A quadtree-based allocation method for a class of large discrete Euclidean location problems*

[1]Said Salhi and [2,3]Chandra Ade Irawan

[1]Centre for Logistics & Heuristic Optimization (CLHO), Kent Business School, University of Kent, Canterbury, Kent CT2 7PE, UK

[2]Centre for Operational Research and Logistics (CORL), Department of Mathematics, University of Portsmouth, Lion Gate Building, Lion Terrace, Portsmouth, PO1 3HF, UK

[3]Department of Industrial Engineering, Institut Teknologi Nasional, Bandung, 40124, Indonesia

Abstract A special data compression approach using a quadtree-based method is proposed for allocating very large demand points to their nearest facilities while eliminating aggregation error. This allocation procedure is shown to be extremely effective when solving very large facility location problems in the Euclidian space. Our method basically aggregates demand points where it eliminates aggregation-based allocation error, and disaggregates them if necessary. The method is assessed first on the allocation problems and then embedded into the search for solving a class of discrete facility location problems namely the *p*-median and the vertex *p*-centre problems. We use randomly generated and TSP datasets for testing our method. The results of the experiments show that the quadtree-based approach is very effective in reducing the computing time for this class of location problems.

Key words: Allocation method, quadtree method, *p*-median and *p*-centre problems, aggregation

1. Introduction

It is a common practice when dealing with large location problems to aggregate demand points known as Basic Spatial Units (BSUs) into a small number of Aggregated Spatial Units (ASUs). Such an aggregation usually leads to error due to both distance measurement and allocation. Many of the aggregation schemes are iterative processes where the allocation must be performed several times to find the best solutions having the least errors. A two-phase approach is commonly used where in the first phase an aggregated (smaller) problem is constructed by solving a clustering problem, and in the second phase the aggregated location-allocation problem is then solved. It was noted that the design of aggregation schemes that minimize aggregation error is itself a hard problem which has not yet been solved

* This study was mostly conducted while the second author was at the University of Kent

successfully for a large number of demand points, see Francis and Lowe (1992) and Francis *et al.* (2003, 2009).

Hillsman and Rhoda (1978) introduced three sources of error arising from demand point aggregation, known as source ABC error. Source A error happens when the distance between an ASU and a facility is applied in the model instead of the true distance between a BSU and a facility. Source B error appears in the special case when a facility is located at an ASU whereas source C error occurs when a BSU is assigned to the wrong facility. Several schemes are introduced to reduce or eliminate these types of aggregation error which are usually grouped under two categories. These include data manipulation and aggregation design which are briefly described next. For more details on aggregation methods for location problems, see Francis *et al.* (2009).

Data manipulation

Current and Schilling (1987) proposed pre-processing the demand data. Their method eliminates source A and B errors by assigning the correct total weight BSU–facility distance to each ASU–facility cell in the weighted distance matrix. The method does not address source C error. To eliminate source C error, Hodgson and Neuman (1993) utilise continuous space, a set of Voronoi polygons and a GIS overlay procedure to "aggregate on the fly". Their method though is successful in addressing source C error, it fails to eliminate source A and B errors. Hodgson *et al* (1997) introduced a new type of error known as source D error. This occurs if some of the BSU locations happen to be at the potential sites. Bowerman *et al* (1999) introduced a demand portioning method that applies the Current and Schilling (1987) approach to eliminate source A and B errors while producing ASUs on the fly when using a vertex interchange procedure to eliminate source C error. Hodgson and Hewko (2003) studied aggregation and surrogation errors for the *p*-median problem using Edmonton, Canada data. The authors showed that the surrogation error was more a serious problem than the aggregation error.

Aggregation Zone Design

Francis and Lowe (1992), Francis *et al* (1996, 2000, 2003, 2009), and Andersson *et al.* (1998) dealt with aggregation error by developing aggregation zones for which error bounds can be determined. Their methods established rectangular zones which can be long and narrow, and hence prone to aggregation errors. Erkut and Bozkaya (1998) empirically

evaluated some aggregation methods. A primal-dual VNS metaheuristic for large *p*-median clustering problems was proposed by Hansen *et al*. (2009) where a Reduced VNS is used to get good initial solutions which are then fed into a VNS with decomposition. Qi and Shen (2010) investigated the worst-case analysis of demand point aggregation for the Euclidean *p*-median problem on the plane. García *et al*. (2010) developed an alternative covering based formulation which has a small subset of constraints and variables. This method is shown to be more efficient especially when *p* is relatively large. Avella *et al*. (2012) proposed an aggregation heuristic based on Lagrangean relaxation for large scale *p*-median problem that produced excellent results. Very recently Irawan and Salhi (2013) and Irawan *et al.* (2014) developed a multi-phase approach by solving a series of subproblems either optimally or heuristically where the obtained facility locations are then used as promising potential sites. Competitive results were generated when compared to the best known solutions. For more details on aggregation error measurements and papers dealing with aggregation to location problems, the reader will find the excellent survey paper by Francis *et al*. (2009) informative and very valuable. The authors also point out effective as well as ineffective errors measures.

The process of determining an aggregation scheme with a minimum error is an NP-hard problem, see Francis and Lowe (1992). This difficulty has led us to develop a method where we do not aggregate demands by designing a general aggregation of demand points but we conduct aggregation with reference to a specific set of given facilities. In the location-allocation context, the method would aggregate demand relative to *p* trial facilities as they arise during the search as usually applied in heuristics and metaheuristics.

The main contribution of this paper is the development of an effective quadtree method (QM) used for allocating the demand points to their nearest facility when solving a class of large Euclidean discrete location problems. This allocation technique could easily be incorporated in those recent powerful algorithms for large-scale location problems as this mechanism could enhance their efficiency even further.

The paper is organized as follows. A brief on the quadtree method is given in Section 2 followed by a quadtree-based methodology in Section 3. The computational results, comparing QM against the classical allocation methods, are presented in the fourth section. The integration of QM in solving both the discrete *p*-median and the *p*-centre problems is attempted in the fifth section. The last section provides a summary of our findings and highlights some suggestions for future research.

2. A brief on the quadtree method

This section demonstrates an efficient aggregation scheme that eliminates all types of allocation aggregation errors. The main idea of the scheme is adopted from the presentation given at the INFORMS conference in Montreal (1998) by Hodgson and Salhi. The method utilised a spatial data compression, known as quadtrees (Samet, 1990), to partition the study area. The demand points could obviously also be partitioned by Voronoi polygons. Figure 1 shows a Voronoi polygons scheme with a number of demand points and three facilities (p = 3). Hodgson and Neuman (1993) used this method to eliminate source C error. Noaves *et al*. (2009) also adopted Voronoi polygons for solving continuous location-districting problems.

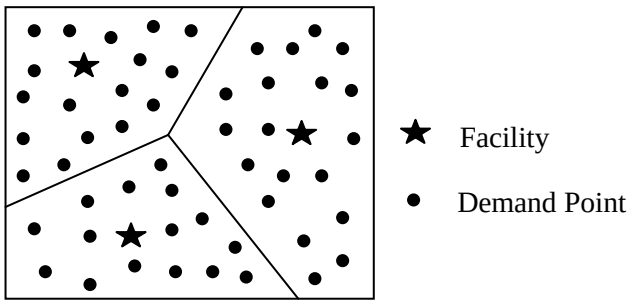


Figure 1. A Voronoi polygons scheme

Though the Voronoi polygons-based allocation can be an efficient method to allocate demand points to facilities, one of the limitations is that for each new set of facilities, the new set of polygons must be generated which can be time consuming. The quadtree data structure, inspired from a raster Geographic Information System (GIS), is adapted to overcome this difficulty. The heart of QM is to pre-generate an appropriate set of common polygons with which we can systematically allocate spatial grouping of demand points to their common closest facility until all demand points have been allocated. A hierarchical organization of successively generating smaller spatial groupings is required to eliminate all aggregation errors.

A map is partitioned by raster GIS into a tessellation of square grid cells called pixels. Each pixel has its attributes, usually by assigning a number. For example a land use map might utilise 1 for green area, 2 for water, 0 for no data, and so on. Figure 2 shows an illustration of a quadtree system where a raster grid is partitioned into a hierarchy (tree) of quadrants.

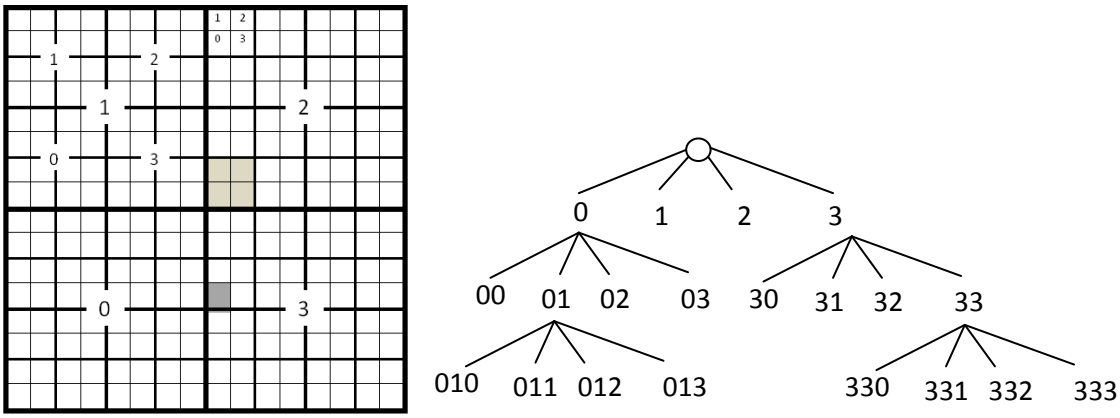


Figure 2. Quadtree partitioning and numbering system

The quadtree system first partitions the map into four quadrants, each quadrant assigned with a single digit between 0 and 3. Then, each of these quadrants is then partitioned into four quadrants, each address with a second such digit. This procedure continues until a certain number of levels where each successive partition is assigned its corresponding digit between 0 and 3. Figure 2 also presents the numbering system and the quadtree partitioning. The figure shows that the lightly shaded patch of four grid cells is assigned 200 whereas the darker grid cell is addressed 3100.

In GIS, the quadtrees are usually utilized to capture areas with the same data characteristic or attribute. Many adjacent pixels may have the same attribute in a rasterized map. In the location-allocation problem, we develop a method that adapts the quadtree structure to capture areas with the same spatial attribute (i.e., areas that are entirely closer to one facility than to any other). The number of allocations is significantly reduced by quadtrees. Figure 3 shows how the quadtree structure deals with a location-allocation problem. There are 32 x 32 raster and each pixel is to be allocated to the closest of the three facilities. There is also the Voronoi polygon to recognize the correct allocation. Let L represent the quadtree level, with 0 denoting the original undivided study area.

Table 1 shows the result of an example in Figure 3. At level 1, the entire quadrant 0 (16x16 pixels) is closer to one facility than to any other. Its entirety can be allocated to that facility, it means that 256 pixels are aggregated and allocated accurately at once. Four level 2 quadrants (aggregations) are each allocated to the closest facility; in other word 256 pixels are accurately allocated. At level 3, sixteen quadrants assign another 256 pixels. Three quarters of the study area's 1024 pixels has now been accurately allocated by using 21 quadrants at the top three levels. Thirty two quadrants (128 pixels) are each allocated to the

closest facility at level 4. Finally, at level five, 72 pixels can be allocated leaving 56 split only.

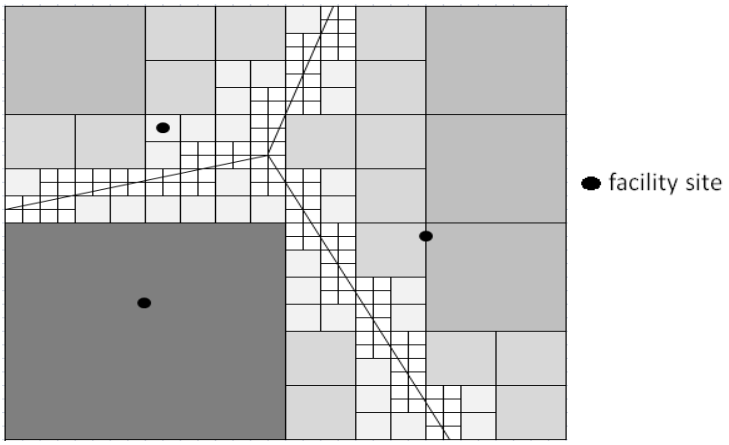


Figure 3. Quadtree partitioning of Voronoi allocation areas

Table 1. Allocation of pixels by the quadtree level for the example problem (case of p=3)

Level	Pixels per quadrant	Entire quadrants	Pixels Allocated	Cumulative allocation (%)
1	256	1	256	25.00
2	64	4	256	50.00
3	16	16	256	75.00
4	4	32	128	87.50
5	1	72	72	94.53
	<1	56	56	100.00
	Total	**181**	**1024**	

At some level, the limited amount of aggregation error remaining may be accepted by assigning all contained demand points to a single or several nearby facilities. To eliminate aggregation error, each demand point in the split quadrants is allocated to their nearest facilities. In this example, 125 quadrants of varying sizes have been allocated, but only 56 of the entire grid of the 1024 pixels are left unassigned and which need to be assigned in the classical manner. It means that we have allocated 94.53 percent (100·[1024-56]/1024) of the pixels with only 5.47 percent (100·56/1024) of the number of allocations left to be allocated using the classical allocation. Moreover, to allocate 100 percent of the pixels, we need 17.68 percent (100·181/1024) of the number of allocations. It is therefore interesting to determine the right balance between the number of levels and the percentage of left over demand points to allocate in the classical way.

The quadtree structure partitions the study area by ignoring the distribution of the demand points. By allocating quadrants accurately, the demand points contained in them are

also allocated accurately. For these demand points, source C (allocation) error is totally eliminated. To eliminate the other errors (source A and B error/distance measurement error), the total weighted distance from all demand points in a quadrant must be determined, but only to the single, known closest facility, not to all facilities. This method creates an allocation process which is more efficient as it reduces the computational effort/time of allocating demand points to facilities. However, it depends on a quadtree structure with a well-defined knowledge of which demand points belong to each quadrant at each level. Therefore, this method clearly needs considerable computational overhead. The computational success of the quadtree method relies on the relationship between the time saved due to the allocation and the overhead times/cost. In the following section, algorithm procedures and reduction tests that aim to reduce the amount of computational overhead required to perform the allocations are presented.

3. The quadtree-based methodology

The quadtree method (QM) consists of two phases namely (i) the construction of the quadtree database and (ii) the allocation procedure. For any set of demand points, a database that defines the quadtree's hierarchical structure is created and then demand points are allocated to quadrants at each level of the hierarchy. The quadtree database needs to be calculated only once for a given set of demand points. For any new set of facilities, full quadrants of demand points must be allocated to the closest of these facilities.

3.1 The construction of the quadtree database

We propose the following steps to construct the quadtree database. These include the construction of the area of study, the definition of the corners of the quadrants, the building of the quadtree structure, and finally the allocation of demand points to the quadtrees.

1) Design of the Demand Point Area

We define a rectangular study area of known dimensions that contains all the demand points and all the potential facilities. Let n be the number of demand points and (X_i, Y_i) be the coordinates of the i^{th} demand point, where $i = 1,...,n$. Then let $X_a = \underset{i}{Min}(X_i)$ and

$X_b = \underset{i}{Max}(X_i)$ similarly $Y_a = \underset{i}{Min}(Y_i)$ and $Y_b = \underset{i}{Max}(Y_i)$. These X_a, Y_a, X_b and Y_b values form the key border points of our rectangular demand study area with lower left and upper right corners are being defined by (X_a, Y_a) and (X_b, Y_b) respectively.

2) Building corner objects

To establish the corner objects defining the corners of the quads, the quadtree level L needs to be set. The number of corners (N_L) required for a given quadtree with L levels is

$$N_L = (2 + \sum_{i=0}^{L-1} 2^i)^2 \tag{1}$$

For instance, if L = 2, the number of corners is 25 and these are (cor[0], cor[1], ... , cor[24]). The horizontal and the vertical distance between consecutive corners can be written as $\delta x = \frac{X_b - X_a}{2^L}$ and $\delta y = \frac{Y_b - Y_a}{2^L}$ respectively. As the values of X_a, Y_a, δx, and δy are obtained, $(\tilde{x}_k, \tilde{y}_k)$, the *x* and *y* coordinate of the k^{th} corner object can be determined. At any level, a quad's location can be derived using the coordinates of its lower left corner, cor[0] = (X_a, Y_a), and its upper right corner, cor[N_L-1] = (X_b, Y_b). Given the *x* and *y* values representing $(\tilde{x}_k, \tilde{y}_k)$, the value of *k* can be determined from:

$$k = \frac{\tilde{x}_k - X_a}{\delta x} + \tilde{N}_L \frac{\tilde{y}_k - Y_a}{\delta y} \tag{2}$$

where $\tilde{N}_L = 2 + \sum_{i=0}^{L-1} 2^i = \sqrt{N_L}$ represents the total number of corners in a row.

Similarly $\tilde{x}_k$ and $\tilde{y}_k$ can be determined as $\tilde{x}_k = X_a + \delta x(k \bmod (\tilde{N}_L))$ and $\tilde{y}_k = Y_a + \delta y \left\lfloor \frac{k}{\tilde{N}_L} \right\rfloor$.

For illustration, consider Figure 4 with L = 2 and $\tilde{N}_L = 5$. Also for simplicity take $\delta x = \delta y = 1$ and (X_a, Y_a)=(0,0) and (X_b,Y_b)=(4,4). Here (2) reduces to $k = \tilde{x}_k + \tilde{N}_L \tilde{y}_k$. As an example, consider *k* = 17 which leads to

$\tilde{x}_k = 0 + 1(17 \bmod 5) = 2;\ \tilde{y}_k = 0 + 1\left\lfloor \frac{17}{5} \right\rfloor = 3$ and $cor[17] = (2,3)$.

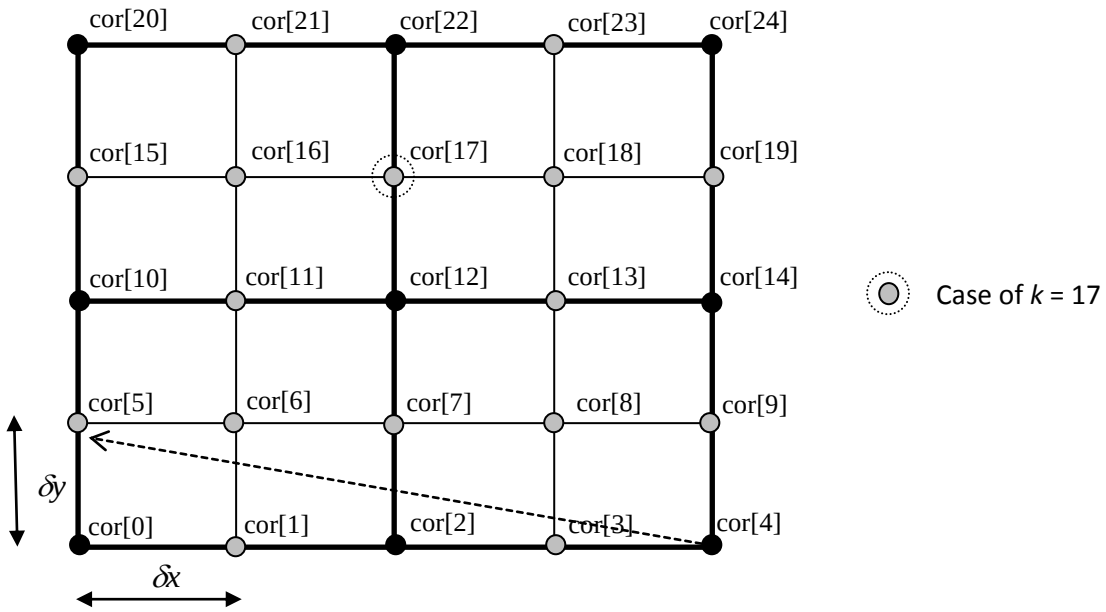


Figure 4. An example of corner objects for L = 2

3) Building the quadtree structure

To build the quadtree structure, we first determine the number of quads at each level $(Q_l, l = 1,...,L)$ which can be formulated as $Q_l = 4^l$.

Initially (level 0) we have a rectangle with vertices (X_a,Y_a), (X_a,Y_b), (X_b,Y_b), and (X_b,Y_a), then to obtain the quads at level 1 this rectangle is then divided into four equal size quads. To determine the location of these quads, we just need to find out the coordinate of the middle point of this rectangle which is $X_m = \frac{X_a + X_b}{2}$ and $Y_m = \frac{Y_a + Y_b}{2}$ (see Figure 5).

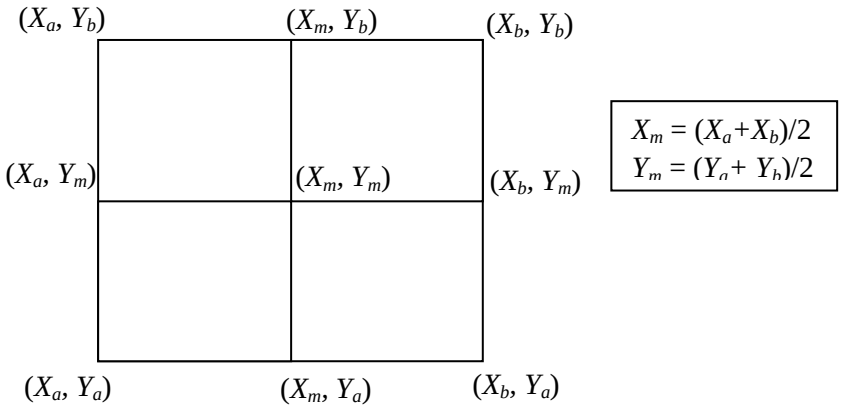


Figure 5. An illustration of how the quads at level 1 are developed

At level 2, each quad is also divided into four equal size quads totaling 16 quads. This process continues until level L.

In this method, each quad is assigned with a double digit L-O, where L is the level and O represents the quad number put in order. In this method, the numbering system is slightly different with the original quadtree system. In level 1, there are 4 quads (1-0, 1-1, 1-2, and 1-3) whereas in level 2 there are 16 quads (2-0, ..., 2-15). The i^{th} quad at the higher level L_H consists of the p^{th} until the q^{th} quad at the lower level L_L, where $p = 4^{(L_L - L_H)} i$ and $q = 4^{(L_L - L_H)} i + 4^{(L_L - L_H)} - 1 = 4^{(L_L - L_H)}(1+i) - 1$.

Figure 6 shows how to arrange the quadtree structure for $L = 2$ where the quads at level 2 are derived from the quads at level 1.

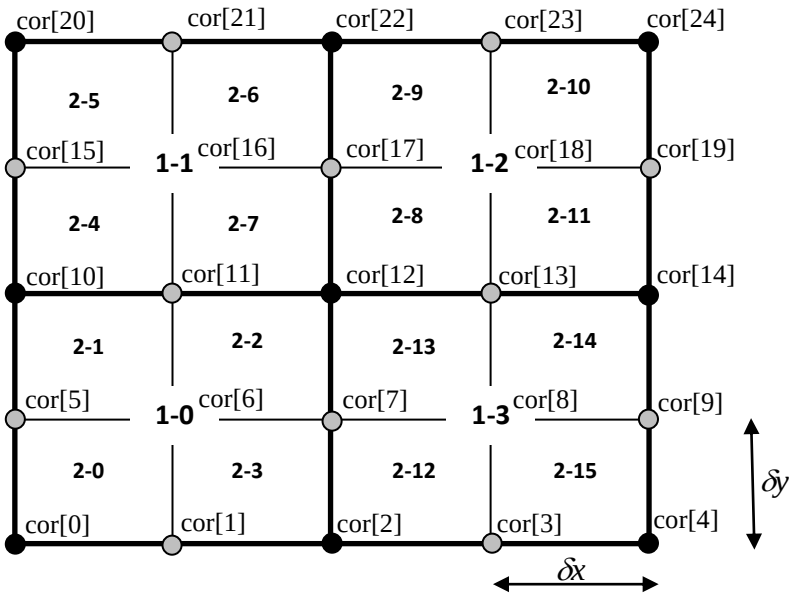


Figure 6. An example of a quadtree structure for L = 2

For example quads 2-4, 2-5, 2-6, and 2-7 are built from quad 1-1. To develop these quads, the lower left corner coordinate of quad 1-1 (cor[10]) = (X_a, Y_a) and the upper right corner coordinate (cor[22]) = (X_b, Y_b) are used. The midpoint defined by X_m and Y_m is then computed. This construction continues until all of the quad locations, from level 1 until level L, are determined.

4) The allocation of the demand points to the corresponding quadtrees

In this process, the quadtree is 'populated' by assigning demand points to quads at each level. This process uses a bottom-up procedure. At the lowest level (level L), each demand point is assigned to the correct quad. At the same time, we create a list of all demand points (put in order from the 0^{th} quad until the $(4^L\text{-}1)^{\text{th}}$ quad) and a list of the number of demand points for

each quad (put in order from the 0^{th} quad until the $(4^L-1)^{th}$ quad). Let θ and ν be two vectors denoting the list of all demand points and the list of the number of demand points for each quad respectively. For instance consider the example of 30 demand points as shown in Figure 7. Here, we have θ = {1,16,21,3,10,19,5,20,4,6,7,13,17,22,23,8,15,12,14,25,26,9,29, 28,27,30,18,2,11,24} and ν = {3,3,2,4,3,2,0,2,2,2,1,2,1,2,1,0}.

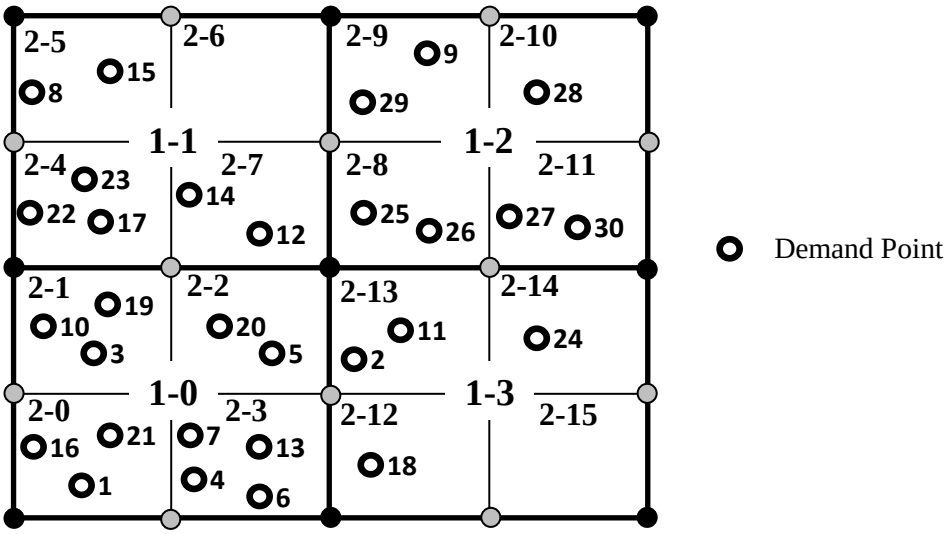


Figure 7. An example of 30 demand points located in a quadtree structure (L = 2)

Quad membership

Determining the lowest-level quad memberships is the most time consuming part in the procedure. At successively higher levels, the task becomes easier as we need only assemble/concatenate the list of the number of demands from the one at the lower level. This procedure continues until the number of demand points for each quad at level 1 has been determined. It is worth noting that the quadtree structure is independent of the facility locations and hence the computational burden of this procedure is carried out only once for a given set of demand points and hence can be seen as requiring the only fixed cost.

3.2 The strength of the allocation procedure

The strength of this procedure relies on the following theorem and the lemma that follows.

Theorem

Let $S \subset \Re^n$ be a convex polytope with vertices $\bar{v}_i, i = 1,...,r$, and let $\bar{q}$ be an arbitrary point in S. For $\bar{x}, \bar{y} \in \Re^n$, let $d(\bar{x}, \bar{y})$ denote the Euclidean distance between $\bar{x}$ and $\bar{y}$. If $d(\bar{v}_i, \bar{x}) \le d(\bar{v}_i, \bar{y})$ for all *i*, *i* = 1,…,*r*, then $d(\bar{q}, \bar{x}) \le d(\bar{q}, \bar{y})$.

Proof

In general $\forall \bar{x}, \bar{y}, \bar{u}, \bar{v} \in \Re^n$, we have:

- $d(\bar{x}, \bar{y}) \le d(\bar{u}, \bar{v}) \Rightarrow d^2(\bar{x}, \bar{y}) \le d^2(\bar{u}, \bar{v})$.
- $d^2(\bar{x}, \bar{y}) = \|\bar{x} - \bar{y}\|^2 = \|\bar{x}\|^2 + \|\bar{y}\|^2 - 2\bar{x}\bar{y}$ where $\bar{x}\,\bar{y}$ is the scalar product of $\bar{x}$ and $\bar{y}$.
- If $\bar{q} \in S$ then $\bar{q}$ can be written as a convex combination of the vertices, that is, there exist $\lambda_i \ge 0$, $i = 1,...,r$, verifying $\sum_{i=1}^{r} \lambda_i = 1$, such that $\bar{q} = \sum_{i=1}^{r} \lambda_i \bar{v}_i$.
- As $d(\bar{v}_i, \bar{x}) \le d(\bar{v}_i, \bar{y}) \Rightarrow \|\bar{v}_i - \bar{x}\|^2 \le \|\bar{v}_i - \bar{y}\|^2 \Rightarrow$

  $\|\bar{x}\|^2 - 2\bar{v}_i\bar{x} \le \|\bar{y}\|^2 - 2\bar{v}_i\bar{y}$. (3)

  Multiplying both sides in (3) by $\lambda_i, i = 1,...,r$, and summing, we obtain:

  $\|\bar{x}\|^2 - 2(\sum_{i=1}^{r} \lambda_i \bar{v}_i)\bar{x} \le \|\bar{y}\|^2 - 2(\sum_{i=1}^{r} \lambda_i \bar{v}_i)\bar{y}$ (4)

  From (4), as $\bar{q} = \sum_{i=1}^{r} \lambda_i \bar{v}_i$, it follows that:

  $\|\bar{x}\|^2 - 2\bar{q}\bar{x} \le \|\bar{y}\|^2 - 2\bar{q}\bar{y}$

  Adding $\|\bar{q}\|^2$ to both sides

  $\|\bar{q}\|^2 + \|\bar{x}\|^2 - 2\bar{q}\bar{x} \le \|\bar{q}\|^2 + \|\bar{y}\|^2 - 2\bar{q}\bar{y} \Rightarrow \|\bar{q} - \bar{x}\|^2 \le \|\bar{q} - \bar{y}\|^2 \Rightarrow$

  $\|\bar{q} - \bar{x}\| \le \|\bar{q} - \bar{y}\| \Rightarrow d(\bar{q}, \bar{x}) \le d(\bar{q}, \bar{y})$. □

Based on the above theorem, a lemma, which relates to the allocation of a quad and the demand points contained in it, as constructed in this study, is proposed.

Lemma

If all 4 corners of a quad are closest to one facility then the quad with all of its customers are also allocated to that facility.

The proof is based on the theorem where a quad is a convex polytope with $r = 4$ with its 4 corners corresponding to the vertices $\bar{v}_i; i = 1,...,4$. □

Besides allocating all the demand points of the quad to the facility nearest to the four corners as claimed by our Lemma, there is no need to explore this quad any further. We term such a quad as fathomed and all of its demand points allocated to that one facility. The more populated is the quad the larger the saving in computational time is.

Further Fathoming

We also introduced two other rules that cause a quad to be fathomed:

(i) At any level, if a quad happens to contain no demand points (empty).

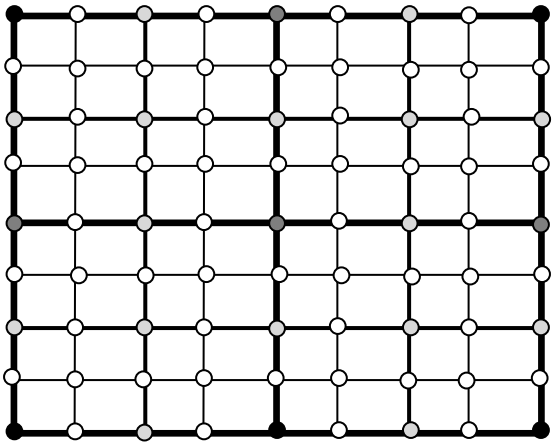
(ii) A quad that contains no more than five demand points is treated differently. Its customers are allocated in the classical manner as it is quicker to do it this way than to continue partitioning this quad.

The Allocation Procedure

This process uses a top-bottom procedure. All quads at each level are successively considered, and then descend to the next level where the higher level quads have not been fathomed. This process continues until all quads have been fathomed, or the bottom of the tree is reached. Some of the quads at level L (lowest level) may remain unallocated and those demand points inside these quads are then allocated to the nearest facility in the classical way.

As discussed above, this method allocates quad corners to facilities. Each quad has four corners that must be allocated. Figure 8 shows that we start at the top level (level 0) and determine the allocations of all four corners of the largest quad. At level 1, we need to evaluate the allocation of five corners only instead of all the nine corners as four were already allocated at the previous level. At level two, only 16 of the 25 corners need to be allocated, whereas at level three 56 of the 81 corners need to be defined. It is worth noting that each reduction in evaluating a corner represents a reduction of *p* distances to be calculated and compared against. This reduction contributes to the overall saving which can be substantial.

Allocating quads to facilities is an aggregation procedure. Because of the convexity of the way the quads are constructed and the use of Euclidean distance, the demand points in each allocated quad are allocated to the correct facility as demonstrated by our lemma. By individually allocating the demand points inside the quads that are not assigned yet, these are also allocated to the correct facility using the classical way.



Symbol	Level	Total Corners	New Corners
●	0	4	4
●	1	9	5
●	2	25	16
○	3	81	56

Figure 8. The saving in the allocation of corners to facilities

Though the allocation of demand points to the facilities is completed, in many applications the distance from these demand points to the nearest facility is required to record the total distance. This is where source A and B errors are introduced in the conventional aggregation procedures, where the demand points are aggregated and assumed to have the same location as the centroid of their aggregated group. In the classical allocation procedure, the distances from each demand point to all facilities are calculated and the minimum distance determined. In the quadtree procedure, the distance from each demand point to its known closest facility is calculated once as there is no need to compute distances to other facilities. In other words, the demand points are aggregated for the allocation purpose, but they are disaggregated when it comes to distance calculations. For example, if there are 60 facilities, each demand point is assigned to one facility only; the 59 distances and weighted distance calculations are obviated for each of the demand points in that fathomed quad. If there are, say, 10 demand points in a quad, a saving of 590 distance calculations are saved besides the time required to determine the smallest distance between a demand point to all the facilities.

3.3 A time evaluation of the two allocation procedures

We define the 'conventional allocation method' as the brute force approach, where the allocation of the demand points, say n, is carried out for each demand point (i=1,…,n) by calculating the smallest distance to each of the *p* facilities. For instance, if T_0 refers to the cost of computing one distance value between a demand point and a facility and T_1 T_1 is the cost for computing the minimum distance from the i^{th} demand point to all the *p* facilities (i.e., $Min\{d(i,j)\}, j = 1,..., p$), the computational cost will be $pnT_0 + nT_1 = n(pT_0 + T_1)$. If a multiple allocation is required (say *k* iterations) to different sets of *p* facilities, then the overall computational cost, say TC_0, becomes $kn(pT_0 + T_1)$.

Using the quadtree allocation, there are two primary sources of cost: (i) the cost to construct the quadtree database, say *F*, and (ii) the cost to allocate the quadtrees to the facilities. In (i), *F* consists of the cost of determining the quadtree structure and the cost of populating the quadtree database with the demand points. This fixed cost depends on the number of hierarchical levels (L) only but it is independent of the number of facilities (*p*) and the number of demand points (*n*). The time to populate the quadtree database is proportional to *n* and *L*, but it is independent of *p*.

In other words, the conventional allocation does not incur this fixed cost. The efficiency of the quadtree method will improve as the number of iterations increases. Let TC_1 be the cost of this approach, $TC_1 = F + kT_2$ with T_2 representing the cost of assigning the corners to the facilities. The cost of allocating the quads to facilities depends on the degree to which the quadtree structure can be exploited. Its cost will be relatively low if many higher level quads are allocated but the fixed cost will be even higher. The cost of the quadtree method is related to the number of facilities. As this number increases, the likelihood of a higher order quads being split also increases as fathoming would not occur earlier which leads to an increase in computational effort. The number of iterations required ($\hat{k}$) for the quadtree method to outperform the conventional method is derived as follows:

$TC_1 = F + kT_2 < kn(pT_0 + T_1) = TC_0$. This leads to:

$$k > \frac{F}{n(pT_0 + T_1) - T_2} = \hat{k} \tag{5}$$

where $T_2 = f(p, N_L)$.

4. Computational experiments

This section comprises two subsections. The first subsection describes three allocation methods followed by the computational results on the allocation problems. We refer to three allocation techniques instead of two as there are two variants of the brute force which we also call the conventional method.

4.1 A brief on the three allocation methods

Three allocation methods are proposed in the case of the Euclidian distance. These include the conventional method, the modified-conventional method, and the quadtree method described below. Here, there are *n* demand points and *p* facilities where $(X_i, Y_i), i = 1,..., n$ and $(\alpha_j, \beta_j), j = 1,..., p$ are the coordinates of the demand points and facilities respectively. We would like to allocate each demand point to its closest facility.

A) The Conventional Method (CM)

The conventional method allocates demand points as follows: for each demand point *i* (*i* = 1,…,*n*), its nearest facility is computed as $Arg \underset{j=1,...,p}{Min} d(i, j)$, where

$$d^2(i, j) = (X_i - \alpha_j)^2 + (Y_i - \beta_j)^2, i = 1,..., n; j = 1,..., p \tag{6}$$

B) The Modified-Conventional Method (MM)

This method improves the way to calculate the distance between facility and demand point where the allocation procedure is performed a large number of times. The method is based on Al-Zoubi and Al-Rawi (2008) where they developed a simple but efficient approach for computing the silhouette coefficients when evaluating clusters.

The Euclidian distance, $d(i, j)$ as defined in (6), can also be written as $d^2(i, j) = X_i^2 + Y_i^2 + \alpha_j^2 + \beta_j^2 - 2X_i\alpha_j - 2Y_i\beta_j$. The terms $\gamma_i = X_i^2 + Y_i^2$ and $\varphi_j = \alpha_j^2 + \beta_j^2$

need to be calculated only once as γ_i is a function of i only (the same for φ_j). Using this observation, there is obviously a fixed cost to calculate and store the γ_i and the φ_j though relatively negligible with respect to the large number of iterations used. This scheme can potentially save a large amount of computational time.

In brief, the idea is to calculate and store γ_i $(i = 1,...,n)$ and φ_j $(j = 1,..., p)$, and for each demand point i $(i = 1,...,n)$, its closest facility is computed as $Arg(\underset{j=1,...,p}{Min}\, d(i, j))$, using (7) instead of (6)

$$d^2(i, j) = \gamma_i + \varphi_j - 2X_i\alpha_j - 2Y_i\beta_j \tag{7}$$

C) The Quadtree Method (QM)

The Quadtree method is also designed for solving the allocation problem where the allocation must be performed a large number of times. Here, MM is also used to calculate the distance between facility and demand point. The followings steps of this method are given below.

1. Calculate and store $\gamma_i = X_i^2 + Y_i^2$, $i = 1,...,n$ and $\varphi_j = \alpha_j^2 + \beta_j^2$, $j = 1,..., p$.
2. Find the minimum and maximum of X and Y coordinate ((X_a, Y_a) and (X_b, Y_b)).
3. Build the quadtree database.

   3a- Determine the corners $(\tilde{x}_k, \tilde{y}_k)$, $k = 0,...,(N_L - 1)$ with the first corner being located at (X_a, Y_a).

   3b- Establish the quadtree structure. We develop the quads from the highest level (level 1) until the lowest level (level L), see section 3.1 for more details.

   3c- Assign all demand points to quads on the lowest level. At successively higher levels, Assemble the demand point's lists from the quads at the lower level, see section 3.1 for more details.
4. Starting from Level 1 until level L, we allocate each quad, which has not been allocated, to one facility if all corners of this quad are allocated to this facility with all demand points inside this quad being systematically assigned to the same facility. See section 3.2 for more details.

5. Allocate all the remaining unallocated demand points to the facilities individually using the modified conventional allocation scheme (MM) given in subsection 4.1(B).

4.2 Computational Results on the Allocation Problems

Experiments are performed to investigate the relationship among the number of demand points, the number of facilities, the depth of the quadtree, and the number of iterations. Seven test problems with n = 5000, 10000, 20000, 30000, 40000, 70000, and 264000 are conducted. These seven data are generated randomly in a square unit of 1000x1000, 2000x2000, 5000x5000, 5500x5500, 6000x6000, 7500x7500, and 10000x10000 respectively. For example, in the data instance with n=5000, the coordinate (X_i, Y_i) are generated randomly in the range $(1,1000)^2$.

In these experiments, the maximum number of levels (L) is set to 8. For each value of p, the number of runs/iterations (k) of the allocation process is set to 1000 and the facility locations are randomly generated. Computing times for the fixed cost and the variable cost for all three methods are also recorded. Based on these costs, we estimate the number of iterations required for QM to be more efficient than both the CM and MM. The code was written in C++ .Net 2010 and was executed on a PC with an Intel Core i5 CPU 650@ 3.20GHz processor, 4.00 GB of RAM and under Windows 7 (32bit).

Some Observations

Table 2 presents the CPU times of the three allocation methods for n = 30,000 where the CPU times are in milliseconds. The notation in this table is as follow: FC(RD) : Fixed Costs (read demand points file); FC(BQ) : Fixed Costs (build the quadtree database); L01 - L08 : the level of Quadtree (level 1 - 8); VC: Variable Costs (average of allocating time from 1000 iterations); CPU : CPU times required to perform 1000 iterations. In this table, numbers in bold refer to the lowest CPU values. The fixed cost of QM (reading the demand point file and setting up the quadtree database) is proportional to the depth of the hierarchy. The fixed cost of QM is, as one may expect, relatively much higher than that of CM and MM.

Table 2. CPU Times (in millisecond) for the location-allocation using CM, MM, and QM (case of n = 30,000)

	CM		**MM**		**QM**															
					L01		**L02**		**L03**		**L04**		**L05**		**L06**		**L07**		**L08**	
FC(RD)	150		137		137		137		137		137		137		137		137		137	
FC(BQ)	-		-		23		59		195		719		2,801		11,000		43,738		174,461	
p	VC	CPU	VC	CPU	VC	CPU	VC	CPU	VC	CPU	VC	CPU	VC	CPU	VC	CPU	VC	CPU	VC	CPU
5	36.66	36,813	24.04	24,176	30.13	30,289	24.44	24,640	17.38	17,707	13.57	**14,430**	11.88	14,817	12.30	23,437	15.82	59,695	29.69	204,286
10	71.89	72,037	46.28	46,414	53.24	53,399	48.68	48,877	34.68	35,008	23.76	24,613	17.93	**20,867**	17.17	28,309	21.39	65,262	37.98	212,579
15	105.45	105,604	68.24	68,375	74.27	74,432	72.33	72,528	54.85	55,180	36.70	37,551	25.76	**28,696**	23.48	34,617	28.50	72,379	47.62	222,215
20	139.65	139,804	91.26	91,400	95.40	95,564	95.41	95,604	76.64	76,968	51.77	52,629	34.96	**37,899**	31.05	42,188	36.89	80,769	58.73	233,325
25	174.50	174,653	111.84	111,975	116.86	117,020	117.77	117,963	99.21	99,539	68.55	69,407	45.46	**48,399**	39.39	50,524	46.29	90,168	70.82	245,422
30	208.59	208,740	134.46	134,596	138.21	138,371	139.75	139,950	121.71	122,039	86.35	87,207	56.82	59,759	48.48	**59,617**	56.63	100,504	83.80	258,402
35	242.31	242,460	156.56	156,694	159.36	159,519	161.46	161,658	144.73	145,059	105.08	105,938	69.00	71,938	58.36	**69,492**	67.83	111,703	97.81	272,410
40	277.76	277,905	177.45	177,584	180.93	181,094	182.99	183,181	167.17	167,504	124.71	125,567	81.97	84,911	68.98	**80,113**	79.54	123,414	112.26	286,855
45	312.22	312,365	199.49	199,623	202.37	202,529	200.19	200,389	189.73	190,066	145.02	145,876	95.70	98,633	80.09	**91,231**	91.86	135,739	127.39	301,992
50	345.19	345,341	223.16	223,297	223.33	223,494	220.54	220,740	211.98	212,308	165.67	166,528	109.95	112,892	91.70	**102,836**	104.75	148,629	142.93	317,524
55	380.14	380,291	245.93	246,071	244.28	244,436	239.88	240,079	234.00	234,336	187.04	187,895	124.94	127,876	103.95	**115,083**	118.19	162,067	158.97	333,571
60	418.91	419,059	266.55	266,683	266.46	266,622	261.48	261,674	255.79	256,117	208.73	209,586	140.36	143,302	116.71	**127,848**	132.18	176,058	175.74	350,340
65	457.77	457,921	286.10	286,241	290.30	290,458	281.41	281,606	277.77	278,098	230.58	231,434	156.24	159,176	129.86	**140,996**	146.64	190,512	192.73	367,329
70	494.85	495,002	309.41	309,549	308.81	308,966	302.50	302,700	299.26	299,590	252.54	253,391	172.60	175,533	143.56	**154,696**	161.57	205,441	210.32	384,919
75	525.87	526,021	336.24	336,372	330.71	330,872	323.32	323,520	320.80	321,133	274.94	275,798	189.41	192,348	157.78	**168,918**	176.76	220,636	228.06	402,661
80	557.43	557,582	355.94	356,075	351.79	351,953	344.04	344,235	342.39	342,718	297.20	298,051	206.56	209,493	171.96	**183,094**	192.60	236,476	246.14	420,742
Avg	297	296,975	190	189,695	192	191,814	189	188,709	178	178,336	142	142,869	96	99,159	**81**	**92,062**	92	136,216	126	300,911
dev(%)	267	223	134	106	137	108	133	105	120	94	75	55	19	8	0	0	14	48	56	227

Table 2 also presents dev(%) which is the percent gap from the best known CPU Time and is computed as $dev(\%) = 100\frac{CPU_c - CPU_b}{CPU_b}$, where CPU_c and CPU_b correspond to the CPU time obtained with method 'c' and the best CPU time respectively.

Table 3 summarises the average of CPU times for n = 5000, 10000, 20000, 40000, 70000, and 264000 over p =5 to 80 with a step of 5. According to the results, MM outperforms CM as the average allocation time (VC) of MM is shorter than the one of CM in all cases. The average allocation time of QM is shorter than both CM and MM in most cases, especially for larger values of p and/or n. It means that the advantages of QM for allocating demand points are clear. We observe that the trend of the CPU values for the different levels (except for n = 40,000) is decreasing until a certain level and then the values begin to increase (the minimum occurs for L04, L05 or L06, depending on the cases). We also observe that the CPU values for QM-L08 are worse than the values obtained from MM in all cases.

Table 3. The overall summary of the average CPU Times (in millisecond) for the Location-Allocation problem for n = 5,000 to 264,000 over p = 5 to 80

	n = 5,000				n = 10,000				n = 20,000			
	VC	dev(%)	CPU	dev(%)	VC	dev(%)	CPU	dev(%)	VC	dev(%)	CPU	dev(%)
CM	51	102	50,936	100	93	131	93,003	125	183	183	183,330	168
MM	39	53	38,600	52	60	50	60,344	46	122	88	121,671	78
L01	31	23	31,015	22	65	61	64,976	57	124	91	123,915	81
L02	31	22	30,910	22	62	55	62,556	51	123	90	123,422	81
L03	30	18	29,766	17	60	49	60,087	45	118	82	118,142	73
L04	25	**0**	25,434	**0**	52	30	52,521	27	95	46	95,320	39
L05	26	1	26,043	2	40	**0**	41,345	**0**	66	3	68,334	**0**
L06	27	8	29,214	15	43	6	46,583	13	65	**0**	71,941	5
L07	36	42	43,144	70	51	28	66,872	62	76	17	103,940	52
L08	65	158	94,239	271	82	103	140,790	241	109	69	224,262	228
	n = 40,000				n = 70,000				n = 264,000			
	VC	dev(%)	CPU	dev(%)	VC	dev(%)	CPU	dev(%)	VC	dev(%)	CPU	dev(%)
CM	374	271	374,219	223	689	317	689,270	262	2,524	622	2,524,922	336
MM	254	152	254,349	120	422	156	422,250	121	1,603	359	1,604,516	177
L01	247	145	247,254	114	431	161	430,889	126	1,669	378	1,670,282	189
L02	257	155	257,748	123	431	161	431,601	126	1,654	373	1,656,071	186
L03	250	148	250,293	116	411	149	411,990	116	1,585	354	1,587,829	174
L04	198	97	199,384	72	325	97	326,881	71	1,260	261	1,267,307	119
L05	134	33	138,008	19	213	29	220,003	15	792	127	819,002	42
L06	101	**0**	115,792	**0**	165	**0**	190,654	**0**	483	38	578,453	**0**
L07	110	10	168,723	46	182	10	290,580	52	349	**0**	724,095	25
L08	142	40	366,322	216	230	40	658,320	245	385	10	1,875,300	224

Table 4 presents the mean percentage of the number of quads unallocated in the quadtree method for n = 40,000. The demand points, inside those quads, are to be allocated, in the conventional way. The two fathoming procedures trade off against each other: when L is small, more demand points are unallocated because many unsplit quads remain, whereas when L is large (near 8), most of the quads might consist of a small number of demand points (≤ 5). Table 4 shows that when n = 40000, p = 5, and L = 3, a mean of 39.81% of quads have not been allocated. However when p = 80, with many more quads (smaller ones), a mean of 99.10% of quads are not allocated. The demand points of those unallocated quads must be allocated normally which counter balance the benefit of using this approach except if a large value of L is used. The number of unallocated quads decreases with the increase in L but increases with p.

Table 4. The average percentage of quads unallocated (QM) for n = 40,000 with p = 5 to 80

p	**L01**	**L02**	**L03**	**L04**	**L05**	**L06**	**L07**	**L08**
5	97.63	69.29	39.81	20.99	10.75	4.95	0.11	0.00
10	99.98	89.01	59.80	33.46	17.56	8.19	0.18	0.00
15	100.00	95.39	71.56	42.27	22.60	10.64	0.24	0.00
20	100.00	98.05	79.41	49.11	26.71	12.66	0.29	0.00
25	100.00	99.10	84.72	54.69	30.22	14.43	0.33	0.00
30	100.00	99.59	88.59	59.45	33.36	16.02	0.36	0.00
35	100.00	99.82	91.28	63.50	36.12	17.44	0.40	0.00
40	100.00	99.88	93.50	67.07	38.72	18.80	0.43	0.00
45	100.00	99.98	95.08	70.18	41.05	20.04	0.46	0.00
50	100.00	99.98	96.04	72.91	43.21	21.17	0.49	0.00
55	100.00	99.96	96.96	75.34	45.22	22.26	0.51	0.00
60	100.00	100.00	97.65	77.60	47.14	23.32	0.54	0.00
65	100.00	99.99	98.23	79.56	48.95	24.33	0.56	0.00
70	100.00	100.00	98.56	81.31	50.61	25.24	0.59	0.00
75	100.00	100.00	98.87	82.90	52.18	26.15	0.61	0.00
80	100.00	100.00	99.10	84.38	53.72	27.05	0.63	0.00
Avg	99.85	96.88	86.82	63.42	37.38	18.29	0.42	0.00

Table 5 shows the summary of this average percentage from n = 5,000 to 264,000 for p = 5 to 80. For most situations, it has been observed that demand points can be allocated to facilities more efficiently with QM. However, this efficiency comes at a considerable fixed cost. If we wish to make only a few iterations, MM will be much more efficient. However, if a large number of iterations is required, QM that uses the quadtree level with the lowest variable time (allocating time), as documented in the tables, will be more appropriate. The

minimum number of iterations required for QM to outperform CM and MM can be calculated as follows:

$$\rho_T = \frac{\tau_Q - \tau_T}{\omega_T - \omega_Q} \text{ and } \rho_M = \frac{\tau_Q - \tau_M}{\omega_M - \omega_Q} \quad (8)$$

Where: ρ_T, ρ_M : The number of iterations required for QM to outperform CM and MM respectively

τ_Q, τ_T, τ_M : Fixed cost of QM, CM, MM respectively

$\omega_Q, \omega_T, \omega_M$: Variable cost of QM, CM, MM respectively

Table 5. The overall summary of the average percentage of quads unallocated from n = 5,000 to 264,000 over p = 5 to 80

n	**L01**	**L02**	**L03**	**L04**	**L05**	**L06**	**L07**	**L08**
5000	99.84	96.88	86.91	63.57	14.17	0.03	0.00	0.00
10000	99.86	96.93	86.98	63.58	34.96	0.74	0.00	0.00
20000	99.80	96.87	86.79	63.46	37.41	7.38	0.02	0.00
30000	99.84	96.89	86.92	63.57	37.46	14.96	0.14	0.00
40000	99.85	96.88	86.82	63.42	37.38	18.29	0.42	0.00
70000	99.85	96.83	86.86	63.43	37.36	19.95	2.74	0.01
264000	99.83	96.82	86.90	63.39	37.31	19.96	10.25	1.21
Avg	99.84	96.87	86.88	63.49	33.72	11.62	1.94	0.17

Figure 9 presents the chart of the number of iterations required for QM to outperform CM and MM when n = 5,000 to 264,000 with p = 40. Here, we set L = 5 (L05) based on the results shown in Table 3. The figure shows that the use of QM on the allocation problem is more efficient than the one of CM and MM when a certain number of iterations is required. The values of ρ_T are 21, 19, 17, 15, 20, 33, and 132 for n = 5000, 10000, 20000, 30000, 40000, 70000, and 264000 respectively. While the values of ρ_M are 39, 48, 32, 30, 40, 66, and 270 for n = 5000, 10000, 20000, 30000, 40000, 70000, and 264000 respectively. The highest values occur for the greatest value of n but none of the sequences is found to be increasing. Also note that MM always outperforms CM as both use the same technique but the former has a more efficient data structure which avoids recomputing unnecessary calculations of γ_i and φ_j.

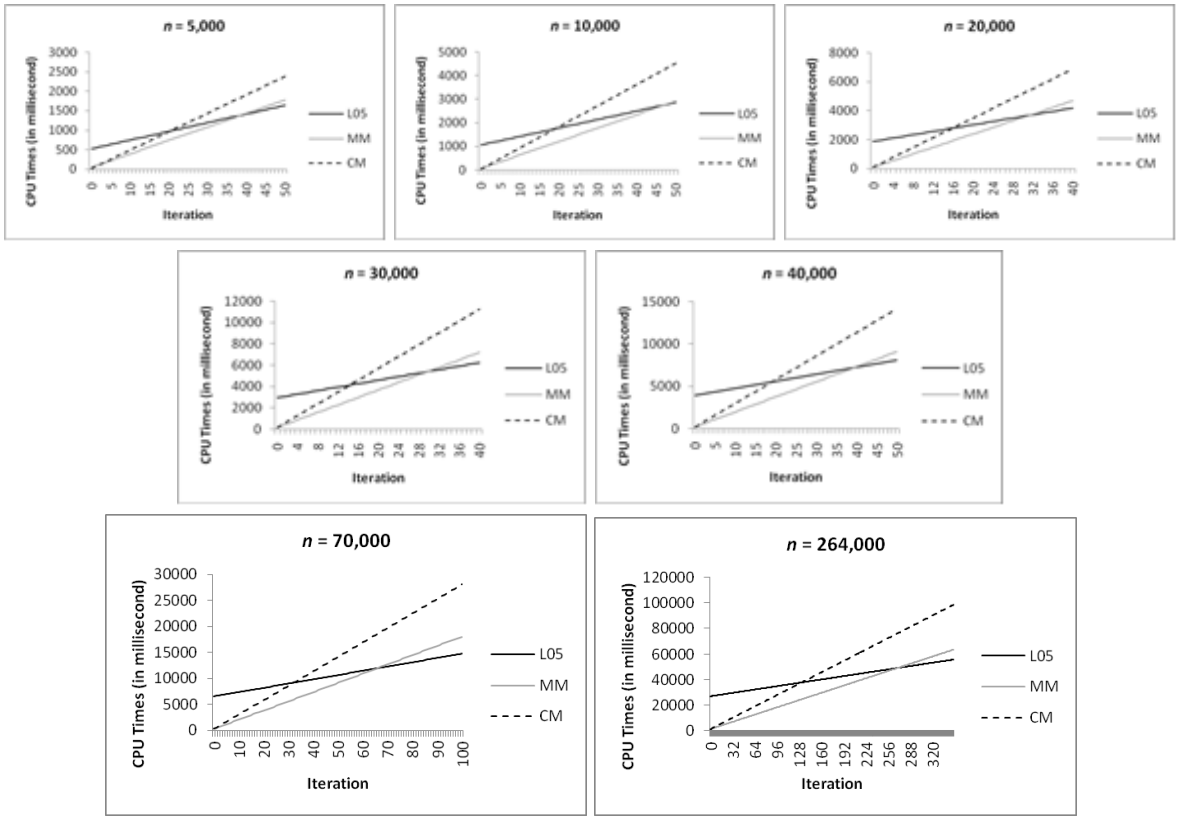


Figure 9. The number of iterations required for QM to outperform CM and MM (case of p = 40)

5. Integration of QM in solving facility location problems

The effect of using QM as an allocation procedure for solving the discrete p-median and p-centre problems is explored. The first subsection describes the integration of QM into the methods used for solving this class of location problems followed by a subsection on the computational results. In this study we consider all customer sites as potential facility sites which we denote by H ($|H| = n$) .

5.1 Two simple facility location methods

We incorporate QM into two methods used for solving the p-median and the p-centre problems namely the multi-start and the Reduced Variable Neighbourhood Search (RVNS) methods. We then compare the performance of these two methods with and without QM.

A) The Multi-start Method

The multi-start method for both the *p*-median and the *p*-centre problems is a simple method where *p* facility locations are chosen randomly and then all demand points are allocated to their nearest facility. This process is repeated *s* times and the facility configuration that yields the smallest objective function value is chosen. In our study, *s* is set to 1,000.

B) The Reduced Variable Neighbourhood Search (RVNS)

VNS combines both local search and neighbourhood search. The first search looks for local optimality while the latter aims to escape from these local optima by systematically using another (usually a larger) neighbourhood if the improvement is not found and then reverts back to the first neighbourhood (usually the smallest one) otherwise. More details about VNS and its various variants and successful application can be found in Hansen *et al*. (2010). RVNS, which is one of the variants of VNS but without a local search procedure, is used in this study. The implementation of RVNS is adopted mainly to obtain good initial solutions. Let *X* denote the facility configuration of the current solution and *f*(X) its corresponding objective function value.

For the *p*-median problem, the k^{th} neighbourhood structure N_k is defined as follows:

$$N_k(X) = \text{use of } N_1(X) \ k \text{ times with } k = 1,...,k_{\max} \text{ and } N_1(X) = X - \tau' \cup \tau'' \qquad (9)$$

where τ'' is chosen randomly in $H - X$ and $\tau' \in X$ is also selected randomly.

Note that $N_1(X)$ can also be rewritten as $N_1(X) = \{X' \in H_p : \rho(X, X') = 1\}$ where $H_p = \{Z \subset H : |Z| = p\}$ and $\rho(Z, Z') = |Z \setminus Z'|, \forall Z, Z' \in H_p$.

For the *p*-centre problem, the neighbourhood structure used in the shaking process includes the construction of the promising area where a chosen facility location may move to. Let i_m refer to the customer whose distance to its nearest facility is largest (r_m) while *U* denotes the list of the potential sites which are located in the promising area as shown in Figure 10.

This is defined as follows: Let $\tilde{C}(F, R)$ be the set of potential facility sites encompassed by the circle centered at point F with a radius R.

Then $U = \tilde{C}(F^*, r_m) \bigcap (\tilde{C}(i_m, r_m) \setminus \tilde{C}(i_m, \frac{r_m}{2}))$ with F^* being the facility site serving i_m.

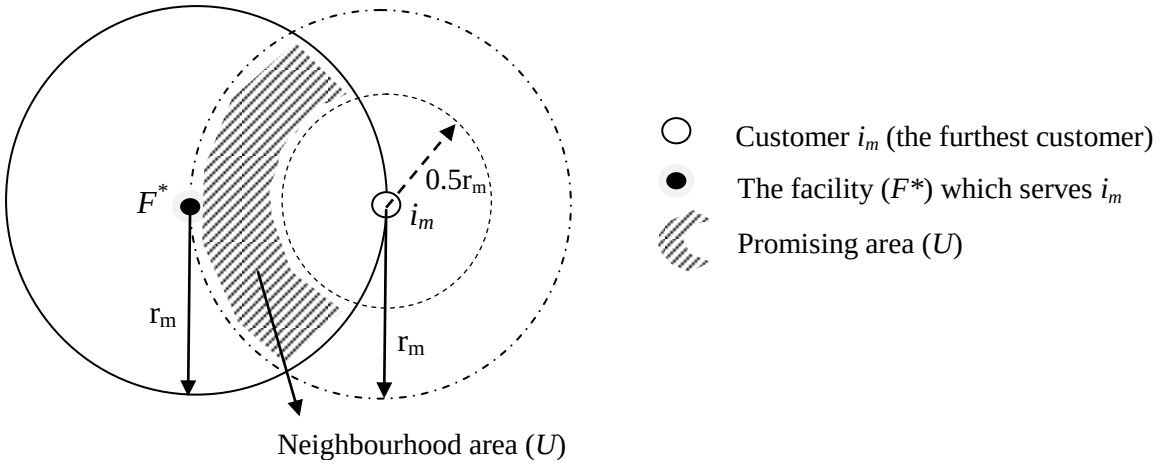


Figure 10. The restricted but guided neighbourhood for the *p*-centre problem

The k^{th} neighbourhood structure NN_k for the *p*-centre is defined as follows:

$$NN_k(X) = \text{use of } NN_1(X) \text{ } k \text{ times with } k = 1,...,k_{\max}$$

$$\text{and } NN_1(X) = X - \sigma' \cup \sigma'' \tag{10}$$

where σ'' is chosen randomly in $U - X$ and $\sigma' \in X$ is a facility with the largest radius (the one which serves customer i_m).

The main steps of the RVNS for both the discrete *p*-median and *p*-centre problems are summarised in Figure 11. For simplicity for the initial solution, we choose the location of the *p* facilities randomly from the set H. Note that the implementation of the shaking procedure in RVNS is slightly different between the two location problems as in the p-centre the new solution needs to be identified at each iteration given the insertion of the added facility belongs to the subset of sites currently assigned to the facility with the largest circle. However this restriction is not necessary in the p-median where both the removed facility and the added one are both randomly selected.

Repeat the following steps until the stopping criterion is reached:

Step 1 Set $k = 1$, let X be an initial solution and $f(X)$ its corresponding objective function value.

Step 2 Shaking

For the *p*-median problem:

Determine $X' \in N_k(X)$ and calculate $f(X')$ using (9)

For the *p*-centre problem:

(a). Let $X'' = X$

(b). Do the following step k times ($l = 1,\ldots,k$)

- determine $X' \in NN_l(X'')$ using (10)
- Calculate $f(X')$, identify the subset of customers encompassed by the largest circle and set $X'' = X'$

Step 3 Move or Not

If $f(X') < f(X)$ set $X = X'$ and $k = 1$ else set $k = k+1$.

Figure 11. The RVNS for the discrete *p*-median and *p*-centre problems

5.2 Computational results

The performance of QM on these two discrete location problems is tested on the TSP dataset which comprises of four instances namely Italy Data (n = 16,862), Sweden Data (n = 24,978), Burma Data (n = 33,708), and China Data (n = 71,009). We vary the value of p from 5 to 30 with an increment of 5. In QM, we set L = 5 as this level showed to yield better performance when compared to the other levels for $p < 30$ (see the previous section).

A. The Multi-start Method

Figure 12 shows the CPU time deviation (dev(%)) of CM and MM with respect to QM for the case of the multi-start method. The figure also presents the best CPU time for QM after 1000 iterations for both the *p*-median and *p*-centre problems. Here, the methods use the same value for the seed when generating random values so the set of p facility locations chosen in the 1000 iterations remain the same for the different allocation procedures.

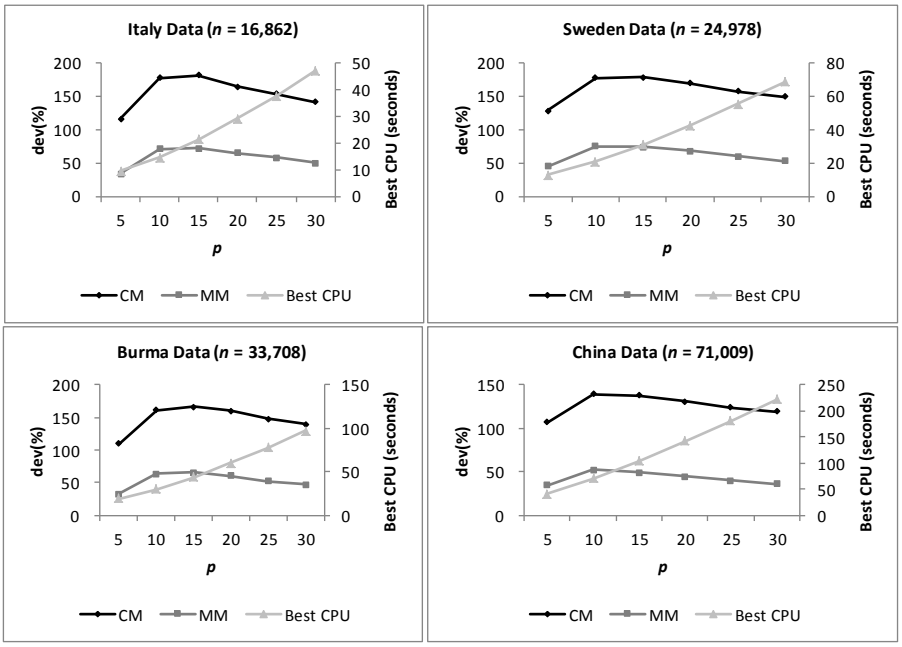


Figure 12. The CPU time Deviation (dev(%)) of CM and MM with respect to QM case of the Multi-start method

Figure 12 indicates that the use of QM in the allocation process outperforms both CM and MM. QM is approximately 150% and 50% faster than CM and MM respectively. In brief, the use of QM is found to be very effective in reducing the computing time. This demonstrates that QM can be incorporated in other powerful algorithms for large-scale location problems such as those given by Hansen *et al*. (2009), Avella *et al*. (2012) and Irawan *et al.* (2014). For illustration purposes we also introduced QM into a Reduced VNS for the case of *p*-median and *p*-centre problems. This is given next.

B. The Reduced Variable Neighbourhood Search (RVNS)

In this study, we set the parameter k_{max} = 2. Here, we use the CPU time based on the best CPU time obtained by the multi-start method as the stopping criterion. Figures 13a and 13b present the comparison of the average Deviation (%) over all values of *p* between CM, MM, and QM when used in the RVNS method for the *p*-median and *p*-centre problems respectively. Deviation (%) is the percent gap from the best solution found by these variants and is computed as $Deviation\ (\%) = 100\dfrac{Z_c - Z_b}{Z_b}$, where Z_c and Z_b correspond to the *Z* value obtained with method 'c' and the best *Z* respectively. The experiments are conducted on the instances tested by the multi-start method. The detailed results are given in the Appendix under Tables A1 and A2 for the p-median and the p-centre problems respectively.

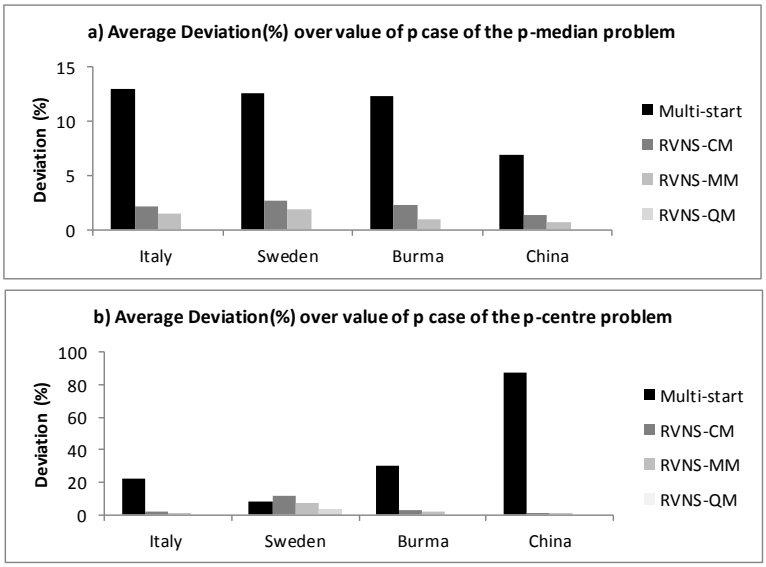


Figure 13. The Deviation (%) between RVNS-CM, RVNS-MM, RVNS-QM, and Multi-start

The case of the p-median problem

Figure 13a reveals that in general the values of the average Deviation (%) obtained by RVNS for the *p*-median problem are much smaller than the ones found by the multi-start method. Figure 13a also shows that incorporating QM in the RVNS method yields the smallest average Deviation (0.00%) meaning that in all instances, the use of QM in RVNS outperforms the use of CM and MM in RVNS as well as QM in the multi-start method. This is because the use of QM in RVNS saves the allocation time so the number of iterations increases. Similar to the previous case, the use of QM in the RVNS method for the *p*-median problem is very effective in reducing the allocation time.

The case of the p-centre problem

Figure 13b shows the comparison in the average Deviation (%) value between the use of CM, MM, and QM in the RVNS method for the *p*-centre problem. The RVNS method, similarly to the case of the *p*-median problem, generally provides better solutions than the multi-start method. The figure also shows that the use of QM in the RVNS method produces the smallest deviation with a value of 0% in all instances except the Sweden Data.

6. Conclusions and suggestions

This paper introduces a special data compression approach based on a quadtree technique for the allocation of a large number of demand points to their nearest facilities. The main

result is that if a quad has all of its four corners allocated to one facility, all the customers in that quad are systematically allocated to that facility. This result is supported by an interesting theorem that is valid for any convex polytope. The experiments show that in most situations, the quadtree allocation procedure outperforms the conventional and the modified-conventional allocation, and overcomes its overhead costs beyond a critical number of iterations. The quadtree method has its greatest value where many different allocations of the same set of demand points to different set of facilities are required. The saving gets more significant when a large number of facilities and iterations are needed as the fixed costs incur only once for any set of demand points.

In our study, we also incorporate the quadtree technique into two methods used for solving large discrete *p*-median and *p*-centre problems. These include the multi-start and the Reduce Variable Neighbourhood Search (RVNS). The computational results show that the use of QM in these methods is very effective when solving very large instances.

The use of the quadtree method can also be incorporated into existing and powerful algorithms used for large-scale location problems. It would be also interesting to explore adopting QM to other combinatorial problems where the allocation task is an important part of the search which is required a large number of times.

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. We are also grateful to the Indonesian Government and Jurusan Teknik Industri-ITENAS Bandung, Indonesia for the sponsorship of the second author. This research has also been partially supported by the Ministry of Economy and Competitiveness of Spain under the research project ECO2011-24927.

Appendix

The detailed results of Z value between multi-start and RVNS with CM, MM, and QM for the p-median and the p-centre problems are presented in Tables A1 and A2 respectively.

Table A1. The comparison in solution quality between multi-start and RVNS with CM, MM, and QM: The case of the *p*-median problem

Description	*N*	*p*	Best *Z*	Deviation (%)			
				Multi-start	RVNS		
					CM	MM	QM
Italy Data	16,862	5	19,738,367.25	2.69	1.29	0.83	0.00
		10	13,161,728.73	12.18	2.20	1.76	0.00
		15	10,713,832.06	11.45	2.65	1.76	0.00
		20	9,099,306.61	14.39	3.82	2.96	0.00
		25	7,962,255.48	18.11	1.58	0.60	0.00
		30	7,271,676.07	18.87	1.57	0.73	0.00
Sweden Data	24,978	5	35,535,818.26	4.90	2.93	1.06	0.00
		10	24,053,092.74	11.39	5.15	3.99	0.00
		15	19,640,207.74	15.13	1.89	1.56	0.00
		20	16,973,915.83	15.79	1.44	0.97	0.00
		25	15,092,119.15	13.98	3.96	2.87	0.00
		30	14,018,174.94	14.72	0.91	0.87	0.00
Burma Data	33,708	5	47,320,059.91	6.42	0.00	0.00	0.00
		10	31,891,058.04	13.70	5.05	1.54	0.00
		15	25,878,057.99	10.41	3.81	1.03	0.00
		20	22,486,038.51	11.66	1.92	1.92	0.00
		25	20,146,402.36	15.33	0.49	0.45	0.00
		30	17,941,167.95	16.87	2.22	0.93	0.00
China Data	71,009	5	352,667,929.12	7.29	0.20	0.13	0.00
		10	302,130,384.31	6.64	0.98	0.92	0.00
		15	280,553,532.95	7.65	1.77	0.14	0.00
		20	266,313,391.35	6.65	2.78	1.61	0.00
		25	262,397,582.09	6.70	1.08	0.83	0.00
		30	256,990,798.52	6.40	1.17	0.58	0.00
Average				11.22	2.12	1.25	**0.00**

Table A2. The comparison in solution quality between multi-start and RVNS with CM, MM, and QM: The case of the *p*-centre problem

Description	*n*	*p*	Best *Z*	Deviation (%)			
				Multi-start	RVNS		
					CM	MM	QM
Italy Data	16,862	5	3,564.99	5.72	0.00	0.00	0.00
		10	2,073.72	24.24	2.84	0.24	0.00
		15	1,923.25	11.56	0.56	0.20	0.00
		20	1,460.31	37.50	1.88	0.05	0.00
		25	1,595.22	9.82	1.39	1.39	0.00
		30	1,168.21	43.94	5.63	1.33	0.00
Sweden Data	24,978	5	4,178.05	0.30	0.00	0.00	0.00
		10	2,973.40	5.49	21.42	0.00	0.00
		15	3,136.70	0.00	11.26	6.27	6.27
		20	2,639.97	9.66	13.17	13.17	0.00
		25	2,055.48	32.14	9.03	7.68	0.00
		30	2,716.05	0.00	15.58	15.58	15.58
Burma Data	33,708	5	2,847.46	42.39	2.22	0.23	0.00
		10	2,095.30	34.19	10.52	8.37	0.00
		15	1,920.14	32.74	3.51	0.00	0.00
		20	2,108.84	9.11	0.00	0.00	0.00
		25	1,618.04	36.45	0.00	0.00	0.00
		30	1,598.09	26.28	0.17	0.17	0.00
China Data	71,009	5	12,983.84	72.00	0.00	0.00	0.00
		10	11,529.30	92.23	2.98	2.98	0.00
		15	13,465.98	64.45	0.00	0.00	0.00
		20	11,620.38	90.30	0.65	0.00	0.00
		25	11,292.48	95.83	1.00	1.00	0.00
		30	10,586.92	108.50	1.10	1.10	0.00
Average				36.87	4.37	2.49	**0.91**

References

1. Al-Zoubi, M.B., Al-Rawi, M. (2008). An efficient approach for computing silhouette coefficients. *Journal of Computer Science, 4*, 252-255.
2. Andersson, G., Francis, R.L., Normark, T., Rayco, M.B. (1998). Aggregation method experimentation for large-scale network location problems. *Location Science, 6*, 25-39.
3. Avella, P., Boccia, M., Salerno, S., & Vasilyev, I. (2012). An aggregation heuristic for large scale p-median problem. *Computers and Operations Research, 39*, 1625-1632.
4. Bowerman, R.L., Calamai, P.H., Brent, H.G. (1999). The demand partitioning method for reducing aggregation errors in *p*-median problems. *Computers & Operations Research, 26*, 1097-1111.
5. Current, J.R, Schilling, D.A. (1987). Elimination of source A and B errors in *p*-median location problems. *Geographical Analysis, 19*, 95-110.
6. Erkut, E., Bozkaya, B. (1999). Analysis of aggregation errors for the p-median problem. *Computers & Operations Research, 26*, 1075-1096.
7. Francis, R.L., Lowe, T.J., Rayco, M.B. (1996). Row-column aggregation for rectilinear distance *p*-median problems. *Transportation Science, 30*, 160-174.
8. Francis, R.L., Lowe, T.J., Rayco, M.B., Tamir, A. (2003). Exploiting self-canceling demand point aggregation error for some planar rectilinear median location problems. *Naval Research Logistics, 50*, 614-637.
9. Francis, R.L., Lowe, T.J., Rayco, M.B., Tamir, A. (2009). Aggregation error for location models: survey and analysis. *Annals of Operations Research, 167*, 171-208.
10. Francis, R.L., Lowe, T.J., Tamir, A. (2000). Aggregation error bounds for a class of location models. *Operations Research, 48,* 294.
11. Francis, R.L., Lowe, T.J. (1992). On worst-case aggregation analysis for network location problems. *Annals of Operations Research, 40*, 229-246.
12. García, S., Labbé, M., & Marín, A. (2010). Solving large *p*-median problem with a radius formulation. *INFORMS Journal on Computing, 23*, 546-556.
13. Hansen, P., Brimberg, J., Urosevic, D., & Mladenovic, N. (2009). Solving large *p*-median clustering problems by primal-dual variable neighborhood search. *Data Mining and Knowledge Discovery, 19*, 351-375.
14. Hansen, P., Mladenovic, N., & Perez, J. A. M. (2010). Variable neighbourhood search: methods and applications. *Annals of Operations Research, 175*, 367-407.

15. Hillsman, E.L., Rhoda, R. (1978). Errors in measuring distances from populations to service centers. *Annals of Regional Science, 12*, 74-88.

16. Hodgson, M.J., Salhi, S. (1998). Using a quadtree structure to eliminate aggregation error in point to point allocation. Presented at INFORS Conference, Montreal.

17. Hodgson, M.J., Hewko, J. (2003). Aggregation and surrogation error in the *p*-median model. *Annals of Operations Research, 123*, 53-66.

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

19. Hodgson, M.J., Shmulevitz, F., Korkel, M. (1997). Aggregation error effects on the discrete-space *p*-median model: The case of Edmonton, Canada. *Canadian Geographer / Le Géographe canadien, 41*, 415-428.

20. Irawan, C.A., Salhi, S. (2013). Solving large *p*-median problems by a multistage hybrid approach using demand points aggregation and variable neighbourhood search. *Journal of Global Optimization*. doi :10.1007/s10898-013-0080-z.

21. Irawan, C.A., Salhi, S., Scaparra, M.P. (2014). An adaptive multiphase approach for large unconditional and conditional *p*-median problems. *European Journal of Operational Research,* 237, 590–605.

22. Noaves, A.G.N., Souza de Cursi, J.E., da Silva, A.C.L, Souza, J.C. (2009). Solving continuous location-districting problems with Voronoi diagrams. *Computers and Operations Research, 36*, 40-59.

23. Qi, L., & Shen, Z. M. (2010). Worst-case analysis of demand point aggregation for the Euclidean *p*-median problem. *European Journal of Operational Research, 202*, 434-443.

24. Samet, H. (1990). *The design and analysis of spatial data structures*. Addison-Wesley, Reading, MA.