778×10^9	3.07081×10^7
Aggl-EA	$\underline{3.45057 \times 10^9}$	3.45057×10^9	$\underline{3.45057 \times 10^9}$	$\underline{3.45057 \times 10^9}$	$\underline{0.00000}$

Note (for all tables): the best results are underlined.

Table A2. Comparative results for BIRCH3 dataset. 10^5 data vectors in $\mathbb{R}^2$, $p = 100$ centers, time limitation 10 s.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	2.27163×10^9	2.54224×10^9	2.44280×10^9	2.46163×10^9	7.42330×10^7
j-means ($SWAP_1$ + Lloyd)	1.52130×10^9	1.88212×10^9	1.69226×10^9	1.67712×10^9	8.03684×10^7
$AGGL_1$	1.56737×10^9	2.14554×10^9	1.81181×10^9	1.80137×10^9	1.60897×10^8
$AGGL_2$	1.49271×10^9	1.57474×10^9	$\underline{1.49822 \times 10^9}$	1.49445×10^9	1.51721×10^7
$AGGL_3$	1.49232×10^9	1.59406×10^9	1.50429×10^9	1.49447×10^9	2.64626×10^7
$AGGL_5$	1.49341×10^9	1.57503×10^9	1.50433×10^9	$\underline{1.49439 \times 10^9}$	2.44200×10^7

Table A2. *Cont.*

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
$AGGL_7$	1.49341×10^9	1.59630×10^9	1.51531×10^9	1.49515×10^9	3.58332×10^7
$AGGL_{10}$	1.49261×10^9	1.60585×10^9	1.51345×10^9	1.50733×10^9	2.79785×10^7
$AGGL_{12}$	1.49358×10^9	1.61554×10^9	1.51105×10^9	1.49754×10^9	3.16092×10^7
$AGGL_{15}$	1.49414×10^9	1.57599×10^9	1.50791×10^9	1.49667×10^9	2.39108×10^7
$AGGL_{20}$	1.49406×10^9	1.57432×10^9	1.51096×10^9	1.50880×10^9	1.83795×10^7
$AGGL_{25}$	1.49386×10^9	1.53280×10^9	1.51017×10^9	1.51159×10^9	1.23389×10^7
$AGGL_{30}$	1.49355×10^9	1.53113×10^9	1.50656×10^9	1.50311×10^9	1.12716×10^7
$AGGL_{50}$	1.49380×10^9	1.52818×10^9	1.50495×10^9	1.50073×10^9	1.11666×10^7
$AGGL_{75}$	1.49415×10^9	1.56935×10^9	1.51541×10^9	1.51203×10^9	1.95288×10^7
$AGGL_{100}$	1.49413×10^9	1.56603×10^9	1.51133×10^9	1.49943×10^9	2.01570×10^7
GH-VNS1	1.49362×10^9	1.66273×10^9	1.54540×10^9	1.51469×10^9	5.07461×10^7
GH-VNS2	1.49413×10^9	1.59073×10^9	1.50692×10^9	1.50235×10^9	1.87495×10^7
GH-VNS3	1.49247×10^9	1.55423×10^9	1.50561×10^9	1.49497×10^9	1.84421×10^7
$SWAP_1$ (the best of SWAP)	1.70412×10^9	2.00689×10^9	1.83218×10^9	1.83832×10^9	6.04418×10^7
GA-1POINT	1.61068×10^9	1.85786×10^9	1.69042×10^9	1.65346×10^9	7.23930×10^7
GA-UNIFORM	1.67990×10^9	1.92521×10^9	1.77471×10^9	1.76783×10^9	6.95451×10^7
Aggl-EA	1.49199×10^9	1.57449×10^9	1.50670×10^9	1.49495×10^9	2.46374×10^7

Table A3. Comparative results for BIRCH3 dataset. 10^5 data vectors in $\mathbb{R}^2$, $p = 300$ centers, time limitation 10 s.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	1.54171×10^9	1.64834×10^9	1.59859×10^9	1.60045×10^9	2.62443×10^7
j-means ($SWAP_1$ + Lloyd)	9.95926×10^8	1.15171×10^9	1.05927×10^9	1.04561×10^9	4.63788×10^7
$AGGL_1$	1.14784×10^9	1.88404×10^9	1.47311×10^9	1.42352×10^9	1.94210×10^8
$AGGL_2$	9.96893×10^8	1.23746×10^9	1.08456×10^9	1.07354×10^9	5.76487×10^7
$AGGL_3$	9.87064×10^8	1.21980×10^9	1.07148×10^9	1.06503×10^9	4.84664×10^7
$AGGL_5$	9.68357×10^8	1.13508×10^9	1.06002×10^9	1.06617×10^9	4.50726×10^7
$AGGL_7$	1.00537×10^9	1.19050×10^9	1.07440×10^9	1.06799×10^9	4.73520×10^7
$AGGL_{10}$	9.78925×10^8	1.17041×10^9	1.06123×10^9	1.05264×10^9	5.10834×10^7
$AGGL_{12}$	9.94720×10^8	1.16133×10^9	1.05659×10^9	1.05930×10^9	3.59604×10^7
$AGGL_{15}$	9.64129×10^8	1.18512×10^9	1.05884×10^9	1.04935×10^9	5.54774×10^7
$AGGL_{20}$	9.53544×10^8	1.15383×10^9	1.03729×10^9	1.03124×10^9	4.58983×10^7
$AGGL_{25}$	9.98103×10^8	1.13062×10^9	1.03990×10^9	1.02958×10^9	3.39141×10^7
$AGGL_{30}$	9.59132×10^8	1.08646×10^9	1.02528×10^9	1.02102×10^9	3.48713×10^7
$AGGL_{50}$	9.48152×10^8	1.07480×10^9	1.00313×10^9	9.97259×10^8	2.65919×10^7
$AGGL_{75}$	9.44390×10^8	1.05320×10^9	9.90754×10^8	9.86511×10^8	3.04626×10^7
$AGGL_{100}$	9.33977×10^8	1.02524×10^9	9.70365×10^8	9.63525×10^8	2.48207×10^7
$AGGL_{150}$	9.20206×10^8	1.02162×10^9	9.61877×10^8	9.60465×10^8	2.42984×10^7
$AGGL_{200}$	9.18310×10^8	1.01805×10^9	9.52666×10^8	9.42071×10^8	2.73422×10^7
$AGGL_{250}$	9.08532×10^8	9.78792×10^8	9.36947×10^8	9.29497×10^8	1.97893×10^7
$AGGL_{300}$	9.10975×10^8	9.77193×10^8	9.39030×10^8	9.39434×10^8	1.27907×10^7
GH-VNS1	1.00289×10^9	1.08471×10^9	1.03856×10^9	1.03485×10^9	2.20294×10^7
GH-VNS2	9.68045×10^8	1.11832×10^9	1.01461×10^9	1.00404×10^9	3.80607×10^7
GH-VNS3	9.12455×10^8	9.44414×10^8	9.31225×10^8	9.32001×10^8	8.25653×10^6
$SWAP_2$ (the best of SWAP by avg.)	1.23379×10^9	1.46395×10^9	1.33987×10^9	1.35305×10^9	5.64094×10^7
$SWAP_3$ (the best of SWAP by median)	1.25432×10^9	1.44136×10^9	1.34388×10^9	1.34424×10^9	5.01104×10^7
GA-1POINT	1.11630×10^9	1.38598×10^9	1.23404×10^9	1.22828×10^9	7.05936×10^7
GA-UNIFORM	1.17534×10^9	1.37190×10^9	1.26758×10^9	1.25424×10^9	5.20153×10^7
Aggl-EA	9.14179×10^8	9.71905×10^8	9.34535×10^8	9.33403×10^8	1.51920×10^7

Table A4. Comparative results for S1 dataset. 5000 data vectors in $\mathbb{R}^2$, $p = 15$ centers, time limitation 1 s.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	1.69034×10^8	2.02847×10^8	1.79099×10^8	1.69034×10^8	1.56374×10^7
j-means ($SWAP_1$ + Lloyd)	1.69034×10^8	2.18797×10^8	1.74375×10^8	1.69034×10^8	1.40223×10^7
$AGGL_1$	1.69034×10^8	2.02619×10^8	1.71265×10^8	1.69034×10^8	8.49108×10^6
$AGGL_{2\text{-}15}$ (equal results)	1.69034×10^8	1.69034×10^8	1.69034×10^8	1.69034×10^8	0.00000
$SWAP_1$ (the best of SWAP)	1.69034×10^8	2.08453×10^8	1.70348×10^8	1.69034×10^8	7.19690×10^6
GA-1POINT	1.69034×10^8	2.02783×10^8	1.76877×10^8	1.69034×10^8	1.44603×10^7
GA-UNIFORM	1.69034×10^8	1.69034×10^8	1.69034×10^8	1.69034×10^8	2.92119
Aggl-EA	1.69034×10^8	1.69034×10^8	1.69034×10^8	1.69034×10^8	0.00000

Table A5. Comparative results for S1 dataset. 5000 data vectors in $\mathbb{R}^2$, $p = 50$ centers, time limitation 1 s.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	1.14205×10^8	1.16737×10^8	1.15594×10^8	1.15666×10^8	6.33573×10^5
j-means ($SWAP_1$ + Lloyd)	1.13500×10^8	1.17631×10^8	1.15035×10^8	1.14954×10^8	9.38353×10^5
$AGGL_1$	1.13121×10^8	1.15265×10^8	1.14219×10^8	1.14035×10^8	6.23645×10^5
$AGGL_2$	1.12430×10^8	1.12572×10^8	1.12475×10^8	1.12464×10^8	3.76180×10^4
$AGGL_3$	1.12426×10^8	1.12548×10^8	1.12465×10^8	1.12457×10^8	2.92460×10^4
$AGGL_5$	$1.12446E \times 10^8$	1.12552×10^8	1.12487×10^8	1.12483×10^8	3.00588×10^4
$AGGL_7$	1.12437×10^8	1.12591×10^8	1.12481×10^8	1.12478×10^8	3.26658×10^4
$AGGL_{10}$	1.12462×10^8	1.12558×10^8	1.12500×10^8	1.12500×10^8	2.68212×10^4
$AGGL_{15}$	1.12462×10^8	1.12579×10^8	1.12526×10^8	1.12528×10^8	2.83854×10^4
$AGGL_{20}$	1.12461×10^8	1.12590×10^8	1.12535×10^8	1.12538×10^8	3.27419×10^4
$AGGL_{25}$	1.12472×10^8	1.12632×10^8	1.12541×10^8	1.12543×10^8	4.34545×10^4
$AGGL_{30}$	1.12523×10^8	1.12662×10^8	1.12572×10^8	1.12566×10^8	3.55121×10^4
$AGGL_{50}$	1.12541×10^8	1.12848×10^8	1.12666×10^8	1.12651×10^8	7.48936×10^4
GH-VNS1	1.12419×10^8	1.12796×10^8	1.12467×10^8	1.12446×10^8	7.47785×10^4
GH-VNS2	1.12472×10^8	1.12601×10^8	1.12525×10^8	1.12519×10^8	3.37221×10^4
GH-VNS3	1.12531×10^8	1.12969×10^8	1.12712×10^8	1.12708×10^8	9.60927×10^4
$SWAP_1$ (the best of SWAP)	1.13142×10^8	1.16627×10^8	1.14430×10^8	1.14412×10^8	8.48529×10^5
GA-1POINT	1.14271×10^8	1.16790×10^8	1.15443×10^8	1.15343×10^8	7.16204×10^5
GA-UNIFORM	1.13119×10^8	1.15805×10^8	1.14384×10^8	1.14405×10^8	6.75227×10^5
Aggl-EA	1.12476×10^8	1.15978×10^8	1.14049×10^8	1.13985×10^8	1.05017×10^6

Table A6. Comparative results for S4 dataset. 5000 data vectors in $\mathbb{R}^2$, $p = 15$ centers, time limitation 1 s.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	25.4490
j-means ($SWAP_1$ + Lloyd)	2.27694×10^8	2.60475×10^8	2.30812×10^8	2.27694×10^8	7.33507×10^6
$AGGL_1$	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	26.4692
$AGGL_2$	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	6.88293
$AGGL_3$	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	7.84212
$AGGL_5$	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	14.9936
$AGGL_7$	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	26.1335
$AGGL_{10}$	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	112.307
$AGGL_{12}$	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	41.2754
$AGGL_{15}$	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	81.9819
GH-VNS1	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	0.00000
GH-VNS2	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	35.8018
GH-VNS3	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	81.1213
$SWAP_1$ (the best of SWAP)	2.27694×10^8	2.38720×10^8	2.28342×10^8	2.27694×10^8	2.48983×10^6
GA-1POINT	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	17.3669
GA-UNIFORM	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	8.55973
Aggl-EA	2.27694×10^8	2.27694×10^8	2.27694×10^8	2.27694×10^8	78.6011

Table A7. Comparative results for S4 dataset. 5000 data vectors in $\mathbb{R}^2$, $p = 50$ centers, time limitation 1 s.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	1.35718×10^8	1.38141×10^8	1.37187×10^8	1.37299×10^8	4.98608×10^5
j-means ($SWAP_1$ + Lloyd)	1.35353×10^8	1.38227×10^8	1.36673×10^8	1.36589×10^8	7.55140×10^5
$AGGL_1$	1.35378×10^8	1.37459×10^8	1.36291×10^8	1.36286×10^8	5.78265×10^5
$AGGL_2$	1.35237×10^8	1.35416×10^8	1.35353×10^8	1.35378×10^8	5.39798×10^4
$AGGL_3$	1.35237×10^8	1.35450×10^8	1.35372×10^8	1.35384×10^8	5.64379×10^4
$AGGL_5$	1.35229×10^8	1.35675×10^8	1.35388×10^8	1.35388×10^8	6.46057×10^4
$AGGL_7$	1.35232×10^8	1.35449×10^8	1.35306×10^8	1.35294×10^8	5.64815×10^4
$AGGL_{10}$	1.35248×10^8	1.35438×10^8	1.35325×10^8	1.35314×10^8	5.64928×10^4
$AGGL_{12}$	1.35259×10^8	1.35422×10^8	1.35321×10^8	1.35312×10^8	4.98093×10^4
$AGGL_{15}$	1.35267×10^8	1.35424×10^8	1.35323×10^8	1.35312×10^8	4.16267×10^4
$AGGL_{20}$	1.35294×10^8	1.35448×10^8	1.35347×10^8	1.35331×10^8	4.45414×10^4
$AGGL_{25}$	1.35264×10^8	1.35470×10^8	1.35348×10^8	1.35342×10^8	5.03965×10^4
$AGGL_{30}$	1.35260×10^8	1.35453×10^8	1.35345×10^8	1.35336×10^8	4.55230×10^4
$AGGL_{50}$	1.35254×10^8	1.35467×10^8	1.35354×10^8	1.35336×10^8	5.50925×10^4
GH-VNS1	1.35219×10^8	1.35758×10^8	1.35408×10^8	1.35383×10^8	1.27073×10^5
GH-VNS2	1.35239×10^8	1.35433×10^8	1.35327×10^8	1.35324×10^8	4.90352×10^4
GH-VNS3	1.35299×10^8	1.35469×10^8	1.35374×10^8	1.35382×10^8	5.29113×10^4
$SWAP_1$ (the best of SWAP)	1.35930×10^8	1.39391×10^8	1.37402×10^8	1.37351×10^8	8.40302×10^5
GA-1POINT	1.35943×10^8	1.38086×10^8	1.36776×10^8	1.36723×10^8	4.96830×10^5
GA-UNIFORM	1.35260×10^8	1.36698×10^8	1.35917×10^8	1.35937×10^8	3.50731×10^5
Aggl-EA	1.35241×10^8	1.35438×10^8	1.35313×10^8	1.35304×10^8	5.26194×10^4

Table A8. Comparative results for Mopsi-Joensuu dataset. 6014 data vectors in $\mathbb{R}^2$, $p = 100$ centers, time limitation 5 s.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	99.7279	123.1325	110.4963	109.9706	5.6551
j-means ($SWAP_1$ + Lloyd)	48.3311	57.1689	51.6673	51.1146	2.0749
$AGGL_1$	51.9696	81.9848	64.2454	62.4350	8.3403
$AGGL_2$	44.4866	50.0924	46.1544	46.0973	1.0708
$AGGL_3$	43.8615	46.3881	45.1735	45.3303	0.7306
$AGGL_5$	43.7373	46.0294	44.8369	44.7245	0.6888
$AGGL_7$	44.0820	48.4483	45.2256	45.0851	0.9135
$AGGL_{10}$	43.8603	45.7149	44.6455	44.6387	0.5400
$AGGL_{15}$	43.7995	45.5927	44.7005	44.7173	0.5203
$AGGL_{20}$	43.6629	46.0316	44.4626	44.3353	0.5953
$AGGL_{25}$	43.6357	46.5550	44.5179	44.3408	0.6543
$AGGL_{30}$	43.6728	44.5830	44.1677	44.1137	0.2945
$AGGL_{50}$	43.6202	45.0433	44.2257	44.3137	0.3590
$AGGL_{75}$	43.6375	45.6856	44.1450	43.9860	0.4683
$AGGL_{100}$	43.6054	44.9035	44.0472	43.9875	0.3061
GH-VNS1	47.7171	59.6970	53.4896	53.1948	3.4123
GH-VNS2	43.7781	46.0085	44.8602	44.9149	0.5772
GH-VNS3	43.8585	46.4490	44.7263	44.6593	0.6018
$SWAP_1$ (the best of SWAP)	47.8800	52.4030	49.8287	49.5523	1.1239
GA-1POINT	61.2677	76.9114	67.2297	66.9182	3.8341
GA-UNIFORM	68.6276	102.0867	84.8705	83.8152	7.4129
Aggl-EA	43.5826	44.4452	43.7486	43.7560	0.1624

Table A9. Comparative results for Mopsi-Joensuu dataset. 6014 data vectors in $\mathbb{R}^2$, p = 30 centers, time limitation 5 s.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	190.1025	229.0199	218.3214	219.6611	8.1825
j-means ($SWAP_1$ + Lloyd)	146.3243	158.0798	151.0530	150.6979	2.7908
$AGGL_1$	145.7872	163.1270	152.6856	153.2432	5.6803
$AGGL_2$	145.7738	146.2262	145.8728	145.7798	0.1532
$AGGL_3$	145.7745	146.2161	145.9185	145.7841	0.1744
$AGGL_5$	145.7742	146.1392	145.8211	145.7830	0.1088
$AGGL_7$	145.7738	146.1416	145.8188	145.7829	0.1084
$AGGL_{10}$	145.7770	146.1370	145.8295	145.7831	0.1209
$AGGL_{15}$	145.7742	146.1392	145.8211	145.7830	0.1088
$AGGL_{20}$	145.7784	145.8113	145.7869	145.7847	0.0084
$AGGL_{25}$	145.7753	146.1479	145.8043	145.7893	0.0680
$AGGL_{30}$	145.7791	146.1466	145.8022	145.7877	0.0670
GH-VNS1	145.7738	146.2193	145.8769	145.7789	0.1537
GH-VNS2	145.7729	146.1506	145.8323	145.7780	0.1243
GH-VNS3	145.7721	146.1273	145.7932	145.7761	0.0664
$SWAP_1$ (the best of SWAP)	145.9265	155.8482	148.4986	148.0995	1.8866
GA-1POINT	148.2359	164.0893	154.9963	154.6930	3.4858
GA-UNIFORM	153.8591	201.7184	175.8969	174.2682	10.9540
Aggl-EA	145.7721	145.7752	145.7738	145.7739	0.0008

Table A10. Comparative results for Mopsi-Joensuu dataset. 6014 data vectors in $\mathbb{R}^2$, p = 300 centers, time limitation 5 s.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	47.8874	54.4850	51.6395	51.1162	1.6453
j-means ($SWAP_1$ + Lloyd)	23.6798	35.3805	31.2554	31.8670	2.7425
$AGGL_1$	34.3990	54.5291	42.5890	41.8879	5.0023
$AGGL_2$	18.6255	22.8995	20.4858	20.3853	1.0785
$AGGL_3$	16.7389	19.8415	18.5376	18.6063	0.7018
$AGGL_5$	16.5944	19.4305	17.7512	17.5557	0.7611
$AGGL_7$	16.1609	20.0563	17.8918	17.8397	0.8182
$AGGL_{10}$	16.4099	19.0087	17.3922	17.2220	0.6228
$AGGL_{15}$	16.0706	17.4835	16.7584	16.7537	0.4276
$AGGL_{20}$	15.7783	17.6122	16.6852	16.7338	0.4712
$AGGL_{25}$	15.6854	18.3011	16.7847	16.6473	0.5885
$AGGL_{50}$	15.7963	17.7948	16.2860	16.2920	0.4237
$AGGL_{100}$	15.1942	16.3370	15.6951	15.6738	0.3081
$AGGL_{150}$	15.2025	16.4996	15.7898	15.7990	0.2939
$AGGL_{200}$	15.1805	16.2245	15.7252	15.7801	0.2843
$AGGL_{250}$	15.0975	16.8500	15.7103	15.6757	0.3640
$AGGL_{300}$	15.2509	16.2803	15.7108	15.7224	0.2694
GH-VNS1	21.5583	31.4467	27.7853	27.9011	2.5876
GH-VNS2	15.5928	17.6197	16.6424	16.6345	0.5166
GH-VNS3	15.3488	16.8864	15.9644	15.8850	0.4229
$SWAP_5$ (the best of SWAP by avg.)	22.3193	30.7398	27.5174	27.9711	1.9760
$SWAP_7$ (the best of SWAP by median)	24.0243	30.9356	27.6329	27.8934	1.9714
GA-1POINT	34.4429	45.0868	40.1539	39.6896	2.3970
GA-UNIFORM	37.4806	53.3750	43.8100	43.7275	3.7585
Aggl-EA	14.8354	19.4531	15.4576	15.3310	0.8131

Table A11. Comparative results for Individual Household Electric Power Consumption (IHEPC) dataset. 2,075,259 data vectors in $\mathbb{R}^7$, p = 30 centers, time limitation 5 min.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	88,145.6484	93,677.3281	90,681.1663	89,967.9297	1818.1249
j-means ($SWAP_1$ + Lloyd)	87,907.7813	95,055.2422	89,657.8895	88,702.9297	2442.1302
$AGGL_1$	91,021.6016	110,467.5625	104,694.9900	109,976.7266	9278.2649
$AGGL_2$	86,291.1406	109,972.0781	99,788.2522	109,817.8125	12,557.4381
$AGGL_3$	109,817.8125	109,999.1328	109,913.6953	109,972.0781	90.3626
$AGGL_5$	86,240.7344	109,999.1328	103,145.4487	109,817.8281	11,519.4208
$AGGL_7$	86,345.9297	109,817.8359	99,781.6283	109,817.8203	12,517.3778
$AGGL_{10}$	86,414.0938	109,999.1563	106,500.3449	109,817.8281	8857.4620
$AGGL_{15}$	87,253.8281	109,817.8438	103,391.2009	109,817.8359	10,975.6502
$AGGL_{20}$	87,616.3984	109,999.1563	106,677.1384	109,817.8438	8405.2571
$AGGL_{25}$	87,409.8203	109,817.8438	106,616.6953	109,817.8438	8469.4358
$AGGL_{30}$	87,852.5938	109,878.0156	106,707.1674	109,853.0547	8314.1220
GH-VNS1	86,228.4844	109,999.1250	99,811.1853	109,817.7734	12,593.8724
GH-VNS2	86,368.7891	109,817.8281	99,807.6674	109,817.8125	12,485.0346
GH-VNS3	86,304.9141	109,817.8281	96,518.5525	86,865.2813	12,441.4202
$SWAP_1$ (the best of SWAP)	86,672.6250	112,683.0469	100,566.9531	109,979.8594	12,666.0305
GA-1POINT	88,170.4141	100,368.3438	92,664.3739	92,332.9922	4102.0677
GA-UNIFORM	87,650.1016	110,271.6172	96,631.6763	92,752.2813	9762.9279
Aggl-EA	86,147.6953	109,817.7969	91,515.1658	86,393.3750	10,377.0956

Table A12. Comparative results for Individual Household Electric Power Consumption (IHEPC) dataset. 2,075,259 data vectors in $\mathbb{R}^7$, p = 300 centers, time limitation 5 min.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	49,901.8086	58,872.1094	53,113.4275	52,599.0273	2781.6693
j-means ($SWAP_1$ + Lloyd)	44,074.6445	47,562.4922	45,375.1657	44,608.1953	1505.4071
$AGGL_1$	67,508.2344	69,943.5000	69,070.9118	69,328.3750	845.3473
$AGGL_2$	43,966.8594	67,854.2500	60,923.9386	67,576.6875	11,525.2718
$AGGL_3$	43,867.1211	68,638.1953	58,670.0921	67,070.9375	11,658.5487
$AGGL_5$	44,337.6641	68,679.3125	61,875.6590	67,577.0625	10,517.0718
$AGGL_7$	43,688.3750	68,976.5469	64,480.6272	67,636.7188	9192.1896
$AGGL_{10}$	66,566.2969	68,004.2500	67,175.4855	67,188.0938	510.9257
$AGGL_{15}$	43,160.3125	67,295.1797	57,406.8198	66,625.8047	12,069.9982
$AGGL_{20}$	43,819.1289	67,265.2188	60,462.2919	66,883.5938	11,234.3160
$AGGL_{25}$	43,324.3203	67,231.0859	60,237.4286	66,842.1719	11,452.1535
$AGGL_{50}$	42,628.9141	66,279.0781	62,340.2411	65,467.8828	8700.5474
$AGGL_{100}$	64,635.6055	65,408.7891	65,055.4102	65,155.6328	284.9207
$AGGL_{150}$	64,458.3047	65,008.1641	64,682.8622	64,706.7969	188.7753
$AGGL_{200}$	41,754.1680	64,694.6328	58,240.8147	64,582.5625	10,883.6398
$AGGL_{250}$	64,385.2461	64,646.9141	64,538.0787	64,547.0820	82.7326
$AGGL_{300}$	41,373.4453	65,051.3594	61,283.1027	64,492.8828	8781.8445
GH-VNS1	67,290.1406	68,655.8125	67,645.6563	67,467.4375	478.1909
GH-VNS2	43,395.0273	67,046.0234	63,081.6233	66,222.0313	8689.9374
GH-VNS3	42,124.9375	64,425.2734	61,204.4487	64,358.4844	8413.3478
$SWAP_3$ (the best of SWAP by avg.)	46,137.9570	70,000.2813	63,618.6362	68,361.8672	9467.2699
$SWAP_{100}$ (the best of SWAP by median)	51,026.5859	70,514.3359	65,787.6618	67,617.5781	6667.2423
GA-1POINT	45,618.3828	52,858.6875	49,198.6964	47,304.4453	3129.2994
GA-UNIFORM	46,171.7148	62,034.5273	52,218.1077	50,642.4961	5712.0544
Aggl-EA	41,795.5078	65,057.9375	51,888.0642	42,525.5000	12,122.2332

Table A13. Comparative results for Individual Household Electric Power Consumption (IHEPC) dataset. 2,075,259 data vectors in $\mathbb{R}^7$, $p = 100$ centers, time limitation 5 min.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	61,803.3047	65,997.1406	64,612.3371	65,037.8281	1431.8760
j-means ($SWAP_1$ + Lloyd)	61,374.5742	70,854.6953	63,433.8962	62,227.1172	3308.9649
$AGGL_1$	60,709.2500	88,798.9453	82,890.4255	86,253.2539	9221.4664
$AGGL_2$	58,098.7578	85,587.0938	78,612.5859	84,406.7891	11,517.2424
$AGGL_3$	58,508.1367	84,416.4219	75,654.4905	83,573.3906	11,916.7366
$AGGL_5$	58,326.4648	83,891.5781	69,386.2210	59,046.0703	13,415.1144
$AGGL_7$	58,623.8203	84,465.2500	69,618.7757	59,477.0938	13,118.7602
$AGGL_{10}$	58,665.4414	84,494.0938	72,985.3951	82,966.5156	13,078.1135
$AGGL_{15}$	58,306.4961	83,287.7891	69,407.9023	59,881.0156	12,347.6134
$AGGL_{20}$	58,666.2383	83,110.8438	76,161.1618	82,986.7578	11,757.2349
$AGGL_{25}$	58,533.1406	83,046.0703	69,305.3025	59,392.3672	12,821.3261
$AGGL_{50}$	58,768.5430	82,938.1797	69,450.1853	60,540.5391	12,428.4255
$AGGL_{75}$	58,538.9922	82,544.6875	75,635.4358	82,477.6172	11,673.3413
$AGGL_{100}$	58,460.1914	82,531.0000	78,951.6490	82,422.8125	9036.9781
GH-VNS1	59,228.6055	85,246.7969	73,899.8482	83,708.7031	13,239.8092
GH-VNS2	82,966.8750	83,334.9688	83,061.5056	83,033.9844	128.9717
GH-VNS3	59,417.9375	82,291.0391	78,905.5815	82,124.6094	8593.7433
$SWAP_{10}$ (the best of SWAP)	61,196.4414	84,698.2109	70,051.5938	62,676.3125	10,652.0083
GA-1POINT	62,192.1719	64,413.2578	63,051.7500	63,028.9922	733.3966
GA-UNIFORM	60,873.5859	66,829.2969	63,656.8555	64,155.6875	2084.6202
Aggl-EA	57,594.0703	72,909.5000	65,782.6094	58,506.6484	3208.6879

Table A14. Comparative results for Mopsi-Finland dataset. 13,467 data vectors in $\mathbb{R}^2$, $p = 100$ centers, time limitation 5 s.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	6.28454×10^6	7.25139×10^6	6.92349×10^6	6.95864×10^6	198,384
j-means ($SWAP_1$ + Lloyd)	3.76240×10^6	4.00651×10^6	3.85266×10^6	3.84859×10^6	59,481.8
$AGGL_1$	4.17611×10^6	5.71925×10^6	4.87105×10^6	4.79499×10^6	417,612
$AGGL_2$	3.64502×10^6	3.67299×10^6	3.65556×10^6	3.65350×10^6	7644.61
$AGGL_3$	3.64508×10^6	3.77758×10^6	3.66037×10^6	3.65225×10^6	28,242.8
$AGGL_5$	3.64502×10^6	3.77520×10^6	3.65956×10^6	3.65221×10^6	31,464.1
$AGGL_7$	3.64352×10^6	3.77238×10^6	3.66457×10^6	3.65259×10^6	36,374.3
$AGGL_{10}$	3.64503×10^6	3.74222×10^6	3.65639×10^6	3.65191×10^6	18,087.8
$AGGL_{12}$	3.64508×10^6	3.77121×10^6	3.65745×10^6	3.65299×10^6	22,688.4
$AGGL_{15}$	3.64351×10^6	3.78305×10^6	3.66115×10^6	3.65050×10^6	32,121.2
$AGGL_{20}$	3.64289×10^6	3.74971×10^6	3.65333×10^6	3.64965×10^6	19,010.0
$AGGL_{25}$	3.64354×10^6	3.66605×10^6	3.64913×10^6	3.64784×10^6	5184.35
$AGGL_{30}$	3.64290×10^6	3.67201×10^6	3.65106×10^6	3.64702×10^6	7780.79
$AGGL_{50}$	3.64511×10^6	3.67594×10^6	3.65120×10^6	3.64940×10^6	6920.69
$AGGL_{75}$	3.64510×10^6	3.68136×10^6	3.65552×10^6	3.65355×10^6	7722.98
$AGGL_{100}$	3.64508×10^6	3.67524×10^6	3.65651×10^6	3.65395×10^6	9106.84
GH-VNS1	3.69265×10^6	5.20291×10^6	4.05038×10^6	3.93463×10^6	336,130
GH-VNS2	3.64503×10^6	3.66884×10^6	3.65123×10^6	3.65015×10^6	6284.54
GH-VNS3	3.64503×10^6	3.67655×10^6	3.65735×10^6	3.65370×10^6	9247.64
$SWAP_1$ (the best of SWAP)	3.68260×10^6	3.79031×10^6	3.74321×10^6	3.74447×10^6	25,846.1
GA-1POINT	4.38348×10^6	5.29721×10^6	4.83383×10^6	4.82569×10^6	198,278
GA-UNIFORM	5.00518×10^6	6.29115×10^6	5.62017×10^6	5.60439×10^6	347,120
Aggl-EA	3.64285×10^6	3.65052×10^6	3.64473×10^6	3.64502×10^6	1467.87

Table A15. Comparative results for Mopsi-Finland dataset. 13,467 data vectors in $\mathbb{R}^2$, p = 300 centers, time limitation 5 s.

Algorithm or Neighborhood	Achieved Objective Function Values (1) Summarized after 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std. Dev
Lloyd (multistart)	3.28826×10^6	3.58614×10^6	3.46203×10^6	3.47443×10^6	80,996.0
j-means ($SWAP_1$ + Lloyd)	1.59039×10^6	2.13796×10^6	1.74473×10^6	1.71790×10^6	120,717
$AGGL_1$	2.34524×10^6	3.59085×10^6	3.03226×10^6	2.97788×10^6	300,773
$AGGL_2$	1.47203×10^6	1.68652×10^6	1.53552×10^6	1.52156×10^6	55,909.7
$AGGL_3$	1.41824×10^6	1.65971×10^6	1.49980×10^6	1.49186×10^6	50,351.7
$AGGL_5$	1.41376×10^6	1.69573×10^6	1.48142×10^6	1.47572×10^6	53,828.0
$AGGL_7$	1.42594×10^6	1.67197×10^6	1.49188×10^6	1.47882×10^6	50,893.8
$AGGL_{10}$	1.42666×10^6	1.61224×10^6	1.48780×10^6	1.48111×10^6	37,416.1
$AGGL_{15}$	1.38648×10^6	1.68429×10^6	1.51728×10^6	1.50331×10^6	61,906.0
$AGGL_{20}$	1.43027×10^6	1.67186×10^6	1.51494×10^6	1.49912×10^6	63,635.8
$AGGL_{25}$	1.42352×10^6	1.74796×10^6	1.51280×10^6	1.49324×10^6	75,237.2
$AGGL_{30}$	1.42871×10^6	1.66177×10^6	1.49960×10^6	1.48752×10^6	51,468.1
$AGGL_{50}$	1.40068×10^6	1.47648×10^6	1.44410×10^6	1.44245×10^6	17,875.2
$AGGL_{75}$	1.38586×10^6	1.48921×10^6	1.42238×10^6	1.42229×10^6	23,714.1
$AGGL_{100}$	1.37383×10^6	1.43596×10^6	1.40575×10^6	1.41181×10^6	17,442.5
$AGGL_{150}$	1.36240×10^6	1.40877×10^6	1.38853×10^6	1.38929×10^6	11,719.5
$AGGL_{200}$	1.35686×10^6	1.42109×10^6	1.38415×10^6	1.38600×10^6	17,504.0
$AGGL_{250}$	1.35268×10^6	1.40887×10^6	1.38198×10^6	1.38182×10^6	13,577.4
$AGGL_{300}$	1.35163×10^6	1.41077×10^6	1.38198×10^6	1.37998×10^6	15,384.2
GH-VNS1	1.70022×10^6	2.04728×10^6	1.87817×10^6	1.88773×10^6	101,777
GH-VNS2	1.43218×10^6	1.78314×10^6	1.51093×10^6	1.49528×10^6	70,431.8
GH-VNS3	1.36493×10^6	1.43665×10^6	1.38037×10^6	1.37633×10^6	14,603.0
$SWAP_1$ (the best of SWAP)	1.49832×10^6	1.72366×10^6	1.57134×10^6	1.56292×10^6	47,997.3
GA-1POINT	2.69754×10^6	3.29563×10^6	2.91834×10^6	2.88716×10^6	144,687
GA-UNIFORM	2.55194×10^6	3.53833×10^6	2.91604×10^6	2.89257×10^6	205,946
Aggl-EA	1.36496×10^6	1.40537×10^6	1.38311×10^6	1.38149×10^6	12,302.2

References

1. Drezner, Z.; Hamacher, H. *Facility Location: Applications and Theory*; Springer: Berlin, Germany, 2004.
2. Khachumov, M.V. Distances, metrics and data clustering. *Sci. Tech. Inf. Proc.* **2012**, *39*, 310–316. [CrossRef]
3. Çolakoglu, H.B. A Generalization of the Minkowski Distance and a New Definition of the Ellipse. Available online: https://arxiv.org/abs/1903.09657v1 (accessed on 12 March 2021).
4. France, S.; Carroll, J.D.; Xiong, H. Distance metrics for high dimensional nearest neighborhood recovery: Compression and normalization. *Inform. Sci.* **2012**, *184*, 92–110. [CrossRef]
5. Weiszfeld, E.; Plastria, F. On the point for which the sum of the distances to n given points is minimum. *Ann. Oper. Res.* **2009**, *167*, 7–41. [CrossRef]
6. Kuhn, H.W. A note on Fermat's problem. *Math. Program.* **1973**, *4*, 98–107. [CrossRef]
7. Weiszfeld, E. Sur le point sur lequel la somme des distances de n points donnes est minimum. *Tohoku Math. J.* **1937**, *43*, 335–386.
8. Sturm, R. Ueber den Punkt kleinster Entfernungssumme von gegebenen Punkten. *J. Rein. Angew. Math.* **1884**, *97*, 49–61.
9. Beck, A. Weiszfeld's Method: Old and New Results. *J. Optim. Theory Appl.* **2015**, *164*, 1–40. [CrossRef]
10. Garey, M.; Johnson, D.; Witsenhausen, H. The complexity of the generalized Lloyd—Max problem (Corresp.). *IEEE Trans. Inf. Theory* **1982**, *28*, 255–256. [CrossRef]
11. Farahani, R.Z.; Hekmatfar, M. *Facility Location Concepts, Models, Algorithms and Case Studies*; Springer: Berlin/Heidelberg, Germany, 2009. [CrossRef]
12. Hakimi, S.L. Optimum locations of switching centers and the absolute centers and medians of a graph. *Oper. Res.* **1964**, *12*, 450–459. [CrossRef]
13. Masuyama, S.; Ibaraki, T.; Hasegawa, T. The computational complexity of the m center problems on the plane. *Trans. Inst. Electron. Commun. Eng. Jpn.* **1981**, *64E*, 57–64.
14. Kariv, O.; Hakimi, S.L. An algorithmic approach to network location problems. The p-medians. *SIAM J. Appl. Math* **1979**, *37*, 539–560. [CrossRef]
15. Cooper, L. The weber problem revisited. *Comput. Math. Appl.* **1981**, *7*, 225–234. [CrossRef]
16. Lawrence, M. Ostresh. On the convergence of a class of iterative methods for solving the weber location problem. *Oper. Res.* **1978**, *26*, 597–609.

17. Plastria, F.; Elosmani, M. On the convergence of the Weiszfeld algorithm for continuous single facility location allocation problems. *TOP* **2008**, *16*, 388–406. [CrossRef]
18. Vardi, Y. The multivariate L1-median and associated data depth. *Proc. Natl. Acad. Sci. USA* **2000**, *97*, 1423–1426. [CrossRef] [PubMed]
19. Badoiu, M. Approximate clustering via core-sets. In Proceedings of the 34th Annual ACM Symposium on Theory of Computing, Montréal, QC, Canada, 19–21 May 2002; pp. 250–257.
20. Kuhn, H.W. An efficient algorithm for the numerical solution of the generalized Weber problem in spatial economics. *J. Reg. Sci.* **1962**, *4*, 21–34.
21. Mladenovic, N.; Brimberg, J.; Hansen, P.; Moreno-Perez, J.A. The p-median problem: A survey of metaheuristic approaches. *Eur. J. Oper. Res.* **2007**, *179*, 927–939. [CrossRef]
22. Reese, J. Solution methods for the p-median problem: An annotated bibliography. *Networks* **2006**, *48*, 125–142. [CrossRef]
23. Hakimi, S.L. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Oper. Res.* **1965**, *13*, 462–475. [CrossRef]
24. Kuenne, R.E.; Soland, R.M. Exact and approximate solutions to the multisource Weber problem. *Math. Program.* **1972**, *3*, 193–209. [CrossRef]
25. Ostresh, L.M.J. The stepwise location-allocation problem: Exact solutions in continuous and discrete spaces. *Geogr. Anal.* **1978**, *10*, 174–185. [CrossRef]
26. Rosing, K.E. An optimal method for solving the (generalized) multi-weber problem. *Eur. J. Oper. Res.* **1992**, *58*, 414–426. [CrossRef]
27. Rabbani, M. A novel approach for solving a constrained location allocation problem. *Int. J. Ind. Eng. Comput.* **2013**, *4*, 203–214. [CrossRef]
28. Fathali, J.; Rad, N.J.; Sherbaf, S.R. The p-median and p-center problems on bipartite graphs. *Iran. J. Math. Sci. Inf.* **2014**, *9*, 37–43. [CrossRef]
29. Avella, P.; Sassano, A.; Vasil'ev, I. Computational study of large-scale p-median problems. *Math. Program.* **2007**, *109*, 89–114. [CrossRef]
30. Avella, P.; Boccia, M.; Salerno, S.; Vasilyev, I. An aggregation heuristic for large-scale p-median problem. *Comput. Oper. Res.* **2012**, *39*, 1625–1632. [CrossRef]
31. Resende, M.G.C. Metaheuristic hybridization with greedy randomized adaptive search procedures. *Inf. TutORials Oper. Res.* **2008**, 295–319. [CrossRef]
32. Resende, M.G.C.; Ribeiro, C.C.; Glover, F.; Marti, R. Scatter search and path relinking: Fundamentals, advances, and applications. In *Handbook of Metaheuristics*; Gendreau, M., Potvin, J.-Y., Eds.; Springer: Boston, MA, USA, 2010; pp. 87–107. [CrossRef]
33. Brimberg, J.; Drezner, Z.; Mladenovic, N.; Salhi, S. A New Local Search for Continuous Location Problems. *Eur. J. Oper. Res.* **2014**, *232*, 256–265. [CrossRef]
34. Drezner, Z.; Brimberg, J.; Mladenovic, N.; Salhi, S. New heuristic algorithms for solving the planar p-median problem. *Comput. Oper. Res.* **2015**, *62*, 296–304. [CrossRef]
35. Drezner, Z.; Brimberg, J.; Mladenovic, N.; Salhi, S. Solving the planar p-median problem by variable neighborhood and concentric searches. *J. Glob. Optim.* **2015**, *63*, 501–514. [CrossRef]
36. Mladenovic, N.; Alkandari, A.; Pei, J.; Todosijevic, R.; Pardalos, P.M. Less is more approach: Basic variable neighborhood search for the obnoxious p -median problem. *Int. Trans. Oper. Res.* **2019**, *27*, 480–493. [CrossRef]
37. Bernábe-Loranca, M.; González-Velázquez, R.; Granillo-Martinez, E.; Romero-Montoya, M.; Barrera-Cámara, R. P-median problem: A real case application. In *Intelligent Systems Design and Applications. ISDA 2019. Advances in Intelligent Systems and Computing*; Springer: Cham, Switherlands, 2021; Volume 1181. [CrossRef]
38. Arthur, D.; Vassilvitskii, S. k-Means++: The Advantages of Careful Seeding. In Proceedings of the SODA'07, SIAM, New Orleans, LA, USA, 7–9 January 2007; pp. 1027–1035.
39. Hromkovic, J. *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*; Springer: Berlin/Heidelberg, Germany, 2011.
40. Ng, T. Expanding Neighborhood Tabu Search for facility location problems in water infrastructure planning. In Proceedings of the 2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC), San Diego, CA, USA, 5–8 October 2014; pp. 3851–3854. [CrossRef]
41. Kochetov, Y.; Mladenovic, N.; Hansen, P. Local search with alternating neighborhoods. *Discret. Anal. Oper. Res.* **2003**, *10*, 11–43. (In Russian)
42. Hansen, P. Variable neighborhood search: Principles and applications. *Eur. J. Oper. Res* **2001**, *130*, 449–467. [CrossRef]
43. Hansen, P.; Mladenovic, N. Development of Variable Neighborhood Search. In *Essays and Surveys in Metaheuristics*; Ribeiro, C.C., Hansen, P., Eds.; Kluwer Academic Publishers: Dordrecht, The Netherlands, 2002; pp. 415–440.
44. Mladenovic, N. Variable neighborhood search. *Comput. Oper. Res.* **1997**, *24*, 1097–1100. [CrossRef]
45. Kochetov, Y.A. Local Search Methods for Discrete Location Problems. Ph.D. Thesis, Sobolev Institute of Mathematics SB RAS, Novosibirsk, Russia, 19 January 2010. (In Russian).
46. Hansen, P. Variable Neighborhood Search. In *Search Methodology*; Bruke, E.K., Kendall, G., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 211–238. [CrossRef]

47. Brimberg, J.; Mladenovic, N. A variable neighborhood algorithm for solving the continuous location-allocation problem. *Stud. Locat. Anal.* **1996**, *10*, 1–12.
48. Hansen, P.; Mladenovic, N.; Perez-Brito, D. Variable neighborhood decomposition search. *J. Heuristics* **2001**, *7*, 335–350. [CrossRef]
49. Brimberg, J.; Hansen, P.; Mladenovic, N.; Taillard, E. Improvements and comparison of heuristics for solving the uncapacitated multisource Weber problem. *Oper. Res.* **2000**, *48*, 444–460. [CrossRef]
50. Kochetov, Y.; Alekseeva, E.; Levanova, T.; Loresh, M. Large neighborhood local search for the p-median problem. *Yugosl. J. Oper. Res.* **2005**, *15*, 53–63. [CrossRef]
51. Lopez, F.G.; Batista, B.M.; Moreno-Perez, J.; Moreno-Vega, M. The parallel variable neighborhood search for the p-median problem. *J. Heuristics* **2002**, *8*, 375–388. [CrossRef]
52. Rozhnov, I.P.; Orlov, V.I.; Kazakovtsev, L.A. VNS-Based algorithms for the centroid-based clustering problem. *FACTA Univ. Ser. Math. Inform.* **2019**, *34*, 957–972.
53. Still, S.; Bialek, W.; Bottou, L. Geometric clustering using the information bottleneck method, Advances. In *Neural Information Processing Systems. 16*; MIT Press: Cambridge, UK, 2004.
54. Sun, Z.; Fox, G.; Gu, W.; Li, Z. A parallel clustering method combined information bottleneck theory and centroid-based clustering. *J. Supercomput.* **2014**, *69*, 452–467. [CrossRef]
55. Houck, C.R.; Joines, J.A.; Kay, M.G. Comparison of genetic algorithms, random restart and two-opt switching for solving large location-allocation problems. *Comput. Oper. Res.* **1996**, *23*, 587–596. [CrossRef]
56. Maulik, U.; Bandyopadhyay, S. Genetic algorithm-based clustering technique. *Pattern Recognit.* **2000**, *33*, 1455–1465. [CrossRef]
57. Krishna, K.; Murty, M.M. Genetic k-means algorithm. *IEEE Trans. Syst. Man Cybernetics. Part B* **1999**, *29*, 433–439. [CrossRef] [PubMed]
58. Neema, M.N.; Maniruzzaman, K.M.; Ohgai, A. New genetic algorithms based approaches to continuous p-median problem. *Netw. Spat. Econ.* **2011**, *11*, 83–99. [CrossRef]
59. Tuba, E.; Strumberger, I.; Tuba, I.; Bacanin, N.; Tuba, M. Water cycle algorithm for solving continuous p-median problem. In Proceedings of the SACI 2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics, Timiuoara, Romania, 17–19 May 2018; pp. 351–354. [CrossRef]
60. Levanova, T.V.; Gnusarev, A.Y. Simulated annealing for competitive p–median facility location problem. *J. Phys. Conf. Ser.* **2018**, *1050*, 012044. [CrossRef]
61. Zhao, H.; Zhang, C. An online-learning-based evolutionary many-objective algorithm. *Inf. Sci.* **2020**, *509*, 1–21. [CrossRef]
62. Dulebenets, M.A. An adaptive island evolutionary algorithm for the berth scheduling problem. *Memetic Comp.* **2020**, *12*, 51–72. [CrossRef]
63. Liu, Z.Z.; Wang, Y.; Huang, P.Q. AnD: A many-objective evolutionary algorithm with angle-based selection and shift-based density estimation. *Inf. Sci.* **2020**, *509*, 400–419. [CrossRef]
64. Ruiz, E.; Soto-Mendoza, V.; Barbosa, A.E.R.; Reyes, R. Solving the open vehicle routing problem with capacity and distance constraints with a biased random key genetic algorithm. *Comput. Ind. Eng.* **2019**, *133*, 207–219. [CrossRef]
65. Bae, H.; Moon, I. Multi-depot vehicle routing problem with time windows considering delivery and installation vehicles. *Appl. Math. Model.* **2016**, *40*, 6536–6549. [CrossRef]
66. Pasha, J.; Dulebenets, M.A.; Kavoosi, M.; Abioye, O.F.; Wang, H.; Guo, W. An optimization model and solution algorithms for the vehicle routing problem with a "factory-in-a-box". *IEEE Access* **2020**, *8*, 134743–134763. [CrossRef]
67. D'Angelo, G.; Pilla, R.; Tascini, C.; Rampone, S. A proposal for distinguishing between bacterial and viral meningitis using genetic programming and decision trees. *Soft Comput.* **2019**, *23*, 11775–11791. [CrossRef]
68. Panda, N.; Majhi, S.K. How Effective is the Salp Swarm Algorithm in Data Classification. In *Computational Intelligence in Pattern Recognition. Advances in Intelligent Systems and Computing*; Das, A., Nayak, J., Naik, B., Pati, S., Pelusi, D., Eds.; Springer: Singapore, 2020; Volume 999. [CrossRef]
69. Falkenauer, E. *Genetic Algorithms and Grouping Problems*; Wiley: New York, NY, USA, 1998.
70. Alp, O.; Erkut, E.; Drezner, Z. An efficient genetic algorithm for the p-median problem. *Ann. Oper. Res.* **2003**, *122*, 21–42. [CrossRef]
71. Kazakovtsev, L.A.; Antamoshkin, A.N. Genetic algorithm with fast greedy heuristic for clustering and location problems. *Informatica* **2014**, *38*, 229–240.
72. Hosage, C.M.; Goodchild, M.F. Discrete space location-allocation solutions from genetic algorithms. *Ann. Oper. Res.* **1986**, *6*, 35–46. [CrossRef]
73. Blum, C.; Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *Acm Comput. Surv.* **2001**, *35*, 268–308. [CrossRef]
74. Kazakovtsev, L.; Rozhnov, I.; Popov, A.; Tovbis, E.M. Self-adjusting variable neighborhood search algorithm for near-optimal k-means clustering. *Computation* **2020**, *8*, 90. [CrossRef]
75. Lloyd, S.P. Least Squares Quantization in PCM. *IEEE Trans. Inf. Theory* **1982**, *28*, 129–137. [CrossRef]
76. MacQueen, J.B. Some methods of classification and analysis of multivariate observations. In Proceedings of the 5th Berkley Symposium on Mathematical Statistics and Probability, California, CA, USA, 21 June–18 July 1965; Volume 1, pp. 281–297.

77. Kazakovtsev, L.A.; Rozhnov, I.P. Comparative study of local search in SWAP and agglomerative neighbourhoods for the continuous p-median problem. In Proceedings of the IOP Conference Series: Materials Science and Engineering, Volume 1047, III International Conference MIST: Aerospace 2020: Advanced Technologies in Aerospace, Mechanical and Automation Engineering (Aerospace 2020), Krasnoyarsk, Russia, 20–21 November 2020; Volume 1047. [CrossRef]
78. Droste, S.; Jansen, T.; Wegener, I. On the analysis of the (1+1) evolutionary algorithm. *Theor. Comput. Sci.* **2002**, *276*, 51–81. [CrossRef]
79. Borisovsky, P.A.; Eremeev, A.V. A study on performance of the (1+1)-Evolutionary Algorithm. In *Foundations of Genetic Algorithms*; De Jong, K., Poli, R., Rowe, J., Eds.; Morgan Kaufmann: San Francisco, CA, USA, 2003; pp. 271–287.
80. Eremeev, A.V.; Borisovsky, P.A. Comparing evolutionary algorithms to the (1+1)–EA. *Theor. Comput. Sci.* **2008**, *403*, 33–41. [CrossRef]
81. Sung, C.W.; Yuen, S.Y. Analysis of (1+1) evolutionary algorithm and randomized local search with memory. *Evol. Comput.* **2011**, *19*, 287–323. [CrossRef]
82. Doerr, B.; Johannsen, D.; Schmidt, M. Runtime analysis of the (1+1) evolutionary algorithm on strings over finite alphabets. In Proceedings of the 11th Workshop on Foundations of Genetic Algorithms (FOGA'11), Schwarzenberg, Austria, 5–9 January 2011; pp. 119–126. [CrossRef]
83. Peng, X. Performance analysis of (1+1)EA on the maximum independent set problem. In *Lecture Notes in Computer Science*; Springer: Cham, Switherlands, 2015; Volume 9483. [CrossRef]
84. Xia, X.; Zhou, Y. Approximation performance of the (1+1) evolutionary algorithm for the minimum degree spanning tree problem. In *Communications in Computer and Information Science*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 562. [CrossRef]
85. Bian, C.; Qian, C.; Tang, K.; Yu, Y. Running time analysis of the (1+1)-EA for robust linear optimization. *Theor. Comput. Sci.* **2020**, *843*, 57–72. [CrossRef]
86. Doerr, B.; Le, H.P.; Makhmara, R.; Nguyen, T.D. Fast genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017*; Bosman, P.A.N., Ed.; Spriger: Berlin, Germany, 2017; pp. 777–784. [CrossRef]
87. Cooper, L. Heuristic methods for location-allocation problems. *SIAM Rev.* **1964**, *6*, 37–53. [CrossRef]
88. Jiang, J.L.; Yuan, X.M. A heuristic algorithm for constrained multi-source Weber problem. The variational inequality approach. *Eur. J. Oper. Res.* **2007**, *187*, 357–370. [CrossRef]
89. Dempster, A.P.; Laird, N.M.; Rubin, D.B. Maximum likelihood from incomplete data via the EM algorithm. *J. R. Stat. Soc. Ser. B* **1977**, *39*, 1–38. [CrossRef]
90. O'Callaghan, L.; Mishra, N.; Meyerson, A.; Guha, S.; Motwani, R. Streaming-data algorithms for high-quality clustering. In Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, 26 February–1 March 2002; pp. 685–694. [CrossRef]
91. Ackermann, M.R.; Martens, M.; Raupach, C.; Swierkot, K.; Lammersen, C.; Sohler, C. Streamkm: A clustering algorithm for data streams. *J. Exp. Algorithms* **2010**, *17*, art.2.4. [CrossRef]
92. Kazakovtsev, L.; Stashkov, D.; Gudyma, M.; Kazakovtsev, V. Algorithms with Greedy Heuristic Procedures for Mixture Probability Distribution Separation. *Yugosl. J. Oper. Res.* **2018**, *29*, 51–67. [CrossRef]
93. Nikolaev, A.; Mladenovic, N.; Todosijevic, R. J-means and I-means for minimum sum-of-squares clustering on networks. *Optim. Lett.* **2017**, *11*, 359–376. [CrossRef]
94. Clustering Basic Benchmark. Available online: http://cs.joensuu.fi/sipu/datasets/ (accessed on 25 September 2020).
95. Dua, D.; Graff, C. UCI Machine Learning Repository 2019. Available online: http://archive.ics.uci.edu/ml (accessed on 30 September 2020).
96. Kazakovtsev, L.; Rozhnov, I.; Shkaberina, G.; Orlov, V. K-Means genetic algorithms with greedy genetic operators. *Math. Probl. Eng.* **2020**, *2020*, 8839763. [CrossRef]
97. Kazakovtsev, L.; Rozhnov, I. Application of algorithms with variable greedy heuristics for k-medoids problems. *Informatica* **2020**, *44*, 55–61. [CrossRef]
98. Luebke, D.; Humphreys, G. How GPUs work. *Computer* **2007**, *40*, 96–100. [CrossRef]
99. Lim, G.; Ma, L. GPU-based parallel vertex substitution algorithm for the p-median problem. *Comput. Ind. Eng.* **2013**, *64*, 381–388. [CrossRef]
100. AlBdaiwi, B.F.; AboElFotoh, H.M.F. A GPU-based genetic algorithm for the p-median problem. *J. Supercomput.* **2017**, *73*, 4221–4244. [CrossRef]
101. Herda, M. Parallel genetic algorithm for capacitated p-median problem. *Procedia Eng.* **2017**, *192*, 313–317. [CrossRef]
102. Zechner, M.; Granitzer, M. Accelerating K-Means on the Graphics Processor via CUDA. In Proceedings of the International Conference on Intensive Applications and Services, Valencia, Spain, 20–25 April 2009; pp. 7–15. [CrossRef]
103. Charikar, M.; Guha, S.; Tardos, E.; Shmoys, D.B. A constant-factor approximation algorithm for the k-median problem. In Proceedings of the 31st Annual ACM Symposium on Theory of Computing, Atlanta, GA, USA, 1–4 May 1999; pp. 1–10.
104. Jain, K.; Vazirani, V. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM* **2001**, *48*, 274–296. [CrossRef]
105. Fränti, P.; Sieranoja, S. K-means properties on six clustering benchmark datasets. *Appl. Intell.* **2018**, *48*, 4743–4759. [CrossRef]

106. Smucker, M.D.; Allan, J.; Carterette, B.A. Comparison of Statistical Significance Tests for Information Retrieval. In Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management (CIKM'07), Lisbon, Portugal, 6−10 November 2007; pp. 623–632.
107. Park, H.M. *Comparing Group Means: The t-Test and One-Way ANOVA Using STATA, SAS, and SPSS*; Indiana University: Bloomington, IN, USA, 2009.
108. Mann, H.B.; Whitney, D.R. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *Ann. Math. Stat.* **1947**, *18*, 50–60. [CrossRef]
109. Fay, M.P.; Proschan, M.A. Wilcoxon-Mann-Whitney or t-Test? On Assumptions for Hypothesis Tests and Multiple Interpretations of Decision Rules. *Stat. Surv.* **2010**, *4*, 1–39. [CrossRef]