Solving Large *p*-Median Problems by a Multistage Hybrid Approach Using Demand Points Aggregation and Variable Neighbourhood Search*

Chandra A. Irawan • Said Salhi

Abstract A hybridisation of a clustering-based technique and of a Variable Neighbourhood Search (VNS) is designed to solve large-scale *p*-median problems. The approach is based on a multi-stage methodology where learning from previous stages is taken into account when tackling the next stage. Each stage is made up of several subproblems that are solved by a fast procedure to produce good feasible solutions. Within each stage, the solutions returned are put together to make up a new promising subset of potential facilities. This augmented *p*-median problem is then solved by VNS. As these problems used aggregation, a cost evaluation based on the original demand points instead of aggregation is computed for each of the 'aggregation'-based solution. The one yielding the least cost is then selected and its chosen facilities included into the next stages. This multi-stage process is repeated several times until a certain criterion is met. This approach is enhanced by an efficient way to aggregate the data and a neighbourhood reduction scheme when allocating demand points to their nearest facilities. The proposed approach is tested, using various values of *p*, on the largest data sets from the literature with up to 89,600 demand points with encouraging results.

Keywords *Variable neighbourhood search • Location problem • Aggregation • p-median*

Chandra A. Irawan • Said Salhi
Centre for Logistics and Heuristic Optimisation (CLHO)
Kent Business School
University of Kent, United Kingdom
Chandra: e-mail: ca259@kent.ac.uk
Said: s.salhi@kent.ac.uk

*This research has been partially supported by the Ministry of Science and Innovation of Spain under the research project ECO2011-24927, in part financed by the European Regional Development Fund (ERDF), and the Fundacion Seneca under the research project 15254/PI/10, and also by the Algerian Ministry of Education (Sciences Fondamentales), under research project PNR 8/U160/64.

1. Introduction

The objective of the *p*-median problem, which is also known as the minisum problem, is to find the location of *p* facilities among *n* discrete potential sites in such a way to minimise the sum of the distances between customers and their associated facilities. In certain circumstances, *p*-median problems may consist of a large number of demand points. These problems arise, for example, in urban or regional areas where the demand points are individual private residences. Francis et al. [7] stated that it may be sometimes impossible and time consuming to solve optimally the location problems involving a large number of demand points. It is quite common to aggregate demand points when solving large scale location problems. The idea behind the aggregation is to reduce the number of demand points to be small enough that an optimiser can be used. In this case, the original location problem is partitioned into smaller problems which can be solved within a reasonable amount of computing time. However, this aggregation may introduce errors in the data used by location models and models output. In the next section, we review those researchers that have studied the effects of aggregation on the solution of location problems.

The *p*-median problem is originally formulated by ReVelle and Swain [16] as follows:

Minimise $\sum_{i\in I}\sum_{j\in J} w_i d_{ij} Y_{ij}$ (1)

Subject to

$\sum_{j} Y_{ij} = 1 \quad \forall i \in I$ (2)

$\sum_{j\in J} X_j = p$ (3)

$Y_{ij} - X_j \le 0, \quad \forall i, j$ (4)

$X_j \in \{0,1\} \quad \forall j$ (5)

$Y_{ij} \in \{0,1\} \quad \forall i, j$ (6)

Where

(*I,J*) : set of customers $(i \in I = \{1,...,n\})$ and set of potential sites $(j \in J = \{1,...,M\})$ (ie : $n = |I|$ and $M = |J|$) respectively

w_i : demand or weight of customer *i*;

d_{ij} : distance between customer *i* and potential site *j*;

p : number of facilities to locate;

Y_{ij} = 1, if customer *i* is fully served by a facility at site *j* and = 0 otherwise;

X_j = 1, if a facility is opened at potential site *j* and = 0 otherwise;

The objective function is to minimise the total demand-weighted distance. Constraint (2) ensures that each customer *i* is assigned to one facility only. Constraint (3) guarantees that the number of facilities to be located is equal to *p* facilities. Constraint (4) states that the demand at customer *i* can only be allocated to a facility *j* (Y_{ij} = 1) only if a facility is opened at site *j* (X_j = 1).

In this paper, we establish a method that dynamically solves the large scale *p*-median problems tested in the literature using a multistage approach. This is based on aggregating demand points and using a powerful meta-heuristic such as Variable Neighbourhood Search (VNS). The paper is organized as follows. Section 2 describes a brief review of the related literature. The proposed methodology is presented in Section 3. In Section 4, the computational results are given. Our conclusion is summarised with a highlight of some suggestions in the last section.

2. Literature Review

This section provides an overview of aggregation techniques for large scale *p*-median problems, see Francis et al. [7] for an excellent review on this topic. The authors described aggregation error measurements and surveyed some of the principal papers about aggregation errors. They also classified the review into two main categories, namely median problems and centre/covering problems.

Table 1 describes our notation in location models which are focused on an aggregation approach, with BSU and ASU being short for Basic Spatial Unit and Aggregate Spatial Unit respectively.

Aggregation error was first formally defined by Hillsman and Rhoda [11] where three types of aggregation errors, namely source A, B, and C errors are proposed. These errors are also usually used in many papers to measure the aggregation scheme performance. Source A error appears when the distance between an ASU and a facility is used to solve a facility location problem, instead of the true average distance between a BSU and a facility. Source B error occurs when a facility is located at an ASU whereas source C error happens when a BSU is assigned to the wrong facility.

Table 1 Notation in location model

Notation	Description
$N = \{1,2,3,...,n\}$	the set of all BSUs
$C = (c_1, c_2, c_3, \ldots, c_n)$	the list of BSUs ($c_i = i$ for simplicity)
$M = \{1,2,3,...,m\}$	the set of all ASUs
$C' = (c'_1, c'_2, c'_3, \ldots, c'_m)$	the list of ASUs (c'_r : representative point of the r^{th} ASU)
N_k	the set of BSU in the k^{th} ASU, $k = \{1,2,3,...,m\}$ with $\bigcup_{k=1}^{m} N_k = N$
p	the number of facilities to be located
F	the optimal locations of the p facilities found with the original model (i.e., full model)
F′	the optimal locations of the p facilities found with the aggregated location model
$d(i, j)$	the distance between the i^{th} BSU and the j^{th} BSU
$\hat{d}(k, j)$	the distance between the k^{th} ASU and the j^{th} potential facility (in our study, each ASU acts as a potential facility)
$\tilde{d}(i,k) = d(i,c'_k)$	the distance between the i^{th} BSU and the k^{th} ASU
$f(F:C)$	objective function evaluated using F based on the original problem
$f(F':C)$	objective function evaluated using F‘ based on the original problem
$f(F':C')$	objective function evaluated using F‘ based on the aggregated problem

A method for eliminating source A and source B errors is introduced by Current and Schilling [3]. The method emphasized the way to measure the weighted travel distances in the *p*-median problem. To eliminate source A errors, a distance between the k^{th} ASU and the j^{th} facility is set as $\hat{d}(k, j) = \sum_{i \in N_k} w_i d(i, j)$ instead of $|N_k| \tilde{d}(j,k)$ with N_k being the set of aggregated BSUs in the k^{th} ASU. The former equation measures the true weighted travel distance to the potential facility from all BSUs. This measurement method can also eliminate source B errors. Note that this method cannot eliminate source C errors.

Casillas [2] showed that the A, B, and C errors cause two types of errors, namely the cost error $(ce = f(F':C) - f(F':C'))$ and the optimality error $(oe = f(F:C) - f(F':C))$. It is found that the optimality error was small for small *p*, but increased when the values of *p* and *m* were large. Hodgson and Neuman [13] introduced a Geographical Information System (GIS) method for eliminating source C error. The method spatially disaggregates data as needed during the solution procedure (“on the fly”) using Voronoi polygon. A median row-column aggregation method was given by Francis et al. [6] to find an aggregation which gives a small error bound. Hodgson et al. [14] studied the aggregation error effects on the discrete-space *p*-median model and introduced source D error in addition to the A, B, and C errors. This new error appears when a BSU is also a potential facility location. An earlier review on the issue of aggregation errors for the *p*-median

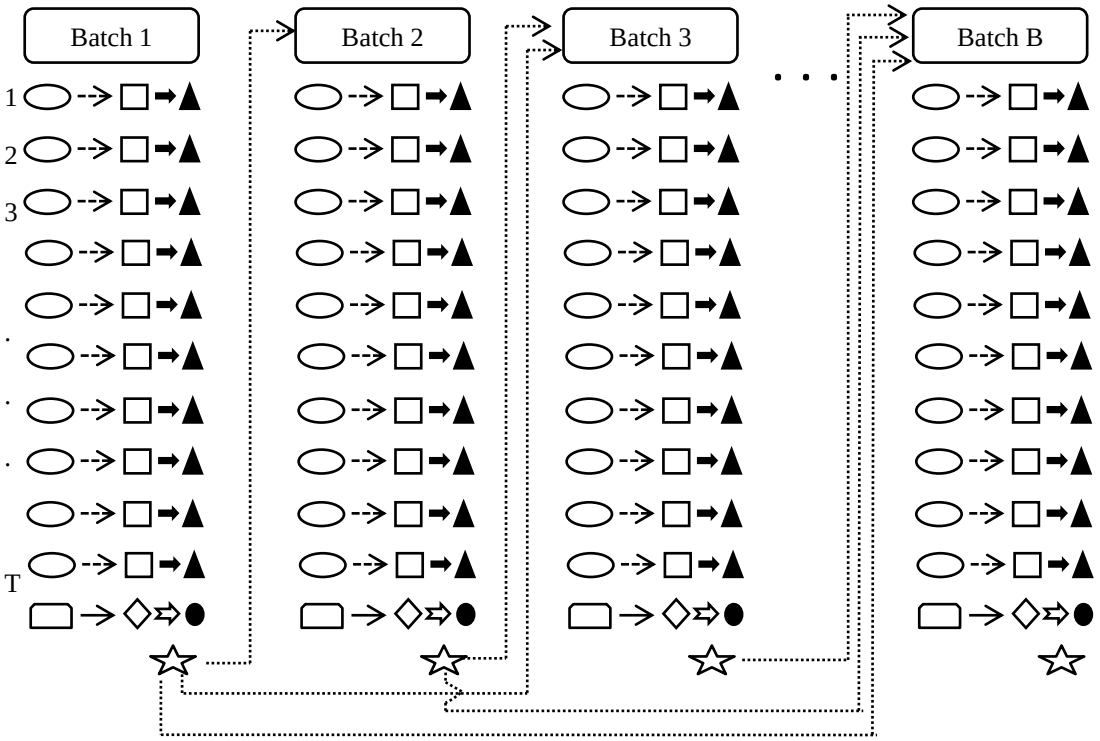
problem was provided by Erkut and Bozkaya [5]. Data surrogation error in *p*-median models was studied by Hodgson [12]. This error occurs when an inappropriate variable is used to stand in for a target population's demand.

Hansen et al. [9] developed a primal-dual VNS metaheuristic for solving large *p*-median datasets directly without recourse to any type of sampling. The authors used a Reduced VNS to get good initial solutions which are then used in their VNS with decomposition which is aimed to tackle large problems. Qi and Shen [17] studied the worst-case analysis of demand point aggregation for the Euclidean *p*-median problem on the plane. They utilised a "honeycomb heuristic" algorithm introduced by Papadimitriou [15] to develop a "multi-pattern tiling" to obtain smaller worst-case aggregation error bounds. Garcia et al. [8] investigated large *p*-median problems using a model based on covering formulation which has a small subset of constraints and variables. This method is very efficient due to an efficient branch-and-bound framework based on dynamic reliability branching within CPLEX. Their experiments showed that the method is able to solve large problems especially for large values of *p* due to the formulation reduction they proposed. Very recently, an aggregation heuristic was proposed by Avella et al. [1] where they used a heuristic approach based on Lagrangean relaxation to deal with large-scale median problems. The core heuristic is defined by a subset of the most promising variables found according to the Lagrangean reduced costs associated with the open facilities as well as those associated with the allocation variables. As this heuristic was failing for small values of *p*, an aggregation heuristic was introduced to solve the original problem with a much larger value of open facilities that are then considered as centres for aggregation. Their results were encouraging when compared to the ones given in [9].

3. Solution Approach

In this section, we propose an adaptive search method to solve the large scale *p*-median problems tested in the literature [1]. The method uses a clustering procedure to aggregate *n* BSUs (Basic Spatial Units) into *m* ASUs (Aggregated Spatial Units), where $m << n$. This is a multistage approach where in each stage (called batch) a number of aggregated sub-problems of *m* ASUs each is solved using a fast local search which we call the "Mini VNS". This generates a number of 'promising' facilities that are put together as potential sites. This augmented *p*-median problem is then solved with VNS

using still the aggregated problem. All the obtained solutions are re-evaluated for the original problem (i.e. all demand points) by incorporating an efficient reduction scheme. The best solution is then fed into the next stage. The process is repeated for future stages while adding the location of previously chosen facilities as potential sites. Once the process is stabilised (no improvement), a local search using the original problem is then carried out. An illustration of our methodology is presented in Fig. 1. The main steps of this method are given in Fig. 2 but more details are presented in the subsequent subsections. These include the mini VNS, the full VNS, the reduction scheme, and the local search. We refer to this method as the Hybrid Multistage Heuristic (HMH).



An aggregated problem (m demand points)

Solving the aggregated p-median problem using "Mini VNS (m, p)"

The obtained facilities location (F_t') found by "Mini VNS (m, p)", t= 1,...,T

Allocating all demand points in C to the nearest point in F_t'

The objective function ($f_t(F':C)$) calculated using F_t' based on the original problem

An aggregated problem made up of all solutions ($L_b = \bigcup_{t=1}^{T} F_t'$)

Solving the aggregated p-median problem using a full VNS ($|L_b|$, p) with $|L_b|$ potential sites

The obtained facilities location (F_b') found by the full VNS ($|L_b|$, p)

Allocating all demand points in C to the nearest point in F_b'

The objective function ($f_b(F':C)$) calculated using F_b' based on the original problem

The selected facilities from the solution with the smallest $f(F':C)$

Feeding the facilities found in ☆ to the next batches

Fig. 1 An Illustration of the Hybrid Multistage Heuristic (HMH)

Step 1. Set the number of batches (B), the number of ASUs (*m*), and the number of samples for each batch (T). Set $L_b = \emptyset$, *b* = (*1, …,* B), where L_b denotes a list of distinct facilities obtained in batch *b*. Set S = Ø, where S is a list of promising facility locations.

Step 2. Construct Cells covering all demand points based on *m*.

Step 3. For each batch, say *b* = (*1, …,* B).

Step 3a For each sample, say *t* = (*1, …,* T)

(i). Aggregate BSUs into *m* ASUs including the already found |S| facility locations, construct *m* clusters, and calculate the distance $\hat{d}(k, j)$, *k=1,..,m; j=1,…,m*.

(ii). Solve the aggregated *p*-median problem using a "Mini VNS (*m, p*)". Let $f_t(F':C)$ be the cost and store the obtained facility locations (F_t') into L_b, where $L_b = \bigcup_{t=1}^{T} F_t'$ and $F_t' = (\sigma_1^t, \sigma_2^t, ..., \sigma_p^t)$. All duplicate locations are obviously discarded when using $\bigcup$

Step 3b Construct $|L_b|$ clusters around the facilities in L_b, and calculate the distance $\hat{d}(k, j)$, *k=1,..,* $|L_b|$; *j=1,…,* $|L_b|$. Solve the *p*-median problem with VNS ($|L_b|$, *p*). Let $f_b(F':C)$ be the cost and F_b' the obtained facility locations.

Step 3c Add the promising facilities location into S. If $f_b(F':C) < \text{Min}\{f_t(F':C), t = 1,...,T\}$ then set $S = S \cup F_b'$ else $S = S \cup F_{t^*}'$ with $t^* = \text{Arg}(\underset{t}{\text{Min}}\ f_t(F_t', C))$.

Step 4. Starting from the obtained solution, solve the original *p*-median problem using a new local search (*n, p*) to be explained later.

Fig. 2 The main steps of the Hybrid Multistage Heuristic (HMH)

3.1. Determining the initial parameters (Step 1 of Fig. 2)

Firstly, we need to determine the number of batches (B). We assume that learning from previous batches will enhance the solution quality as extra 'useful' information are fed into the next batches. However, note that at some point, learning may not be very effective as the quality of solution may not necessarily get considerably better any more.

Secondly, we also need to choose the number of ASUs (*m*). The quality of the solution is affected by *m*. The higher the value of *m* is, the higher is the chance in getting a better solution. However, a higher value increases the computing time and requires a larger computer memory.

Finally, the number of samples (T) per batch also influences the quality of the solution. If T is high, the opportunity to get a good solution may increase as well due to diversification but at the expense of a longer computing time.

3.2. The construction of the cells (Step 2 of Fig. 2)

In this section, we describe a procedure to construct cells which will cover all demand points. We call this the Basic Cell Approach (BCA). The information of these cells will be used for determining *m* ASUs. The BCA is adopted from Salhi and Gamal [18]. The aim of constructing these cells is to overcome the weakness of the random process in dealing with clustered demand points. Here, we enhance their method in the following three ways. Firstly, BCA constructs c_L x c_W square cells that cover the entire study area and records the cumulative probability distribution of the number of demand points in the cells. In [18], c_0 x c_0 rectangular cells are constructed and the number of cells in column and row are the same. In our method, we construct c_L x c_W square cells and the number of cells in column and row are not necessary the same. Secondly, we pseudo randomly choose *m* cells based on the cumulative probability distribution. Thirdly, in each of the selected cell, we choose randomly a demand point as an ASU. The following are the main steps of the BCA:

- Construct c_L x c_W square cells of length δ and determine the number of cells (we aim to have more or less *m* cells). The following formula is used to compute δ.

$$\delta = \frac{x_{max} - x_{min}}{\sqrt{m\left(\frac{x_{max} - x_{min}}{y_{max} - y_{min}}\right)}} \qquad (7)$$

Proof:

As $\delta = \frac{x_{max} - x_{min}}{c_L} = \frac{y_{max} - y_{min}}{c_W}$ (8)

and our aim is to have *m* cells, therefore $m = c_L \cdot c_W \Rightarrow c_W = m / c_L$

Hence, $\frac{x_{max} - x_{min}}{c_L} = \frac{(y_{max} - y_{min})c_L}{m} \Rightarrow c_L = \sqrt{m\left(\frac{x_{max} - x_{min}}{y_{max} - y_{min}}\right)}$ (9)

Substituting (9) into (8) leads to (7).

The c_L and c_W can be obtained by $c_W = (y_{max} - y_{min})/\delta$ and $c_L = (x_{max} - x_{min})/\delta$. A cell is defined by its bottom-left corner. Let (X_z, Y_z), $z = 1,\ldots,(c_L.c_W)$, denote the coordinates of the bottom-left corner of the z^{th} cell. The cells are recorded as

follows: cell 1 has its bottom-left corner $(X_1, Y_1) = (x_{min}, y_{min})$ and successive cells, say cell *z* as follow:

$(X_z, Y_z) = (x_{min} + \delta(z \bmod c_L), y_{min} + \delta(z \bmod c_W))$

- Record the number of demand points in each cell, (say cell z) as N_z and determine its corresponding probability distribution, say $P_z = N_z / n$, $z = 1,\ldots,(c_L.c_W)$.
- Generate randomly $\beta \in (0,1)$ and choose $\tilde{z}$ st $\tilde{z} = F_{(z)}^{-1}(\beta)$ with $F_{(z)} = \sum_{t=1}^{z} P(t = z)$.

3.3. The location of the ASUs and the aggregation of the demand points (Step 3a(i) of Fig. 2)

This section will explain how the *m* ASUs are determined and the way the distance $\hat{d}(k, j)$ *k=1,..,m; j=1,…,m*, is calculated. The following steps summarise this mechanism.

- Store all the elements of S in *C'*. Note that in the first batch, |S| = 0.
- Repeat until there are *m* points in *C'*.
  (a) Choose a random number between 0 and 1, and then determine the corresponding cell *z* based on the cumulative probability distribution. Note that cell *z* is not the cell that has a point in S as its member.
  (b) Choose randomly a demand point in the cell *z* and store this point in *C'*. Duplicated points in *C'* are removed.
- Construct *m* clusters by allocating all demand points (*C*) to the nearest points in *C'*. Record the number of demand points in each cluster, o_k , $k = 1, \ldots, m$, where $\sum_k o_k = n$.
- Calculate the approximate distance between all the ASUs using $\hat{d}(k, j) = o_k \tilde{d}(j,k)$ with *j* denoting the facility representing the j^{th} ASU acting as a potential site and *k* refers to the k^{th} ASU (*k, j*=1,…,*m*). To reduce the computing time, we use such approximate distance measure instead of the real distance as proposed by Current and Schilling [3].

3.4. The allocation of demand points

We propose an efficient procedure to allocate all demand points (*n* points) to the nearest points in *C'* (*m* points). The following two steps constitute our fast procedure:

(a) An efficient recording of the Euclidian distance

The way to calculate the Euclidian distance is based on Zoubi and Rawi [20] where they developed an efficient way for computing silhouette coefficients when identifying 'good' clusters. Note that, the Euclidian distance, $d^2(i, j) = (x_i - x_j)^2 + (y_i - y_j)^2$, can also be rewritten as $x_i^2 + y_i^2 + x_j^2 + y_j^2 - 2x_i x_j - 2y_i y_j$. The terms $\gamma_i = x_i^2 + y_i^2$ and $\gamma_j = x_j^2 + y_j^2$ are based on *i* or *j* but not both. The authors observe that for each *i* (*i=1,…,n*), γ_i can be computed only once at the outset. It means that when we calculate the distance, the formulation $d^2(i, j) = \gamma_i + \gamma_j - 2x_i x_j - 2y_i y_j$ is used instead, yielding a saving in computing time as $O(n^2)$ operations are reduced to $O(n)$ only. However, there is a small fixed cost to compute the γ_i which is negligible especially when a large number of calls to the distance matrix is needed.

(b) An efficient allocation of points to ASU

Firstly, we need to find the nearest ASU from the k^{th} ASU. Instead of searching for all points, we limit our search to a smaller subset only. We define the covering radius as $\breve{d}(k)$, where $\breve{d}(k) = 0.5 * \text{Min}\{\hat{d}(k, j), k \neq j, j = 1,\ldots,m\}, k = 1,\ldots,m$. We allocate all demand points in *C* to the nearest ASU in *C'* using the following rule. If $\tilde{d}(i,k) \leq \breve{d}(k)$ then the demand point *i* is allocated to ASU *k*, otherwise the point *i* will be assigned to its nearest ASU using the classical allocation method. This reduction scheme will avoid the need for checking all the remaining ASUs. This rule also guarantees that a wrong allocation of the demand points is not possible.

An Illustrative Example

Fig. 3 illustrates the method of our efficient allocation method where there are 20 BSUs which will be aggregated into 3 ASUs by allocating each BSU to the nearest ASU. For example, when finding the nearest ASU, BSU 1, 2, …, 5 can be allocated to (aggregated into) ASU 1 directly without the need for checking all the remaining ASUs. This is because those BSUs are within the covering area of ASU 1 ($\tilde{d}(i,1) \leq \breve{d}(1), i = 1,2,\ldots,5$). On the other hand, BSU 6, 7, 11, 12, 18, 19, and 20 need the checking of all ASUs as these BSUs are outside the covering area of all ASUs. It means that those BSUs are allocated to their nearest ASU by the classical allocation

method. In other words, we avoided checking 13 out of the 20 BSUs when assigning them to their nearest ASUs (i.e. a 65% time reduction).

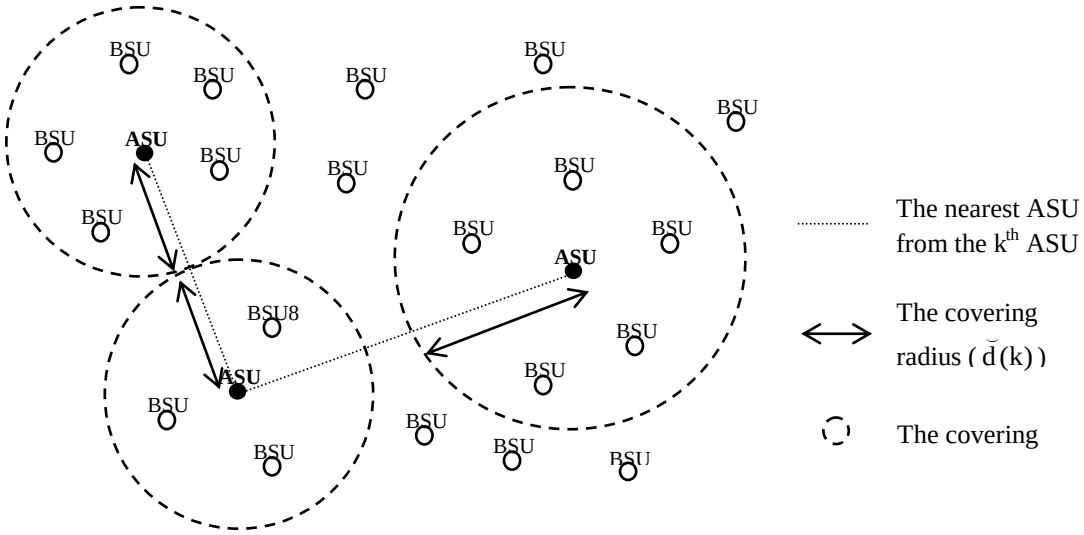


Fig. 3 Illustration of our efficient allocation method

A Small Experiment

We conducted a small experiment to test the performance of our allocation algorithm. In this experiment, *m* clusters are built by allocating *n* demand points to the nearest ASU location (there are *m* ASUs). The BIRCH instances, varying in size from *n* = 25,000 to 89,600, are used. The details of these instances will be given in the Computational Study section. The results show that our allocation algorithm is much faster than the traditional one, requiring around 40% (with at most 50%) of the original time. The summary results are given in Table 2. Bold refers to the solution with the highest deviation value. These show that our algorithm is more than twice faster than the traditional one with an average of 111.8%. This saving in computing time is made significant as our method is an iterative-based approach which avoids to re-compute already computed parts of the distance function namely the γ_i, *i=1,…,n*. The % deviation is defined as:

$\text{Deviation}\,(\%) = 100\left(\frac{\text{TM} - \text{EM}}{\text{EM}}\right)$ where TM and EM refer to the Traditional and the Enhancement methods respectively.

Table 2 Comparison between the Traditional and the Enhancement Methods

Case of BIRCH instances of type 1					Case of BIRCH instances of type 3				
n	m	Allocating Time (millisecond)		Deviation (%)	n	m	Allocating Time (millisecond)		Deviation (%)
		TM	EM				TM	EM	
25,000	2,500	2,744	1,372	100.00	25,000	2,500	2,811	1,366	105.78
36,000	3,600	6,191	2,979	107.82	36,000	3,600	6,450	3,062	110.65
49,000	4,900	11,748	5,509	113.25	49,000	4,900	12,541	5,770	117.35
64,000	6,400	20,657	9,684	113.31	64,000	6,400	21,494	10,081	113.21
30,000	3,000	4,115	1,996	106.16	30,000	3,000	4,230	1,941	117.93
43,200	4,320	9,125	4,282	113.10	43,200	4,320	9,438	4,393	114.84
58,800	5,880	17,284	8,064	114.34	58,800	5,880	18,092	8,352	116.62
76,800	7,680	29,889	14,269	109.47	76,800	7,680	32,003	14,519	**120.42**
35,000	3,500	5,753	2,752	109.05	35,000	3,500	6,107	2,881	111.98
50,400	5,040	12,640	5,826	**116.96**	50,400	5,040	13,464	6,134	119.50
68,600	6,860	22,888	11,015	107.79	68,600	6,860	25,108	11,512	118.10
89,600	8,960	43,883	21,474	104.35	89,600	8,960	44,103	21,941	101.01

TM : Traditional Method

EM : Enhancement Method

3.5. Solving the aggregated problem by the Mini VNS (Step 3a(ii) of Fig. 2)

We solve the aggregated *p*-median problem with the following local search which we call "Mini VNS (*m, p*)". Here, we reduce the demand points and the potential facility locations from *n* points to *m* points. In this step, we do not use the full VNS procedure to solve the aggregated problem as there is no guarantee that the solution with a good objective function in the aggregated problem will also provide a good objective function value in the disaggregated (original) problem. Moreover, finding the best solution with a full VNS in the aggregated problem for all samples $t = 1,\ldots,T$ and all batches $b = 1,\ldots,B$ will be too time consuming. In this study, we adopted a mini VNS made up of one neighbourhood only. This could obviously be considered as a perturbation enhanced by a local search. The full VNS will be presented later as it will be used in Step 3b of Fig. 2.

In the Mini VNS, our local search is based on the *fast interchange heuristic* proposed by Whitaker [19]. We adapted the algorithm to reduce the computing time further at the expense of a small loss in quality. Here, when finding the best demand point to be inserted and to replace facility *j*, we just look for the demand point served by facility *j*. In other words, we restrict the search to the allocated demand points to *j* only. Moreover, when calculating the saving from this swapping process, we just include the demand points served by the neighbouring facilities which we consider to be the α nearest

facilities from facility *j*. In this case, we use α = max{0.1*p*, 10} which was found empirically using a small sample. In other words, we opt for such a neighbourhood reduction as we assume that it is not necessarily worthwhile to swap facility *j* with a demand point which is far from facility *j*. The procedure to find the best demand point to be swapped with facility *j* is given in Fig. 4 which we refer to as "FindBestCustomer".

Procedure FindBestCustomer (*IndexFout*, x_{now}, α, c_{all}, a_1, a_2, *loss*, *BestIndex*, *BestFins*)

- Find the α nearest facilities from facility *IndexFout* in x_{now} (do not include facility *IndexFout*) and store those facilities in x_{near}.
- Determine the demand points in c_{all} served by x_{near} and store them into c_{near}. Determine the customers served by facility *IndexFout*, and then store them into c_{facj}.
- Find the customer in c_{near} to be swapped by using the following procedure.

  For each demand point *i* in c_{facj} do the following:

  If ($i \neq x_{now}$[*IndexFout*]) then
  Set *w* = 0 and *g* = 0 (loss due to the swapping process; negative value = saving)
  For each demand point *z* in c_{facj} do the following:
  If ($d(z,i) < d(z, a_1(z))$) Set $w = w + d(z,i) - d(z, a_1(z))$
  Else
  If ($d(z,i) < d(z, a_2(z))$) Set $g = g + d(z,i) - d(z, a_1(z))$
  Else $g = g + d(z, a_2(z)) - d(z, a_1(z))$
  End If
  End for *z*
  For each demand point *z* in c_{near}
  If ($d(z,i) < d(z, a_1(z))$) Set $w = w + d(z,i) - d(z, a_2(z))$
  $w = w + g$
  If ($w <$ *loss*) Set *loss = w, BestFins = i,* and *BestIndex = IndexFout*
  End If
  End for *i*
- Return *loss, BestIndex, BestFins*.

Fig. 4 The local search procedure "FindBestCustomer"

The Mini VNS algorithm is given in Fig. 5. The shaking is performed by swapping a randomly chosen facility (say facility *j*) with a demand point served by facility *j* which is obtained by the procedure "FindBestCustomer". In this mini VNS, for the first batch, *p* points from *C'* (*m* demand points) are chosen randomly as the initial solution, but for the second batch and onward, we use the best solution from the previous batches. The open facilities ($\mathrm{F_t'}$) are obtained from the t^{th} sample problem, where $\mathrm{F_t'} = (\sigma_1^{\mathrm{t}}, \sigma_2^{\mathrm{t}}, ..., \sigma_{\mathrm{p}}^{\mathrm{t}})$. The objective function, $\mathrm{f_t(F' : C)}$, is calculated using $\mathrm{F_t'}$ on the original problem and then $\mathrm{F_t'}$ is stored into L_b. All duplicate locations are discarded. This set of 'potential' facility locations can be defined as $\mathrm{L_b} = \bigcup_{\mathrm{t}=1}^{\mathrm{T}} \mathrm{F_t'}$.

Step 1 Initialization

(a) For the first batch, p points from C' are chosen randomly as the initial solution. For the second batch and onward, we use the best solution from the previous batches. The initial solution is set as x_{best}.

(b) Calculate the objective function f_{best} using the aggregated data and find arrays a_1 (the nearest facility in x_{best} from each demand point in C') and a_2 (the second nearest facility) by allocating all demand points in C' to the nearest facility in x_{best}.

(c) Set $x_{now} = x_{best}$, $a_{1now} = a_1$, $a_{2now} = a_2$, $f_{now} = f_{best}$, and $\alpha = \max\{0.1p, 10\}$.

Step 2 Shaking

(a) Randomly remove a facility from x_{now} → facility x_{now}[*IndexFout*]

(b) Set *loss* = a big positive number (current best loss; negative value = saving)

(c) Find a facility to be inserted (*BestFins*) by using the procedure "FindBestCustomer (*IndexFout*, x_{now}, α, C', a_{1now}, a_{2now}, *loss*, *BestIndex*, *BestFins*)" which is described in Fig. 4.

(d) Set x_{now}[*IndexFout*] = *BestFins*

(e) Calculate the objective function f_{now} and find arrays a_{1now} (the nearest facility in x_{now} from each demand point) and a_{2now} (the second nearest facility) by allocating all demand points in C' to the nearest facility in x_{now}.

Step 3 Local Search

Do the following steps until Imp=false (i.e. No Improvement)

(a) Set Imp = false and *loss* = big number

(b) For j=1 to p, remove facility j from x_{now} (facility $x_{now}[j]$) and find facility to be inserted (*BestFins*) by using the procedure "FindBestCustomer (j, x_{now}, α, C', a_{1now}, a_{2now}, *loss*, *BestIndex*, *BestFins*)".

(c) Select *BestIndex* and *BestFins* as the swapped facilities (i.e. x_{now}[*BestIndex*] = *BestFins*)

(d) Calculate f_{now} and find arrays a_{1now} and a_{2now} as in Step 2(e).

(e) If ($f_{now} < f_{best}$) then set Imp = true, $x_{best} = x_{now}$, $a_1 = a_{1now}$, $a_2 = a_{2now}$, $f_{best} = f_{now}$.

End Loop Do

Fig. 5 The "Mini VNS" (Step 3a(ii) of Fig. 2)

3.6. Solving the aggregated *p*-median problem with the full VNS (Step 3b of Fig. 2)

At the end of each batch made up of T samples (T solutions/problems), we take all obtained facilities from all solutions (L_b). Note that at the beginning of each batch, the list is empty (i.e. $L_b = \{\emptyset\}$). Each point in L_b acts as an ASU totalling $|L_b|$ ASUs. We then construct $|L_b|$ clusters by allocating all demand points in C to the nearest points in L_b. We calculate the distance between each point in L_b using Current and Schilling [3].

The corresponding p-median problem is solved by VNS ($|L_b|$, p) introduced by Hansen and Mladenovic [10]. We can afford to use a full VNS here as, in total, we will be

needing B runs of VNS only, with $|L_b|$ varying from one batch to another due to duplications. We use the simple VNS version as the size of the *p*-median problem ($|L_b|$, *p*) is relatively small. Let $\mathrm{F_b'}$ be the obtained facility locations and $\mathrm{f_b(F':C)}$ the objective function evaluated using $\mathrm{F_b'}$ on the original problem. In those experiments, we limit the computing time for solving the problem by VNS to T_{vns} seconds which we found empirically to be $\mathrm{T_{vns} = 20n^{0.25}p^{0.5}m^{0.5}/1000}$. In this study, we use $k_{max} = p$.

In Hansen and Mladenovic [10], the set of neighbourhood structures (N_k), k = 1, 2, …, k_{max} is induced by swapping k times a randomly chosen facility with the one in the current solution obtained by the *move* procedure of the *interchange* heuristic introduced by Whitaker [19]. In local search, the *interchange* heuristic was applied. Note that, in [10], they used a best improvement strategy instead of a first improvement as suggested in [19].

3.7. Local Search for the original problem (Step 4 of Fig. 2)

This step acts as an additional post optimisation to solve the disaggregated problem (original problem) starting from the best solution found in the previous batches. This local search is similar to the one in Step 3a(ii) of HMH, except that we do not construct the distance matrix for all demand points, but those points used in the "FindBestCustomer" procedure only. In other words, we do not require a large amount of memory capacity. The procedure of our local search for the disaggregated problem is given in Fig. 6.

Initialization

- Take the best solution from the previous batches as the initial solution, and store it as x_{best}.
- Calculate the objective function f_{best} and find arrays a_1 (the nearest facility in x_{best} from each demand point) and a_2 (the second nearest facility) by allocating all demand points (C) to the nearest facility in x_{best}.
- Set $x_{now} = x_{best}$, $a_{1now} = a_1$, $a_{2now} = a_2$, $f_{now} = f_{best}$, and $\alpha = \max\{0.1p, 10\}$.

Local Search

Apply the local search of Step 3 in Fig. 4, but using C (original demand points) instead of C' (aggregated demand points).

Fig. 6 The Local Search Procedure for Solving the Original Problem

4. Computational Study

A computational study to assess our solution method was conducted. The project developed to solve the problems is written in C++ .Net 2010. The specification of the computer used to execute all problems is a PC Intel Core i5 CPU 650@ 3.20GHz of processor, 4.00 GB of RAM and under Windows 7(32bit). In our computational study, we use $m = 0.1n$, B = 5, and T = 10. Those parameters were chosen based on a small preliminary study. The value of m is found to be large enough to represent the original problem while being small enough to be solved. This choice provides an acceptable performance for both the quality of the solution and the computational effort. The value of B and T are also chosen to produce a reasonable number of runs and batches. Also, in our preliminary study it was observed that the improvement of the solution was getting relatively small from the 5^{th} batch onward. For instance, on the 2^{nd} batch onward, the rate of improvement of the solution starts to reduce. A larger value of T ($T \geq 10$) though may increase the chance in obtaining a better solution in each batch but requires an increase in the computing time while making the use of the full VNS less attractive due to the excessive time required. An extensive testing using a statistical analysis could be conducted to provide better estimates for m, B, and T if necessary. In this study, to be consistent when repeating the experiments, the seed for the random generator is set to a constant, here we set it to m.

Two types of instances are used in our computational experiments. The first set consists of the BIRCH instances kindly provided by Avella et al. [1] in http://iv.icc.ru/Papers.hatml (n ranges from 25,000 to 89,600). According to [1], these instances are the largest instances tested in the literature. The second set contains the TSP instances from http://www.tsp.gatech.edu/world/countries.html. The instances of Italy, Sweden, Burma, and China (n ranges from 16,862 to 71,009) are used in our testing. The results in both cases are summarised in Tables 3 and 4 respectively. For completeness, we also report the solution with and without Step 4 of Fig. 2.

Case 1: BIRCH instances

The performance of our method is compared with the one of AH short for Avella et al. [1] and the VNSH short for the VNS of Hansen et al. [9]. The computational results of both AH and VNSH methods are taken from [1]. Computational experiments for AH and VNSH were carried out by Avella et al. on an Intel Core 2Quad CPU 2.6 GHz, 4.00 GB of

RAM and under Windows XP64. To provide a fair comparison in terms of CPU, we use the following transformation as given by Dongarra [4] with $T_2 = T_1 \frac{Nf_1}{Nf_2}$, where T_1 represents the reported time in Machine 1 and T_2 the estimated time in Machine 2. Nf_1 and Nf_2 denote the number of Mflops in Machines 1 and 2 respectively. The software used to record the values of Nf_1 and Nf_2 can be downloaded from http://www.roylongbottom.org.uk. As we could not obtain precisely the number of Mflops of the computer used by Avella et al. [1], we provide an approximation based on a slightly slower but similar computer available to us namely a PC Intel Core 2Duo 2.6GHz, 4 GB of RAM.

Table 3 shows the computational results for our method (HMH) on the BIRCH instances. The first three columns refer to the problem name, the number of demand points, and the number of medians. The next three blocks of 4 columns each refer to the objective function value (Z), the CPU time in seconds and the deviation (in %). The latter measure is defined as:

$\text{Deviation} = 100\left(\frac{Z_c - Z_b}{Z_b}\right)$, where Z_c and Z_b correspond to the Z value for the corresponding method *'c'* and the best Z value respectively.

Method *'c'* refers to VNSH, AH, HMH, and HMH$^-$ (i.e. HMH without the post optimisation, Step 4 of Fig. 2). 'Bold' shows the new best solutions. The results demonstrate that HMH provides better solutions compared to AH on all these instances. HMH and VNSH produce similar objective function values on the BIRCH instances of type 1, but HMH yields a slightly better deviation (0.0113%). The proposed approach also produces 22 out of 24 new best solutions in these BIRCH instances. Note that our heuristic still outperforms the others even without the post optimisation step (Step 4 of Fig. 2, i.e. HMH$^-$).

In Avella et al. [1] the upper bound of VNSH on the BIRCH instances of type 3 are not reported. On these instances, our method (with and without using the post optimisation) is better than AH. The post optimisation in HMH only improves slightly on the solution quality (0.0028%) while requiring a lot of extra computing time (100(827.15-740.52)/740.52 = 11.7%).

Table 3 Computational Results for the HMH method on BIRCH instances

File Name	n	p	Z					Time				Deviation (%)			
			VNSH	AH	HMH	HMH–		VNSH	AH	HMH	HMH–	VNSH	AH	HMH	HMH–
BIRCH instances of type 1															
1	25,000	25	31,363.6	31,282.6	**31,229.3**	31,229.4		206	447	157	147	0.430	0.171	0.000	0.000
2	36,000	36	**45,115.6**	45,226.3	**45,115.6**	**45,115.6**		590	780	373	366	0.000	0.245	0.000	0.000
3	49,000	49	**61,384.1**	61,569.7	**61,384.1**	61,385.7		818	1,216	612	582	0.000	0.302	0.000	0.003
4	64,000	64	**79,987.3**	80,337.4	80,053.9	80,054.1		1,527	2,258	1,110	1,078	0.000	0.438	0.083	0.083
5	30,000	25	37,564.1	37,617.1	**37,563.6**	**37,563.6**		321	559	223	215	0.001	0.142	0.000	0.000
6	43,200	36	**54,191.4**	54,305.8	**54,191.4**	54,192.5		767	1,003	434	403	0.000	0.211	0.000	0.002
7	58,800	49	**73,626.8**	73,854.7	**73,626.8**	73,627.4		1,454	1,691	792	748	0.000	0.310	0.000	0.001
8	76,800	64	**95,989.1**	96,393.4	96,039.4	96,040.1		2,931	2,834	1,510	1,414	0.000	0.421	0.052	0.053
9	35,000	25	**43,902.1**	43,972.1	**43,902.1**	**43,902.1**		569	768	353	342	0.000	0.159	0.000	0.000
10	50,400	36	**63,169.2**	63,329.2	**63,169.2**	**63,169.2**		1,185	1,472	645	629	0.000	0.253	0.000	0.000
11	68,600	49	85,833.6	86,082.0	**85,833.5**	85,833.6		1,787	2,441	1,149	1,083	0.000	0.289	0.000	0.000
12	89,600	64	**112,059.2**	112,485.2	**112,059.2**	**112,059.2**		3,678	4,501	2,069	2,002	0.000	0.380	0.000	0.000
	# best Z		9	0	**10**	5	Average	1,319.42	1,664.17	**785.62**	**750.87**	0.0360	0.2769	**0.0113**	0.0119
BIRCH instances of type 3															
21	25,000	25		17,718.6	**17,696.2**	**17,696.2**			527	125	120		0.127	0.000	0.000
22	36,000	36		27,476.1	**27,423.0**	27,423.1			913	365	350		0.193	0.000	0.000
23	49,000	49		44,282.5	**44,202.3**	44,202.4			1,760	526	496		0.181	0.000	0.000
24	64,000	64		58,991.5	**58,902.3**	58,903.0			2,624	1,049	919		0.151	0.000	0.001
25	30,000	25		21,865.1	**21,829.9**	**21,829.9**			832	454	447		0.161	0.000	0.000
26	43,200	36		32,391.6	**32,339.4**	32,339.8			1,873	492	428		0.161	0.000	0.001
27	58,800	49		50,985.1	**50,857.9**	**50,857.9**			2,692	899	885		0.250	0.000	0.000
28	76,800	64		66,944.7	**66,741.8**	66,759.0			4,393	1,892	1,441		0.304	0.000	0.026
29	35,000	25		24,833.7	**24,811.0**	24,811.5			972	288	257		0.091	0.000	0.002
30	50,400	36		38,162.3	**38,102.7**	**38,102.7**			2,297	611	581		0.157	0.000	0.000
31	68,600	49		62,007.4	**61,882.4**	61,882.5			3,556	1,035	1,015		0.202	0.000	0.000
32	89,600	64		79,245.3	**78,777.5**	78,779.2			5,779	2,189	1,948		0.594	0.000	0.002
	# best Z			0	**12**	4	Average		2,351.50	**827.15**	**740.52**		0.2145	**0.0000**	0.0028
						T1			2,351.50	827.15	740.52				
						CPU (Mflops)			3,545.00	4,415.00	4,415.00	using 32 bit SSE MFLOPS			
						T2			1,888.12	**827.15**	**740.52**				

HMH– : HMH without post optimisation (i.e. solving the diaggreged p-median problem by local search)

Table 3 also shows that HMH is a rather fast hybrid multi-stage heuristic. Based on Dongarra's approach, the computer used to execute our method is more or less 25% faster than the one used by Avella et al. [1]. Based on the transformed computing time (T_2 in Table 3), our method is found to be more than twice as fast as the one by Avella et. al. [1] while generating relatively better results.

Case 2: TSP instances

Table 4 shows the computational results for the HMH method on the TSP instances. Each instance is solved with *p* varying from 25 to 100 with an increment of 25, totalling 16 instances. When *p* is large, the problem becomes more complex and hence HMH needs more time to solve the problem. There is even a more serious handicap when *p* is very large and gets closer to the number of ASU (*m*). Further research is needed to find the best ratio between *m* and *p*. Conducting the local search on the original problem also improves the objective function value by a tiny fraction of 0.0971% on average at the expense of a massive average extra computing time (37.29%). However, this extra step seems to produce better solutions on all the 16 instances which can be used for further benchmarking if necessary.

Table 4 Computational Results for the HMH method on TSP instances

File Name	Description	n	p	Z			CPU Time (seconds)		
				HMH	HMH⁻	Deviation (%)	HMH	HMH⁻	Deviation (%)
41	Italy Data	16,862	25	**7,411,193.07**	7,412,970.62	0.0240	96.82	67.36	43.7277
41	Italy Data	16,862	50	**5,110,772.87**	5,119,564.37	0.1720	119.01	82.97	43.4384
41	Italy Data	16,862	75	**4,088,051.12**	4,095,381.98	0.1793	233.80	183.17	27.6422
41	Italy Data	16,862	100	**3,492,806.69**	3,501,633.36	0.2527	350.33	280.84	24.7415
42	Sweden Data	24,978	25	**14,098,813.68**	14,124,941.97	0.1853	348.43	177.92	95.8308
42	Sweden Data	24,978	50	**9,667,897.85**	9,674,202.31	0.0652	278.59	197.02	41.4023
42	Sweden Data	24,978	75	**7,783,165.38**	7,794,498.86	0.1456	450.64	333.29	35.2096
42	Sweden Data	24,978	100	**6,677,281.78**	6,691,055.51	0.2063	637.66	472.94	34.8287
43	Burma Data	33,708	25	**18,227,047.70**	18,228,677.34	0.0089	418.36	346.53	20.7278
43	Burma Data	33,708	50	**12,603,717.78**	12,607,345.85	0.0288	479.49	354.08	35.4174
43	Burma Data	33,708	75	**10,205,835.93**	10,212,849.31	0.0687	651.47	452.53	43.9639
43	Burma Data	33,708	100	**8,748,238.35**	8,757,631.56	0.1074	1,107.66	770.33	43.7913
44	China Data	71,009	25	**113,812,852.56**	113,816,994.15	0.0036	2,227.85	1,942.25	14.7043
44	China Data	71,009	50	**78,633,618.28**	78,656,979.00	0.0297	2,901.61	1,929.33	50.3948
44	China Data	71,009	75	**64,026,411.05**	64,056,314.76	0.0467	3,217.66	2,217.52	45.1019
44	China Data	71,009	100	**54,870,043.92**	54,885,909.63	0.0289	4,030.75	2,975.09	35.4832
			Average	26,216,109.25	26,227,309.41	0.0971	1,096.88	798.95	37.2908

HMH⁻ : HMH without post optimisation (i.e. solving the diaggreged p-median problem by local search)

5. Conclusion

In this paper, we propose an approach to solve the largest *p*-median problems from the literature based on data aggregation and the use of an efficient implementation of VNS. This is a multi-stage hybrid approach that uses sampling based on aggregation, a fast procedure to find good solutions, a powerful VNS used on the promising facilities, and a learning process that feeds information from one stage to another. The computational results show that our approach performs considerably well as it produces relatively very good solutions, finds a large number of best solutions (22 out of 24), and runs quite fast.

The proposed approach was tested on the BIRCH instances (n = 25,000 to 89,600) and compared to the ones by Avella et al. [1] and Hansen et al. [9]. The results show that our method gives better solutions compared to the ones by Avella et al. [1] and relatively similar to the ones by Hansen et al. [9]. In addition, we also assess our method on several new large TSP instances (n = 16,862 to 71,009) that were not tested before. Each instance is solved with p varying from 25 to 100 with an increment of 25 with encouraging results. These can be used for benchmarking purposes in the future.

The proposed methodology of aggregation and optimisation can be extended to include, in certain steps, more powerful VNS implementations and exact methods based on reduced formulation whenever found possible. This study could be adapted to explore other related location problems such as large multisource Weber problems and also large *p*-center problems both in the continuous and in the discrete spaces.

Acknowledgment

The authors would like to thank both referees for their useful suggestions that improved both the content as well as the presentation of the paper.

References

1. Avella, P., Boccia, M., Salerno, S., Vasilyev, I.: An aggregation heuristic for large scale *p*-median problem. Comput. Oper. Res. 39(7), 1625-1632 (2012)

2. Casillas, P.: Aggregation problems in location-allocation modeling. In: Gosh, A., Rushton, G. (eds.) Spatial analysis and location-allocation models, pp. 327-344. Van Nostrand Reinhold, New York (1987)

3. Current, J. R., Schilling, D. A.: Elimination of source A and B errors in *p*-median location problems. Geogr. Anal. 19, 95-110 (1987)

4. Dongarra, J. J.: Performance of various computers using standard linear Equation software. http://www.netlib.org/benchmark/performance.pdf (Accessed online 15 April 2013)

5. Erkut, E., Bozkaya, B.: Analysis of aggregation errors for the *p*-median problem. Comput. Oper. Res. 26, 1075-1096 (1999)

6. Francis, R. L., Lowe, T. J., Rayco, M. B.: Row-column aggregation for rectilinear distance *p*-median problems. Transp. Sci. 30, 160-174 (1996)

7. Francis, R. L., Lowe, T. J., Rayco, M. B., Tamir, A.: Aggregation error for location models: survey and analysis. Ann. Oper. Res. 167, 171-208 (2009)

8. Garcia, S., Labbe, M., Marin, A.: Solving large *p*-median problem with a radius formulation. INFORMS J. Comput. 23, 546-556 (2010)

9. Hansen, P., Brimberg, J., Urosevic, D., Mladenovic, N.: Solving large *p*-median clustering problems by primal-dual variable neighborhood search. Data Min. Knowl. Discov. 19, 351-375 (2009)

10. Hansen, P., Mladenovic, N.: Variable neighbourhood search for the *p*-median. Locat. Sci. 5, 207-225 (1997)

11. Hillsman, E. L., Rhoda, R.: Errors in measuring distances from populations to service centers. Ann. Reg. Sci. 12, 74-88 (1978)

12. Hodgson, M. J.: Data surrogation error in *p*-median models. Ann. Oper. Res. 110, 153-165 (2002)

13. Hodgson, M. J., Neuman, S.: A GIS approach to eliminating source C aggregation error in *p*-median models. Locat. Sci. 1, 155-170 (1993)

14. Hodgson, M. J., Shmulevitz, F., Körkel, M.: Aggregation error effects on the discrete-space *p*-median model: The case of Edmonton, Canada. Can. Geogr. 41, 415-428 (1997)

15. Papadimitriou, C. H.: Worst-case and probabilistic analysis of a geometric location problem. SIAM J. Comput. 10, 542-557 (1981)

16. ReVelle, C.S., R. Swain.: Central Facilities Location. Geogr. Anal. 2, 30–42 (1970)

17. Qi, L., Shen, Z. M.: Worst-case analysis of demand point aggregation for the Euclidean *p*-median problem. Eur. J. Oper. Res. 202, 434-443 (2010)

18. Salhi, S., Gamal, M. D. H.: A genetic algorithm based approach for the uncapacitated continuous location–allocation problem. Ann. Oper. Res. 123, 203-222 (2003)

19. Whitaker, R. A.: A fast algorithm for the greedy interchange for large-scale clustering and median location problems. INFOR 21, 95-108 (1983)

20. Zoubi, M. B., Rawi, M.: An efficient approach for computing silhouette coefficients. J. Comput. Sci. 4, 252-255 (2008)