Electrical, Control and Communication Engineering
ISSN 2255-9159 (online)
ISSN 2255-9140 (print)
2023, vol. 19, no. 1, pp. 29–39
https://doi.org/10.2478/ecce-2023-0005
https://content.sciendo.com

Optimized Centroid-Based Clustering of Dense Nearly-square Point Clouds by the Hexagonal Pattern

Vadim Romanuke* (*Vinnytsia Institute of Trade and Economics of State University of Trade and Economics, Vinnytsia, Ukraine*)
Svitlana Merinova (*Vinnytsia Institute of Trade and Economics of State University of Trade and Economics, Vinnytsia, Ukraine*)
Hanna Yehoshyna (*Northwestern Polytechnic, Grande Prairie, Canada*)

Abstract **– An approach to optimize centroid-based clustering of flat objects is suggested, which is practically important for efficiently solving metric facility location problems. In such problems, the task is to find the best warehouse locations to optimally service a given set of consumers. An example is assigning mobiles to base stations of a wireless communication network. We suggest a hexagonal-pattern-based approach to partition flat nodes into clusters quicker than the *k*-means algorithm and its modifications do. First, a hexagonal cell lattice is applied to nodes to approximately determine centroids of the clusters. Then the centroids are used as initial centroids to start the *k*-means algorithm. The suggested method is efficient for centroid-based clustering of dense nearly-square point clouds of 0.1 million points and greater by using no fewer than 6 lattice cells along an axis. Compared to *k*-means, our method is at least 10 % faster and it is about 0.01 to 0.07 % more accurate in regular Euclidean distances. In squared Euclidean distances, the accuracy gain is 0.14 to 0.21 %. Applying a hexagonal cell lattice determines an upper bound of the clustering quality gap.**

Keywords **– Centroid-based clustering, hexagonal pattern, initialization, square cloud.**

I. Challenge in Centroid-Based Clustering

Clustering flat objects, among the others, has many practical implementations such as image analysis [1], object recognition [2], anomaly detection [3], [4], determining structural similarity of chemicals (in mathematical chemistry) [5], [6], finding weather regimes or preferred sea level pressure atmospheric patterns (in climatology) [7], financial analysis and stock price comovement [8], geological data analysis [9], etc. Another implementation, specifically of centroid-based clustering, consists in assigning mobiles to base stations of a wireless communication network [10], [11]. In general, clustering flat objects is a metric facility location problem, where the task is to find the best warehouse locations to optimally service a given set of consumers. Warehouses are seen as cluster centres (centroids) and the data to be clustered are seen as consumer locations [12], [13].

The centroid-based clustering approach that is the most referred to and used is the method of *k*-means and its modifications like *k*-medoids and *k*-means++ [14], [15]. In general, the centroid-based clustering problem is to partition N observations into k clusters by minimizing the sum of within-cluster squared Euclidean distances [16]. It is an NP-hard optimization problem, so the *k*-means algorithm was developed as an efficient heuristic to converge quickly to a local optimum, i. e., an approximate solution of an acceptable inaccuracy. A challenge exists when too many points (even two-dimensional or, in other words, flat objects) are to be clustered, and, in addition, when the number of clusters is increased. Then the algorithm slows down significantly. For instance, a set of 12 500 points scattered uniformly within a unit square is partitioned into 64 clusters within 0.32 s on a dual-core processor Intel Core i5-7200U@2.50GHz. When the dataset is enlarged twice (within the same square), it takes up to 1.7 s (the computational time variation exists because it depends on the initialization). Doubling the dataset once more, results in the 64 clusters of 50 000 points are obtained in about 2.7 s (Fig. 1).

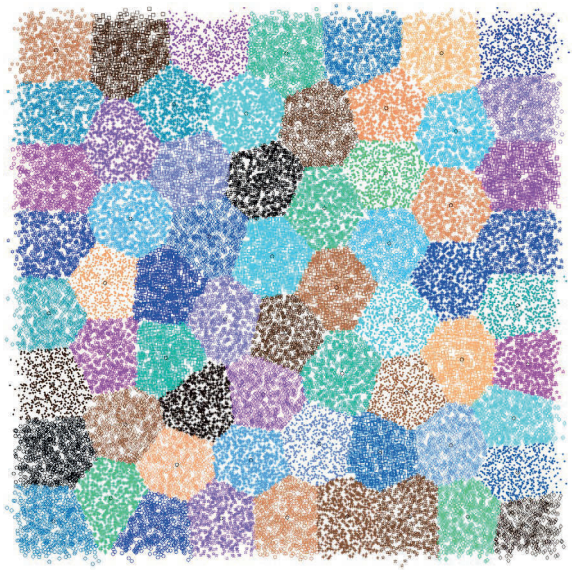

Fig. 1. An example of 50 000 flat points partitioned into 64 clusters by using the *k*-means algorithm. The centroids are marked with circles. The points are scattered such that the point cloud is nearly square. The inner clusters mostly constitute slightly contorted irregular hexagons. The boundary clusters mostly are contorted irregular pentagons with the outer sides "torn" in the visualization.

* Corresponding author. E-mail: v.romanyuk@vtei.edu.ua

According to Fig. 1, one can clearly see that the resulting cluster areas closely resemble irregular polygons, which mostly are contorted irregular hexagons inside the nearly-square point cloud and are contorted irregular pentagons on its boundaries. Moreover, the cluster centroids in this particular example are located such that they remind vertices of a contorted 8-by-8 lattice. Nevertheless, there are no clearly distinguishable eight rows and eight columns of the polygons embracing the clusters.

II. Motivation, Aim, and Tasks to be Accomplished

The centroid-based clustering algorithms do not guarantee convergence to the global optimum [14], [15], [17]. The result depends on choosing initial centroids of the clusters before proceeding with the algorithm iterations [15], [18], [19]. Different initializations and subsequent multiple runs of the algorithm result in sets of clusters that differ in locations of their centroids, cluster size, and eventual sum of within-cluster squared Euclidean distances [14], [18]. Owing to this and to the fact that the *k*-means algorithm is usually fast, it is run multiple times with different initializations (i. e., the starting set of clusters). Thereupon the best result, whose sum is the least, is selected as the solution [15], [17], [20], [21].

The worst-case convergence, when the running time is exponential [22], occurs rarely only for particular certain point sets, not like those in Fig. 1 or similar "box-like" point sets. The smoothed running time of the *k*-means algorithm is polynomial [14], [23]. Moreover, the algorithm is often considered to be of roughly linear complexity in cases, where the data do have a clustering structure (or similar to that in Fig. 1). In such cases, the number of iterations until convergence is often small, and the *k*-means algorithm just slightly improves the clustering result after a few starting iterations [18]–[20].

Furthermore, the algorithm tends to determine clusters of comparable spatial extent (similarly to the clusters in Fig. 1). It spends a lot of processing time computing the distances between each of the k centroids and the N data points. Since points usually stay in the same clusters after a few iterations, much of these operations are useless, making the algorithm very inefficient. In particular, the result in Fig. 1 prompts that clusters in such dense nearly-square point clouds tend to have a form of an irregular polygon, mostly being hexagon. It is natural to expect that assigning points to clusters whose borders are defined by hexagons would not worsen much the quality of clustering. Therefore, the aim is to develop a method that could speed up clustering dense nearly-square point clouds by using a hexagonal pattern. To achieve the aim, the following six standard tasks are to be accomplished:

1. To suggest a hexagonal-pattern-based approach to partition flat points into clusters quicker than the *k*-means algorithm and its modifications do.

2. To suggest a method of measuring the quality of clustering by the suggested approach, including both the accuracy and computational time. The quality of clustering should be inherited by its efficiency.

3. To organise a simulation set-up for gathering statistics of clustering nearly-square point clouds by using the *k*-means and *k*-medoids algorithms along with the suggested clustering approach.

4. Based on the simulation results, to ascertain whether the suggested approach is efficient. If it is efficient, determine limits within which this efficiency is maintained.

5. Whichever the efficiency is, to discuss applicability and significance of the suggested approach.

6. To conclude on the contribution and a possibility of further research.

III. Hexagonal Pattern

Let us denote by $P=\{\mathbf{P}_i=[x_i \quad y_i]\}_{i=1}^{N}$ a set of N two-dimensional points (objects), where x_i and y_i are the horizontal and vertical components, respectively. Given an initial set of k centroids $\{\mathbf{C}_j(1)=[a_j^{(1)} \quad b_j^{(1)}]\}_{j=1}^{k}$ of the clusters, the *k*-means algorithm proceeds by alternating between the assignment and update steps [16], [20], [22], [24]. At assignment step s, cluster j is a set

$$T_j(s)=\Big\{[x_i \quad y_i]:\left(x_i-a_j^{(s)}\right)^2+\left(y_i-b_j^{(s)}\right)^2\leqslant \leqslant\left(x_i-a_q^{(s)}\right)^2+\left(y_i-b_q^{(s)}\right)^2\ \forall q=\overline{1,\,k}\Big\}\ \text{by}\ s=1,2,\ldots, \quad (1)$$

where each point is assigned to exactly one cluster. At the update step, which follows (1), the centroids are recalculated as

$$\mathbf{C}_j(s+1)=\left[a_j^{(s+1)} \quad b_j^{(s+1)}\right]= =\frac{1}{\left|T_j(s)\right|}\cdot\sum_{\mathbf{P}_i\in T_j(s)}\mathbf{P}_i,\ s=1,2,\ldots \quad (2)$$

In *k*-means, the cluster centroid is not necessarily one of the points of the given set $P=\{[x_i \quad y_i]\}_{i=1}^{N}$, but it is just the average (mean) of the points in the cluster. In contrast to the *k*-means algorithm, *k*-medoids takes $\mathbf{C}_j(s)\in P$ at every step s (i. e., it chooses actual data points as centroids referred to as medoids), and thereby allows for greater interpretability of the cluster centroids than in *k*-means.

A hexagonal cell lattice of size $M\times M$ is created by enclosing nodes (1) within rectangle

$$\left[\min_{i=\overline{1,\,N}}x_i-\frac{\max\limits_{i=\overline{1,\,N}}x_i-\min\limits_{i=\overline{1,\,N}}x_i}{2M-1};\ \max_{i=\overline{1,\,N}}x_i+\frac{\max\limits_{i=\overline{1,\,N}}x_i-\min\limits_{i=\overline{1,\,N}}x_i}{2M-1}\right]\times \times\left[\min_{i=\overline{1,\,N}}y_i-\frac{\max\limits_{i=\overline{1,\,N}}y_i-\min\limits_{i=\overline{1,\,N}}y_i}{3M-1};\ \max_{i=\overline{1,\,N}}y_i+\frac{\max\limits_{i=\overline{1,\,N}}y_i-\min\limits_{i=\overline{1,\,N}}y_i}{3M-1}\right]. \quad (3)$$

The area within rectangle (3) is uniformly broken into M^2 hexagonal cells whose vertices have horizontal

$$\left[h_l^{(\text{hor})}\left(m_{\text{hor}},m_{\text{vert}}\right)\right]_{1\times 6}= =\min_{i=\overline{1,\,N}}x_i+\left[0 \quad \frac{\sqrt{3}}{2} \quad \sqrt{3} \quad \sqrt{3} \quad \frac{\sqrt{3}}{2} \quad 0\right]\cdot\frac{\max\limits_{i=\overline{1,\,N}}x_i-\min\limits_{i=\overline{1,\,N}}x_i}{(2M-1)\frac{\sqrt{3}}{2}}-$$

$$-\frac{\max\limits_{i=\overline{1,N}} x_i - \min\limits_{i=\overline{1,N}} x_i}{2M_{\text{hor}}-1} + \left(m_{\text{vert}} - 2\psi\left(\frac{m_{\text{vert}}}{2}\right)\right)\times$$
$$\times \frac{\max\limits_{i=\overline{1,N}} x_i - \min\limits_{i=\overline{1,N}} x_i}{2M-1} + 2\cdot(m_{\text{hor}}-1)\cdot\frac{\max\limits_{i=\overline{1,N}} x_i - \min\limits_{i=\overline{1,N}} x_i}{2M-1}$$
$$\text{for } m_{\text{hor}} = \overline{1,\, M} \text{ and } m_{\text{vert}} = \overline{1,\, M} \qquad (4)$$

and vertical

$$\left[h_l^{(\text{vert})}(m_{\text{hor}}, m_{\text{vert}})\right]_{1\times 6} =$$
$$= \min_{i=\overline{1,N}} y_i + \left[\frac{3}{2} \quad 2 \quad \frac{3}{2} \quad \frac{1}{2} \quad 0 \quad \frac{1}{2}\right]\cdot\frac{2\cdot\left(\max\limits_{i=\overline{1,N}} y_i - \min\limits_{i=\overline{1,N}} y_i\right)}{3M-1} -$$
$$-\frac{\max\limits_{i=\overline{1,N}} y_i - \min\limits_{i=\overline{1,N}} y_i}{3M-1} + (m_{\text{vert}}-1)\cdot\frac{3\cdot\left(\max\limits_{i=\overline{1,N}} y_i - \min\limits_{i=\overline{1,N}} y_i\right)}{3M-1}$$
$$\text{for } m_{\text{hor}} = \overline{1,\, M} \text{ and } m_{\text{vert}} = \overline{1,\, M} \qquad (5)$$

coordinates at hexagonal location $[m_{\text{hor}} \quad m_{\text{vert}}]$, where function $\psi(x)$ returns the integer part of number x. A convex hull over vertices

$$\left\{ h_i^{(\text{hor})}(m_{\text{hor}}, m_{\text{vert}}), h_i^{(\text{vert})}(m_{\text{hor}}, m_{\text{vert}})\right\}_{i=1}^{6}$$

in $\mathbb{R}^2$ is a hexagon $X(m_{\text{hor}}, m_{\text{vert}})$ that can be obtained by stretching (either horizontally or vertically) the respective regular hexagon [25]. This hexagon encloses nodes belonging to the respective cluster created as a set

$$H(m_{\text{hor}}, m_{\text{vert}}) =$$
$$= \left\{ \mathbf{H}_i(m_{\text{hor}}, m_{\text{vert}}) = \left[\hat{x}_i(m_{\text{hor}}, m_{\text{vert}}) \quad \hat{y}_i(m_{\text{hor}}, m_{\text{vert}})\right] = \right.$$
$$\left. = [x_i \quad y_i] \in P : [x_i \quad y_i] \in X(m_{\text{hor}}, m_{\text{vert}})\right\}$$
$$\text{for } m_{\text{hor}} = \overline{1,\, M} \text{ and } m_{\text{vert}} = \overline{1,\, M}\,. \qquad (6)$$

Running operations of checking the membership in (6) is far faster than running iterations by (1) and (2) [26], [27]. Thus, this hexagonal-pattern-based approach partitions flat points into clusters quicker than the *k*-means algorithm and its modifications do.

Obviously, centroids of the clusters created by (3)–(6) are points

$$\mathbf{C}_{\text{hex}}(m_{\text{hor}}, m_{\text{vert}}) = \left[x_{\text{hex}}(m_{\text{hor}}, m_{\text{vert}}) \quad y_{\text{hex}}(m_{\text{hor}}, m_{\text{vert}})\right] = \frac{1}{\left|H(m_{\text{hor}}, m_{\text{vert}})\right|}\cdot \sum_{\mathbf{H}_i(m_{\text{hor}}, m_{\text{vert}})\in H(m_{\text{hor}}, m_{\text{vert}})} \mathbf{H}_i(m_{\text{hor}}, m_{\text{vert}}) =$$
$$= \left[\sum_{\mathbf{H}_i(m_{\text{hor}}, m_{\text{vert}})\in H(m_{\text{hor}}, m_{\text{vert}})} \hat{x}_i(m_{\text{hor}}, m_{\text{vert}}) \quad \sum_{\mathbf{H}_i(m_{\text{hor}}, m_{\text{vert}})\in H(m_{\text{hor}}, m_{\text{vert}})} \hat{y}_i(m_{\text{hor}}, m_{\text{vert}})\right] \in \mathbb{R}^2 \text{ for } m_{\text{hor}} = \overline{1,\, M} \text{ and } m_{\text{vert}} = \overline{1,\, M} \qquad (7)$$

that are not necessarily the vertices

$$\mathbf{V}_{\text{hex}}(m_{\text{hor}}, m_{\text{vert}}) = \left[u_{\text{hex}}(m_{\text{hor}}, m_{\text{vert}}) \quad w_{\text{hex}}(m_{\text{hor}}, m_{\text{vert}})\right] =$$
$$= \left[\frac{1}{6}\cdot\sum_{i=1}^{6} h_i^{(\text{hor})}(m_{\text{hor}}, m_{\text{vert}}) \quad \frac{1}{6}\cdot\sum_{i=1}^{6} h_i^{(\text{vert})}(m_{\text{hor}}, m_{\text{vert}})\right] \in \mathbb{R}^2$$
$$\text{for } m_{\text{hor}} = \overline{1,\, M} \text{ and } m_{\text{vert}} = \overline{1,\, M} \qquad (8)$$

of the hexagonal cell $M\times M$ lattice (that can be obtained by stretching, either horizontally or vertically, the respective $M\times M$ lattice of regular hexagons). Similarly to *k*-means, cluster centroid (7) is not necessarily one of the points of the given set $P = \{[x_i \quad y_i]\}_{i=1}^{N}$. However, unlike *k*-means and *k*-medoids, the centroids do not matter while a set of points is clustered by the hexagonal pattern by (3)–(6).

An example of applying the hexagonal pattern by (3)–(6) to nodes from Fig. 1 is shown in Fig. 2. The 64 clusters of 50 000 points are obtained in about 0.09 s that is 30 times faster than the result in Fig. 1 produced by *k*-means. It is clear that the protruding clusters on the left and right can be 50 % filled at most. The non-filled area of the bottom and upper boundary clusters is, for a unit regular hexagon (whose side length is equal to a unit), equal to $\sqrt{3}/4$ units. The area of a unit regular hexagon is

$$\sqrt{3}/4 + 1\cdot\sqrt{3} + \sqrt{3}/4 = \frac{3\sqrt{3}}{2} \text{ units.}$$

Therefore, such clusters can be filled at most by 5/6 of the hexagon area.

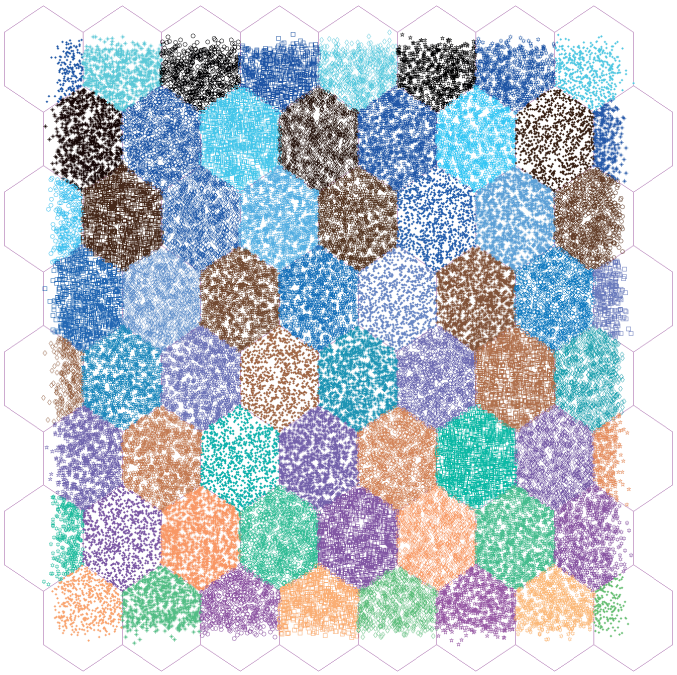

Fig. 2. An example of applying the hexagonal pattern by (3)–(6) to nodes from Fig. 1 for creating 64 clusters. The hexagonal cell 8×8 lattice is also shown. The four protruding clusters on the left and four ones on the right are nearly half-filled. The bottom and upper boundary clusters are nearly 83.33 % filled as well.

Despite the centroid-based clustering by the hexagonal pattern is very fast, it is less accurate. In the example in Fig. 2, the sum of within-cluster squared Euclidean distances, with respect to hexagonal centroids (7), is 1.0509 times greater than that in Fig. 1. Nevertheless, the quality of clustering must not be considered only by the accuracy.

IV. Quality of Clustering

At step s of k-means (k-medoids), the abovementioned sum of within-cluster squared Euclidean distances is calculated as

$$D(s)=\sum_{j=1}^{k}\left(\sum_{\mathbf{P}_i\in T_j(s)}\left[\left(x_i-a_j^{(s)}\right)^2+\left(y_i-b_j^{(s)}\right)^2\right]\right). \quad (9)$$

In the centroid-based clustering algorithms, sum (9) is theoretically minimized. In practice, the algorithm is kept running while

$$\sqrt{\left(a_m^{(s)}-a_m^{(s-1)}\right)^2+\left(b_m^{(s)}-b_m^{(s-1)}\right)^2}\geqslant\varepsilon \text{ for some } \varepsilon>0 \quad (10)$$

at least for cluster m and the maximal number of iterations is not exceeded. Along with (9), the regular Euclidean distances can be used to estimate the accuracy:

$$R(s)=\sum_{j=1}^{k}\left(\sum_{\mathbf{P}_i\in T_j(s)}\sqrt{\left(x_i-a_j^{(s)}\right)^2+\left(y_i-b_j^{(s)}\right)^2}\right). \quad (11)$$

The accuracy of the hexagonal-pattern-based clustering is measured similarly to (9) and (11):

$$D_{\text{hex}}=\sum_{m_{\text{hor}}=1}^{M}\sum_{m_{\text{vert}}=1}^{M}\left(\sum_{\mathbf{P}_i\in H(m_{\text{hor}},\,m_{\text{vert}})}\left(\left[x_i-x_{\text{hex}}\left(m_{\text{hor}},m_{\text{vert}}\right)\right]^2+\left[y_i-y_{\text{hex}}\left(m_{\text{hor}},m_{\text{vert}}\right)\right]^2\right)\right), \quad (12)$$

$$R_{\text{hex}}=\sum_{m_{\text{hor}}=1}^{M}\sum_{m_{\text{vert}}=1}^{M}\left(\sum_{\mathbf{P}_i\in H(m_{\text{hor}},\,m_{\text{vert}})}\sqrt{\left[x_i-x_{\text{hex}}\left(m_{\text{hor}},m_{\text{vert}}\right)\right]^2+\left[y_i-y_{\text{hex}}\left(m_{\text{hor}},m_{\text{vert}}\right)\right]^2}\right). \quad (13)$$

The quality of clustering also depends on how fast it is. Obviously, the hexagonal-pattern-based clustering is fast enough compared to both k-means and k-medoids. Then, if value (12) was tolerably greater than value (9), the hexagonal approach would be efficient.

V. Simulation Set-up

To see how efficient the suggested clustering approach is, let the number of data points be

$$N\in\left\{1000,5000,25000,10^5,5\cdot10^5,10^6\right\} \quad (14)$$

and define the number of lattice cells by

$$M\in\left\{\overline{2,10}\right\}. \quad (15)$$

To generate nearly-square point clouds, every point

$$\left[x_i\quad y_i\right]=\left[\xi_i\quad\zeta_i\right] \text{ for } i=\overline{1,N} \quad (16)$$

where ξ_i and ζ_i are values of two independent random variables distributed uniformly on the open interval (0;1).

Let us denote by D_{hex}^* and R_{hex}^* the approximate minima of sums (12) and (13), respectively. Similarly, let us denote by D_{means}^* and R_{means}^* the approximate minima of sums (9) and (11) calculated for clusters partitioned by k-means. When k-medoids is used, these are denoted by D_{medoids}^* and R_{medoids}^*. In addition to the three approaches to clustering, the fourth one can be based on a specific initialization of the k-means algorithm. In this case,

$$\mathbf{C}_j(1)=\left[a_j^{(1)}\quad b_j^{(1)}\right]=\mathbf{C}_{\text{hex}}\left(m_{\text{hor}},m_{\text{vert}}\right)=$$
$$=\left[x_{\text{hex}}\left(m_{\text{hor}},m_{\text{vert}}\right)\quad y_{\text{hex}}\left(m_{\text{hor}},m_{\text{vert}}\right)\right]$$
$$\text{by } j=\left(m_{\text{hor}}-1\right)\cdot M+m_{\text{vert}}$$
$$\text{for } m_{\text{hor}}=\overline{1,M} \text{ and } m_{\text{vert}}=\overline{1,M}, \quad (17)$$

i. e. centroids (7) of the clusters created by the hexagonal pattern are used as an initial set of k centroids

$$\left\{\mathbf{C}_j(1)=\left[a_j^{(1)}\quad b_j^{(1)}\right]\right\}_{j=1}^{k}$$

of the clusters. Let us denote by $D_{\text{hex-means}}^*$ and $R_{\text{hex-means}}^*$ the approximate minima of sums (9) and (11) for this hexagon-based k-means approach. The respective average time taken by these four approaches is denoted by t_{hex}, t_{means}, t_{medoids}, $t_{\text{hex-means}}$. Alternatively, the initial set of k centroids

$$\left\{\mathbf{C}_j(1)=\left[a_j^{(1)}\quad b_j^{(1)}\right]\right\}_{j=1}^{k}$$

of the clusters is of vertices (8) of the hexagonal cell $M\times M$ lattice:

$$\mathbf{C}_j(1)=\left[a_j^{(1)}\quad b_j^{(1)}\right]=\mathbf{V}_{\text{hex}}\left(m_{\text{hor}},m_{\text{vert}}\right)=$$
$$=\left[u_{\text{hex}}\left(m_{\text{hor}},m_{\text{vert}}\right)\quad w_{\text{hex}}\left(m_{\text{hor}},m_{\text{vert}}\right)\right]$$
$$\text{by } j=\left(m_{\text{hor}}-1\right)\cdot M+m_{\text{vert}}$$
$$\text{for } m_{\text{hor}}=\overline{1,M} \text{ and } m_{\text{vert}}=\overline{1,M}. \quad (18)$$

For this lattice-based k-means approach, let us denote by $D_{\text{lat-means}}^*$ and $R_{\text{lat-means}}^*$ the approximate minima of sums (9) and (11), and the respective average time taken by this approach is denoted by $t_{\text{lat-means}}$.

For gathering "internally" reliable statistics, each algorithm is run 10 times for every pair $\{N, M\}$, whence the minimal sum

of the 10 minima is selected. The computational time is averaged over those 10 runs. The accuracy gains are calculated for both squared and regular Euclidean distances as

$$\delta_{\text{means\_hex}} = \frac{D^*_{\text{means}}}{D^*_{\text{hex}}}, \tag{19}$$

$$\delta_{\text{medoids\_hex}} = \frac{D^*_{\text{medoids}}}{D^*_{\text{hex}}}, \tag{20}$$

$$\delta_{\text{hex-means\_hex}} = \frac{D^*_{\text{hex-means}}}{D^*_{\text{hex}}}, \tag{21}$$

$$\delta_{\text{lat-means\_hex}} = \frac{D^*_{\text{lat-means}}}{D^*_{\text{hex}}}, \tag{22}$$

$$\delta_{\text{means\_hex-means}} = \frac{D^*_{\text{means}}}{D^*_{\text{hex-means}}}, \tag{23}$$

$$\delta_{\text{medoids\_hex-means}} = \frac{D^*_{\text{medoids}}}{D^*_{\text{hex-means}}}, \tag{24}$$

$$\delta_{\text{means\_lat-means}} = \frac{D^*_{\text{means}}}{D^*_{\text{lat-means}}}, \tag{25}$$

$$\delta_{\text{medoids\_lat-means}} = \frac{D^*_{\text{medoids}}}{D^*_{\text{lat-means}}}, \tag{26}$$

$$\rho_{\text{means\_hex}} = \frac{R^*_{\text{means}}}{R^*_{\text{hex}}}, \tag{27}$$

$$\rho_{\text{medoids\_hex}} = \frac{R^*_{\text{medoids}}}{R^*_{\text{hex}}}, \tag{28}$$

$$\rho_{\text{hex-means\_hex}} = \frac{R^*_{\text{hex-means}}}{R^*_{\text{hex}}}, \tag{29}$$

$$\rho_{\text{lat-means\_hex}} = \frac{R^*_{\text{lat-means}}}{R^*_{\text{hex}}}, \tag{30}$$

$$\rho_{\text{means\_hex-means}} = \frac{R^*_{\text{means}}}{R^*_{\text{hex-means}}}, \tag{31}$$

$$\rho_{\text{medoids\_hex-means}} = \frac{R^*_{\text{medoids}}}{R^*_{\text{hex-means}}}, \tag{32}$$

$$\rho_{\text{means\_lat-means}} = \frac{R^*_{\text{means}}}{R^*_{\text{lat-means}}}, \tag{33}$$

$$\rho_{\text{medoids\_lat-means}} = \frac{R^*_{\text{medoids}}}{R^*_{\text{lat-means}}}. \tag{34}$$

The speedup gain is calculated as

$$\tau_{\text{means\_hex}} = \frac{t_{\text{means}}}{t_{\text{hex}}}, \tag{35}$$

$$\tau_{\text{medoids\_hex}} = \frac{t_{\text{medoids}}}{t_{\text{hex}}}, \tag{36}$$

$$\tau_{\text{hex-means\_hex}} = \frac{t_{\text{hex-means}}}{t_{\text{hex}}}, \tag{37}$$

$$\tau_{\text{lat-means\_hex}} = \frac{t_{\text{lat-means}}}{t_{\text{hex}}}, \tag{38}$$

$$\tau_{\text{means\_hex-means}} = \frac{t_{\text{means}}}{t_{\text{hex-means}}}, \tag{39}$$

$$\tau_{\text{medoids\_hex-means}} = \frac{t_{\text{medoids}}}{t_{\text{hex-means}}}, \tag{40}$$

$$\tau_{\text{means\_lat-means}} = \frac{t_{\text{means}}}{t_{\text{lat-means}}}, \tag{41}$$

$$\tau_{\text{medoids\_lat-means}} = \frac{t_{\text{medoids}}}{t_{\text{lat-means}}}. \tag{42}$$

Overall the simulation by (14)–(16) is to be repeated for 20 times, whereupon subsequent gains (19)–(42) are averaged. This will ensure statistical stability and reliability of the performance results by each of the approaches.

VI. Efficiency

The accuracy gains (19)–(26) are presented in Table I, where the right-side column presents the gains averaged over the number of lattice cells. It is clearly seen that the hexagonal-pattern-based clustering is 4 to 11 % less accurate than *k*-means, but the accuracy loss with respect to *k*-medoids is slightly less. The accuracy losses to both hexagon-based *k*-means and lattice-based *k*-means are almost the same. Besides, the latter two approaches outperform both *k*-means and *k*-medoids by up to 2 % (highlighted bold whenever the gain exceeds 1). The only exception is the case of 1000 points, where *k*-means slightly outperforms (by less than 0.7 %). The best result is achieved at a dataset of 5000 points by using the 10×10 lattice for hexagon-based *k*-means versus *k*-medoids (it is almost 7 % accuracy gain).

The accuracy gains (27)–(34) are presented in Table II, where the right-side column presents the gains averaged over *M*. Here the differences by the regular Euclidean distances is shorter than those in Table I. Starting from 0.1 million points and greater, both hexagon-based *k*-means and lattice-based *k*-means outperform *k*-means and *k*-medoids by 0.01 to 0.85 %. The best result is achieved at 25 000 points by using the 10×10 lattice for hexagon-based *k*-means versus *k*-medoids.

The speedup gains (35)–(42) presented in Table III confirm that the hexagonal-pattern-based clustering is much faster. The speedup gain grows as the dataset becomes larger, but this quasi-regularity is violated when either of hexagon-based *k*-means and lattice-based *k*-means is compared to *k*-means. In this case, nevertheless, these approaches are faster at least by a few percent.

TABLE I

ACCURACY GAINS (19)–(26)

		M									
	N, million points	2	3	4	5	6	7	8	9	10	Average
δ_{means_hex}	0.001	0.87229	0.89676	0.9025	0.90376	0.8978	0.89312	0.8919	0.88166	0.86154	0.88903
	0.005	0.87063	0.90947	0.92372	0.93605	0.94397	0.9425	0.94313	0.94078	0.93626	0.92739
	0.025	0.87399	0.90767	0.92979	0.94125	0.95137	0.95757	0.95803	0.96061	0.96106	0.93792
	0.1	0.87251	0.90949	0.93201	0.94482	0.95569	0.96186	0.96494	0.96922	0.96972	0.94225
	0.5	0.87241	0.90922	0.93262	0.94643	0.95692	0.96304	0.96807	0.97135	0.97407	0.94379
	1	0.87253	0.90965	0.93285	0.94621	0.95746	0.9639	0.96836	0.97204	0.97415	0.94413
$\delta_{medoids\_hex}$	0.001	0.87599	0.9048	0.91254	0.9137	0.91821	0.91297	0.91248	0.90101	0.92356	0.90836
	0.005	0.87161	0.91114	0.92963	0.95183	0.96276	0.9724	0.96779	0.96855	0.99601	0.94797
	0.025	0.87415	0.90802	0.93787	0.95806	0.97421	0.99114	0.98963	0.99372	**1.01696**	0.96042
	0.1	0.87256	0.90962	0.93924	0.95903	0.97358	0.98582	0.99061	**1.00165**	**1.00889**	0.96011
	0.5	0.87241	0.90926	0.9336	0.95431	0.96789	0.97324	0.98144	0.98792	0.99264	0.95252
	1	0.87253	0.90967	0.9336	0.95326	0.96466	0.97174	0.97838	0.98393	0.98727	0.95056
$\delta_{hex\text{-}means\_hex}$	0.001	0.87237	0.90039	0.90985	0.91035	0.90605	0.90001	0.89751	0.88634	0.86878	0.89463
	0.005	0.87064	0.90971	0.92739	0.93616	0.94209	0.94104	0.94099	0.93846	0.93088	0.92637
	0.025	0.87399	0.90777	0.93131	0.94329	0.95006	0.95466	0.95624	0.95948	0.95893	0.9373
	0.1	0.87251	0.90952	0.93274	0.94507	0.95396	0.9586	0.96206	0.96554	0.96752	0.94084
	0.5	0.87241	0.90923	0.93298	0.94599	0.95411	0.96007	0.96434	0.96756	0.97008	0.94186
	1	0.87253	0.90965	0.93298	0.94598	0.95437	0.96013	0.9645	0.96799	0.97046	0.94207
$\delta_{lat\text{-}means\_hex}$	0.001	0.87237	0.90039	0.90941	0.90805	0.90754	0.89947	0.89994	0.88787	0.86891	0.89488
	0.005	0.87064	0.90971	0.92739	0.93623	0.94218	0.94105	0.94036	0.93801	0.93147	0.92634
	0.025	0.87399	0.90777	0.93131	0.94329	0.95006	0.9547	0.95622	0.95963	0.95902	0.93733
	0.1	0.87251	0.90952	0.93274	0.94507	0.95396	0.9586	0.96205	0.96554	0.96754	0.94084
	0.5	0.87241	0.90923	0.93298	0.94599	0.95411	0.96007	0.96434	0.96756	0.97008	0.94186
	1	0.87253	0.90965	0.93298	0.94598	0.95437	0.96013	0.9645	0.96799	0.97046	0.94207
$\delta_{means\_hex\text{-}means}$	0.001	0.99991	0.99599	0.99202	0.99301	0.99092	0.99255	0.99367	0.99476	0.99184	0.99385
	0.005	1	0.99974	0.99605	0.99989	**1.00201**	**1.00158**	**1.00227**	**1.00247**	**1.00577**	**1.00109**
	0.025	1	0.99989	0.99836	0.99783	**1.00138**	**1.00305**	**1.00187**	**1.00117**	**1.00222**	**1.00064**
	0.1	1	0.99998	0.99921	0.99973	**1.00182**	**1.00339**	**1.00299**	**1.00382**	**1.00228**	**1.00147**
	0.5	1	0.99999	0.99961	**1.00046**	**1.00294**	**1.00309**	**1.00386**	**1.00391**	**1.00411**	**1.002**
	1	1	0.99999	0.99987	**1.00024**	**1.00323**	**1.00392**	**1.004**	**1.00418**	**1.00381**	**1.00214**
$\delta_{medoids\_hex\text{-}means}$	0.001	**1.00416**	**1.0049**	**1.00305**	**1.0038**	**1.01347**	**1.01458**	**1.01659**	**1.01659**	**1.06319**	**1.01559**
	0.005	**1.00111**	**1.00157**	**1.00241**	**1.01675**	**1.02197**	**1.03339**	**1.02847**	**1.03206**	**1.06999**	**1.02308**
	0.025	**1.00019**	**1.00028**	**1.00705**	**1.01565**	**1.02543**	**1.03822**	**1.03492**	**1.03568**	**1.06052**	**1.02422**
	0.1	**1.00005**	**1.00011**	**1.00697**	**1.01477**	**1.02057**	**1.02839**	**1.02967**	**1.0374**	**1.04277**	**1.02008**
	0.5	**1.00001**	**1.00003**	**1.00066**	**1.00879**	**1.01444**	**1.01372**	**1.01773**	**1.02104**	**1.02326**	**1.01108**
	1	1	**1.00002**	**1.00067**	**1.00769**	**1.01078**	**1.0121**	**1.01439**	**1.01647**	**1.01733**	**1.00883**
$\delta_{means\_lat\text{-}means}$	0.001	0.99991	0.99599	0.9925	0.99553	0.98932	0.99326	0.991	0.99303	0.99163	0.99357
	0.005	1	0.99974	0.99605	0.99981	**1.00191**	**1.00157**	**1.00295**	**1.00296**	**1.00514**	**1.00112**
	0.025	1	0.99989	0.99836	0.99783	**1.00137**	**1.00301**	**1.00189**	**1.00102**	**1.00213**	**1.00061**
	0.1	1	0.99998	0.99921	0.99973	**1.00182**	**1.0034**	**1.003**	**1.00381**	**1.00226**	**1.00147**
	0.5	1	0.99998	0.99961	**1.00046**	**1.00294**	**1.00309**	**1.00386**	**1.00391**	**1.00411**	**1.002**
	1	1	0.99999	0.99987	**1.00024**	**1.00323**	**1.00392**	**1.004**	**1.00418**	**1.00381**	**1.00214**
$\delta_{medoids\_lat\text{-}means}$	0.001	**1.00416**	**1.00491**	**1.00355**	**1.00632**	**1.01182**	**1.01531**	**1.01389**	**1.01483**	**1.06299**	**1.01531**
	0.005	**1.00111**	**1.00157**	**1.00241**	**1.01668**	**1.02186**	**1.03338**	**1.02916**	**1.03256**	**1.06931**	**1.02312**
	0.025	**1.00019**	**1.00028**	**1.00705**	**1.01565**	**1.02542**	**1.03818**	**1.03495**	**1.03553**	**1.06043**	**1.02419**
	0.1	**1.00005**	**1.00011**	**1.00697**	**1.01477**	**1.02057**	**1.02839**	**1.02968**	**1.03739**	**1.04274**	**1.02008**
	0.5	**1.00001**	**1.00003**	**1.00066**	**1.00879**	**1.01444**	**1.01372**	**1.01773**	**1.02104**	**1.02326**	**1.01108**
	1	1	**1.00002**	**1.00067**	**1.00769**	**1.01078**	**1.0121**	**1.01439**	**1.01647**	**1.01732**	**1.00883**

TABLE II

ACCURACY GAINS (27)–(34)

		M									
	N, million points	2	3	4	5	6	7	8	9	10	Average
ρ_{means_hex}	0.001	0.94424	0.95172	0.94652	0.94344	0.94043	0.93276	0.93223	0.9231	0.91157	0.93622
	0.005	0.94511	0.9594	0.96375	0.96774	0.9696	0.96723	0.9663	0.96353	0.9605	0.96257
	0.025	0.94649	0.95868	0.96825	0.97237	0.97638	0.97868	0.97804	0.9789	0.978	0.97064
	0.1	0.94549	0.95991	0.96929	0.97482	0.97962	0.98185	0.98298	0.98471	0.98464	0.9737
	0.5	0.94553	0.9598	0.96986	0.97594	0.98027	0.98303	0.9851	0.98656	0.98778	0.97487
	1	0.9456	0.96005	0.96995	0.97577	0.98059	0.98337	0.98538	0.98706	0.98792	0.97508
$\rho_{medoids\_hex}$	0.001	0.94523	0.95311	0.94765	0.9402	0.93657	0.92516	0.91886	0.90556	0.90505	0.93082
	0.005	0.94534	0.9598	0.96584	0.97399	0.97591	0.97833	0.97406	0.97171	0.98132	0.96959
	0.025	0.94656	0.95879	0.97159	0.97912	0.98554	0.99256	0.9912	0.99228	**1.00084**	0.97983
	0.1	0.9455	0.95997	0.97233	0.98038	0.98684	0.99164	0.99331	0.99796	**1.00049**	0.98094
	0.5	0.94554	0.95983	0.97028	0.97906	0.98453	0.98698	0.99042	0.993	0.99511	0.9783
	1	0.94561	0.96007	0.97027	0.97853	0.98341	0.98643	0.98924	0.99162	0.99302	0.97758
$\rho_{hex\text{-}means\_hex}$	0.001	0.94434	0.95467	0.95503	0.95366	0.95075	0.94681	0.9459	0.9391	0.9287	0.94655
	0.005	0.94513	0.95969	0.96648	0.96986	0.97196	0.97064	0.97021	0.96783	0.96323	0.965
	0.025	0.9465	0.95883	0.96932	0.9739	0.97676	0.97893	0.97902	0.98069	0.97959	0.9715
	0.1	0.94549	0.95994	0.96981	0.97518	0.97919	0.9812	0.98256	0.98406	0.98489	0.97359
	0.5	0.94553	0.95981	0.97007	0.9758	0.97935	0.98209	0.98398	0.98544	0.98652	0.97429
	1	0.9456	0.96005	0.97003	0.97577	0.97953	0.98208	0.98407	0.98567	0.98672	0.97439
$\rho_{lat\text{-}means\_hex}$	0.001	0.94434	0.95465	0.95468	0.95221	0.95169	0.94645	0.94734	0.94	0.92883	0.94669
	0.005	0.94513	0.95969	0.96648	0.96986	0.9719	0.97064	0.96982	0.96763	0.96362	0.96498
	0.025	0.9465	0.95883	0.96932	0.9739	0.97677	0.97895	0.97901	0.98075	0.97966	0.97152
	0.1	0.94549	0.95994	0.96981	0.97518	0.97919	0.9812	0.98255	0.98406	0.98491	0.97359
	0.5	0.94553	0.95981	0.97007	0.9758	0.97935	0.98209	0.98398	0.98544	0.98652	0.97429
	1	0.9456	0.96005	0.97003	0.97577	0.97953	0.98208	0.98407	0.98567	0.98672	0.97439
$\rho_{means\_hex\text{-}means}$	0.001	0.9999	0.99692	0.99112	0.98933	0.98914	0.98524	0.98552	0.98298	0.98161	0.98909
	0.005	0.99998	0.99969	0.99718	0.99782	0.99757	0.9965	0.99597	0.99556	0.99716	0.99749
	0.025	1	0.99984	0.99889	0.99843	0.9996	0.99974	0.999	0.99817	0.99838	0.99912
	0.1	1	0.99997	0.99946	0.99964	**1.00044**	**1.00066**	**1.00042**	**1.00067**	0.99974	**1.00011**
	0.5	1	0.99999	0.99979	**1.00013**	**1.00094**	**1.00096**	**1.00114**	**1.00113**	**1.00128**	**1.0006**
	1	1	0.99999	0.99991	1	**1.00109**	**1.00132**	**1.00133**	**1.00141**	**1.00122**	**1.0007**
$\rho_{medoids\_hex\text{-}means}$	0.001	**1.00094**	0.99836	0.9923	0.9859	0.9851	0.97719	0.97137	0.9643	0.97458	0.98334
	0.005	**1.00022**	**1.00011**	0.99934	**1.00426**	**1.00407**	**1.00795**	**1.00396**	**1.00401**	**1.01878**	**1.00475**
	0.025	**1.00007**	0.99995	**1.00234**	**1.00536**	**1.00898**	**1.01392**	**1.01245**	**1.01182**	**1.02169**	**1.00851**
	0.1	**1.00001**	**1.00003**	**1.0026**	**1.00534**	**1.00782**	**1.01064**	**1.01094**	**1.01413**	**1.01583**	**1.00748**
	0.5	1	**1.00001**	**1.00022**	**1.00333**	**1.00529**	**1.00498**	**1.00654**	**1.00767**	**1.00871**	**1.00408**
	1	1	**1.00001**	**1.00024**	**1.00283**	**1.00396**	**1.00443**	**1.00526**	**1.00603**	**1.00638**	**1.00324**
$\rho_{means\_lat\text{-}means}$	0.001	0.9999	0.99694	0.99149	0.99085	0.98816	0.98566	0.98403	0.98203	0.98146	0.98895
	0.005	0.99998	0.99969	0.99718	0.99781	0.99764	0.9965	0.99637	0.99577	0.99676	0.99752
	0.025	1	0.99984	0.9989	0.99842	0.9996	0.99973	0.99901	0.99811	0.9983	0.9991
	0.1	1	0.99997	0.99946	0.99964	**1.00044**	**1.00067**	**1.00043**	**1.00066**	0.99972	**1.00011**
	0.5	1	0.99999	0.99979	**1.00013**	**1.00094**	**1.00096**	**1.00113**	**1.00113**	**1.00128**	**1.0006**
	1	1	0.99999	0.99991	1	**1.00109**	**1.00132**	**1.00133**	**1.00141**	**1.00121**	**1.0007**
$\rho_{medoids\_lat\text{-}means}$	0.001	**1.00094**	0.99839	0.99267	0.98741	0.98413	0.9776	0.96991	0.96337	0.97444	0.98321
	0.005	**1.00022**	**1.00011**	0.99934	**1.00425**	**1.00413**	**1.00795**	**1.00437**	**1.00422**	**1.01837**	**1.00477**
	0.025	**1.00007**	0.99995	**1.00234**	**1.00536**	**1.00898**	**1.01391**	**1.01246**	**1.01176**	**1.02162**	**1.00849**
	0.1	**1.00001**	**1.00003**	**1.0026**	**1.00534**	**1.00782**	**1.01064**	**1.01095**	**1.01412**	**1.01582**	**1.00748**
	0.5	1	**1.00001**	**1.00022**	**1.00333**	**1.00529**	**1.00498**	**1.00654**	**1.00767**	**1.00871**	**1.00408**
	1	1	**1.00001**	**1.00024**	**1.00283**	**1.00396**	**1.00443**	**1.00526**	**1.00603**	**1.00638**	**1.00324**

TABLE III

SPEEDUP GAINS (35)–(42)

		M									
	N, million points	2	3	4	5	6	7	8	9	10	Average
τ_{means_hex}	0.001	1.89969	4.51871	3.8618	3.14053	2.53084	2.60107	2.32278	2.09121	1.74674	2.74593
	0.005	3.07082	5.99538	8.06446	8.47069	7.94407	7.4883	7.05027	6.30224	6.11294	6.72213
	0.025	2.83038	10.73231	17.09562	22.7483	24.39278	25.33783	26.55466	25.6101	25.02445	20.03627
	0.1	3.39836	15.17621	22.39912	275.0207	33.60841	38.14042	42.23587	44.42325	45.36001	57.75137
	0.5	4.48159	19.628	25.38836	31.7869	35.89336	40.18278	45.0587	47.32611	49.02283	33.19652
	1	4.65057	20.39352	26.28289	31.62051	36.86083	39.6733	43.1334	47.45195	50.11196	33.35321
$\tau_{medoids\_hex}$	0.001	6.79874	24.01548	28.88107	25.17684	21.62531	17.74281	18.57348	18.30023	10.22864	19.03807
	0.005	21.3437	45.25801	46.32436	41.37876	32.49575	25.31083	29.04275	31.42211	20.56455	32.5712
	0.025	30.15302	66.43556	63.5175	61.66577	58.57642	48.01696	55.46062	61.73672	38.83504	53.82196
	0.1	43.39594	88.99055	79.79207	77.18132	75.142	66.64599	76.65757	56.6734	52.05424	68.50368
	0.5	112.2538	195.0321	167.1596	141.9144	124.525	119.8729	104.5019	88.00489	86.48418	126.6387
	1	194.2616	334.2102	306.3857	264.2169	238.4299	208.6688	167.8638	159.8016	164.5271	226.4851
$\tau_{hex\text{-}means\_hex}$	0.001	2.53426	3.7887	3.46011	2.81387	2.36629	2.30662	2.03392	1.87765	1.69367	2.54168
	0.005	3.46014	5.5992	5.46725	5.25466	5.02206	5.37465	4.86463	4.37067	4.81849	4.91464
	0.025	3.46085	10.01829	11.39397	12.51954	13.16258	14.1084	15.52732	14.21198	16.29125	12.29935
	0.1	3.82899	15.65819	14.22204	16.29222	18.86183	20.70084	27.09107	27.85902	28.44096	19.21724
	0.5	5.01383	20.91567	19.12489	21.94877	30.48905	32.5675	35.84348	37.80291	40.1673	27.09704
	1	5.2249	21.39962	20.59938	23.7617	35.94803	34.93683	39.20157	57.66443	45.04419	31.53118
$\tau_{lat\text{-}means\_hex}$	0.001	2.57571	4.06739	3.46465	2.90213	2.38556	2.33816	2.11076	1.94047	1.83962	2.62494
	0.005	3.47949	5.57614	5.51818	5.15354	5.12227	5.55867	5.25433	4.66409	5.06068	5.04304
	0.025	3.54948	10.13742	11.65303	12.89796	13.71983	14.19443	16.24564	14.5464	17.14895	12.67702
	0.1	3.98975	15.79031	14.43418	16.473	19.66281	21.25765	27.71476	28.00613	29.02658	19.59502
	0.5	5.17917	20.98249	19.26278	22.73203	30.6513	33.13446	36.18929	37.76809	40.49365	27.37703
	1	5.39525	21.43099	20.76754	24.06075	36.16675	35.14111	39.35985	44.07382	45.29909	30.18835
$\tau_{means\_hex\text{-}means}$	0.001	0.72309	1.21397	1.14368	1.11789	1.06955	1.12478	1.14435	1.11224	0.99564	1.07169
	0.005	0.88729	1.148	1.52381	1.6568	1.63136	1.44317	1.46536	1.45097	1.28426	1.38789
	0.025	0.8186	1.14556	1.58974	1.92351	1.91008	1.86975	1.79397	1.85347	1.57862	1.60926
	0.1	0.89038	1.0259	1.6231	11.61969	1.86131	1.88568	1.63353	1.62196	1.63377	2.64392
	0.5	0.89713	0.93995	1.35962	1.47975	1.19359	1.24606	1.28281	1.26519	1.23307	1.2108
	1	0.89112	0.9531	1.30007	1.34984	1.02715	1.14121	1.10468	1.04286	1.11518	1.1028
$\tau_{medoids\_hex\text{-}means}$	0.001	2.60284	6.37981	8.46326	8.93119	9.08095	7.69491	9.14442	9.74432	5.841	7.54252
	0.005	6.21533	8.58658	8.79328	8.12356	6.65488	4.85986	6.02661	7.22491	4.33098	6.75733
	0.025	8.76251	7.1972	5.89564	5.23667	4.58038	3.54484	3.77408	4.46051	2.44956	5.10015
	0.1	11.40316	6.01063	5.78297	4.88757	4.15655	3.29363	2.96486	2.06773	1.87568	4.71586
	0.5	22.4827	9.34815	8.94631	6.60174	4.14298	3.72474	2.97167	2.35049	2.17492	6.97152
	1	37.5249	15.62051	15.10354	11.30547	6.64085	5.99947	4.30447	3.50984	3.58935	11.51093
$\tau_{means\_lat\text{-}means}$	0.001	0.70982	1.14945	1.13252	1.08614	1.06121	1.11615	1.10475	1.07716	0.94256	1.0422
	0.005	0.88286	1.15964	1.51106	1.67444	1.58983	1.39739	1.36503	1.35423	1.22488	1.35104
	0.025	0.80012	1.13085	1.54822	1.85723	1.84822	1.85991	1.69553	1.80807	1.48195	1.5589
	0.1	0.85645	1.01352	1.5967	11.62528	1.78395	1.8372	1.59534	1.61189	1.59773	2.61312
	0.5	0.86812	0.93659	1.35226	1.4302	1.18631	1.22319	1.26792	1.26721	1.22365	1.19505
	1	0.8628	0.95168	1.29029	1.33238	1.02116	1.1355	1.10259	1.07984	1.10868	1.09832
$\tau_{medoids\_lat\text{-}means}$	0.001	2.55541	6.04582	8.35436	8.62979	8.99902	7.54891	8.77657	9.37748	5.5099	7.31081
	0.005	6.14294	8.65327	8.68726	8.18229	6.47111	4.70817	5.6051	6.74438	4.12342	6.59088
	0.025	8.54166	7.10218	5.73942	5.05257	4.42638	3.53021	3.56785	4.34768	2.30023	4.95646
	0.1	10.95723	5.93942	5.68942	4.81713	3.98601	3.2075	2.89574	2.05418	1.83524	4.59799
	0.5	21.75165	9.31317	8.8971	6.38284	4.1192	3.65536	2.93805	2.3543	2.15846	6.84113
	1	36.32102	15.59706	15.0017	11.15559	6.60284	5.96848	4.29965	3.63631	3.56574	11.34982

There is another important inference from Tables I and II. As the number of lattice cells is increased, both hexagon-based *k*-means and lattice-based *k*-means increase their accuracy gains. This quasi-regularity is absent for the speedup gain. Table III shows that the top speedup gain is achieved at about 5 to 7 lattice cells along an axis.

The instance with the almost 7 % accuracy gain achieved at 5000 points by the 10×10 lattice for hexagon-based *k*-means versus *k*-medoids corresponds to an accuracy gain of 1.878 % in Table II. Here, the hexagon-based *k*-means approach has performed about 4.33 times faster than *k*-medoids. However, from comparing accuracy gains in both Tables I and II it is inferred that *k*-medoids on average has performed worse than *k*-means. On average

$$\delta_{\text{means\_hex-means}} < \delta_{\text{medoids\_hex-means}}, \quad (43)$$

$$\delta_{\text{means\_lat-means}} < \delta_{\text{medoids\_lat-means}}, \quad (44)$$

$$\rho_{\text{means\_hex-means}} < \rho_{\text{medoids\_hex-means}}, \quad (45)$$

$$\rho_{\text{means\_lat-means}} < \rho_{\text{medoids\_lat-means}}, \quad (46)$$

although inequalities (45) and (46) do not hold at 1000 points, while inequalities (43) and (44) turn into equalities for the case of a million points by using the 2×2 lattice (the respective accuracy gain is just 1, i. e., a couple of compared methods perform with the same accuracy). Therefore, it is correct to compare both hexagon-based *k*-means and lattice-based *k*-means to only *k*-means.

Tables I and II also show that hexagon-based *k*-means and lattice-based *k*-means perform at almost the same accuracy rate. Besides, according to Table III, hexagon-based *k*-means is by 0.41 to 3.23 % faster than lattice-based *k*-means (it is 1.85 % for the example in Fig. 3). Therefore, it is reasonable to make further conclusions on just the hexagon-based *k*-means approach.

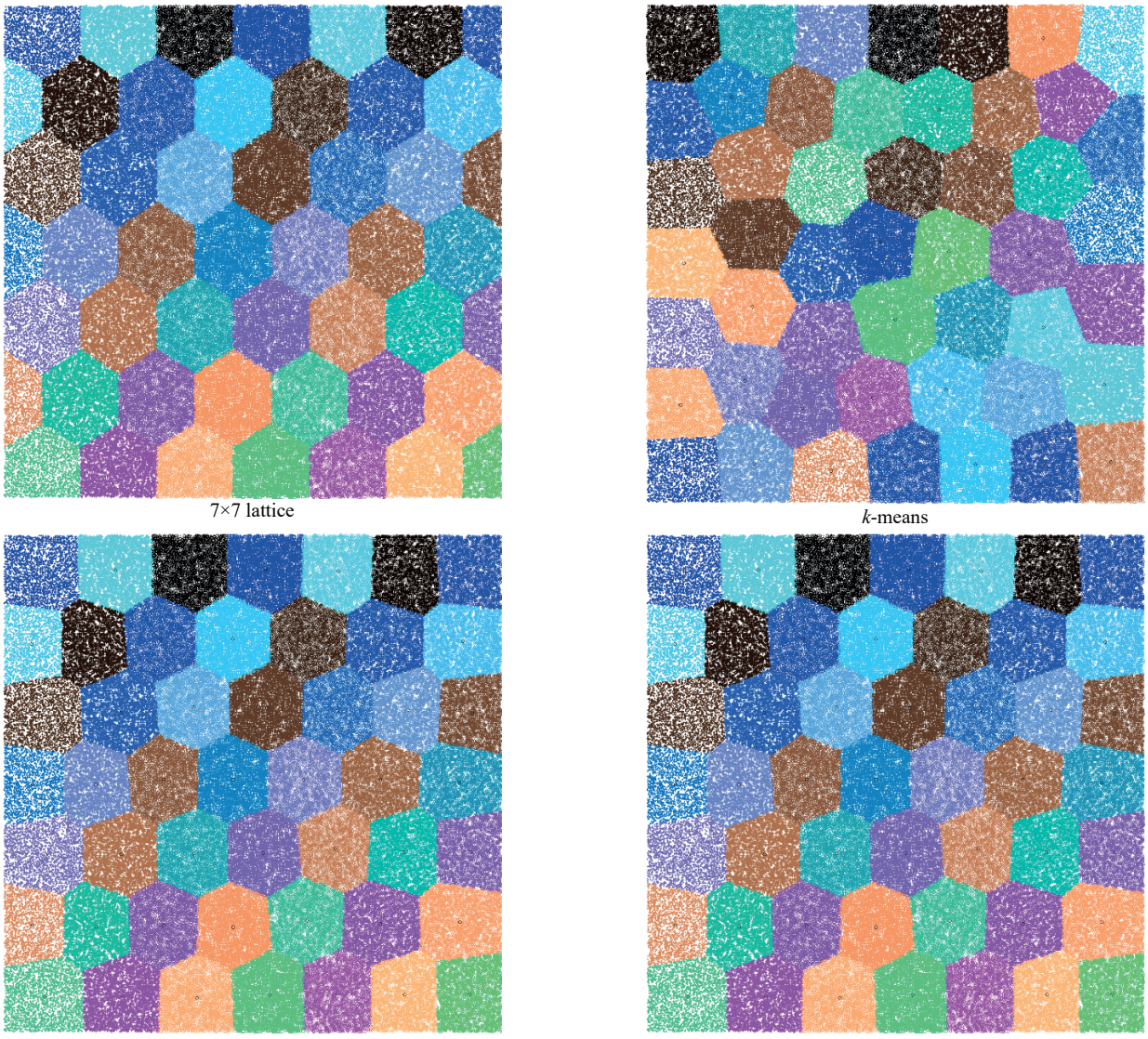

Fig. 3. The four results of partitioning a dense cloud of 125 thousand points into 49 clusters. While applying the hexagonal cell 7×7 lattice singly is at least 30 times faster than *k*-means, it is 3.72 % less accurate by the squared-Euclidean-distance metric and 1.72 % less accurate by the regular-Euclidean-distance metric. The hexagon-based *k*-means approach is respectively 0.41 % and 0.14 % more accurate by these metrics than *k*-means. Besides, it is 1.85 % faster than lattice-based *k*-means (the visual difference is hardly noticeable) and is slightly more accurate than the latter (the difference in their gains is of order of 10^{-7} to 10^{-6}.

VII. Discussion

A hexagonal cell lattice singly, without further *k*-means clustering, is 1.4 to 9 % less accurate (by the regular-Euclidean-distance metric; see Table II); it is an extremely fast method to determine an upper bound of the clustering quality gap. The lattice speedup grows as the number of lattice cells along an axis is increased (see the upper two grey stripes in Table III). It is 42 to 50 times versus the *k*-means approach for 0.1 million points and greater by no fewer than 8 lattice cells along an axis (see the top grey stripe in Table III).

Furthermore, if 0.1 million points and greater are partitioned into at least a few tens of clusters, the hexagonal-pattern-based clustering is just 1.4 to 1.8 % less accurate (in regular Euclidean distances) than the hexagon-based *k*-means approach (see Table II). Nevertheless, the hexagonal-pattern-based clustering is at least 20 times faster for such instances.

Why is hexagon-based *k*-means by 0.41 to 3.23 % faster than lattice-based *k*-means (on average, it is 2.2 % faster)? A plausible explanation is because the centroids by hexagon-based *k*-means are located better than the cell centres by lattice-based *k*-means, so the latter *k*-means takes more steps to converge. Why does the hexagonal pattern improve clustering, after all? Generally speaking, the hexagonal structure is optimal for a planar square region. Therefore, as the density of points increases, the best clustering result tends to have a hexagonal structure, and thus applying the respective hexagonal cell lattice speeds up the convergence.

It is quite obvious that the suggested approach is easily applicable to practical problems dealing with clustering. The hexagonal cell lattice is quickly formed as a two-dimensional array. Creating set (6) as a cluster that is a hexagon enclosing points is a standard computational procedure. While we do not overestimate the significance of our approach, it is a specific supplement to the theory and practice of centroid-based clustering that makes a small yet reasonable and valuable step forward.

VIII. Conclusion

We have suggested a hexagonal-pattern-based approach to partition flat nodes into clusters quicker than the *k*-means algorithm and its modifications do. Our method consists of two steps. First, we apply a hexagonal cell lattice to nodes to approximately determine centroids of the clusters. Second, the centroids are used as initial centroids to start the *k*-means algorithm. Based on the simulation results, we have ascertained that the suggested method is efficient for centroid-based clustering of dense nearly-square point clouds of 0.1 million points and greater by using no fewer than 6 lattice cells along an axis. Compared to *k*-means, our method is at least 10 % faster and it is about 0.01 to 0.07 % more accurate in regular Euclidean distances. In squared Euclidean distances, the accuracy gain is 0.14 to 0.21 %.

In addition, an upper bound of the clustering quality gap is determined the fastest by directly applying a hexagonal cell lattice of the respective size. The upper bound estimation is important to efficiently proceed with a more accurate method. Besides, when the number of clusters is to be optimized, the upper bound can used as a variable of the optimization. We expect that the optimal number of clusters would correspond to the minimum of the upper bound.

Obviously, distinct limitations exist here. First, the lattice size must be compatible with the number of clusters. Second, the cloud of points must not much vary in density and be of nearly-square shape. The first limitation can be almost removed by applying a bigger-sized lattice to cover the point cloud so that the required number of lattice cells would be formed. Some empty lattice cells at boundaries of the cloud do not matter and are ignored. The second limitation, however, is not removable.

Our contribution comes into the optimized centroid-based clustering of flat objects, which is practically important to efficiently solve the respective metric facility location problem. We believe it is possible that the hexagonal-pattern-based approach can be successfully applied to clustering objects with a greater number of features. The efficiency for such cases is yet to be studied separately.

References

[1] V. Srivastava and B. Biswas, “An optimization based framework for region wise optimal clusters in MR images using hybrid objective,” *Neurocomputing*, vol. 541, Jul. 2023, Art. no. 126286. https://doi.org/10.1016/j.neucom.2023.126286

[2] M. Woźniak and D. Połap, “Object detection and recognition via clustered features,” *Neurocomputing*, vol. 320, pp. 76–84, Dec. 2018. https://doi.org/10.1016/j.neucom.2018.09.003

[3] N. Dong, B. Ren, H. Li, X. Zhong, X. Gong, J. Han, J. Lv, and J. Cheng, “A novel anomaly score based on kernel density fluctuation factor for improving the local and clustered anomalies detection of isolation forests,” *Information Sciences*, vol. 637, Aug. 2023, Art. no. 118979. https://doi.org/10.1016/j.ins.2023.118979

[4] M. Nicholson, R. Agrahari, C. Conran, H. Assem, and J. D. Kelleher, “The interaction of normalisation and clustering in sub-domain definition for multi-source transfer learning based time series anomaly detection,” *Knowledge-Based Systems*, vol. 257, Dec. 2022, Art. no. 109894. https://doi.org/10.1016/j.knosys.2022.109894

[5] S. C. Basak, V. R. Magnuson, G. J. Niemi, and R. R. Regal, “Determining structural similarity of chemicals using graph-theoretic indices,” *Discrete Applied Mathematics*, vol. 19, no. 1–3, pp. 17–44, Mar. 1988. https://doi.org/10.1016/0166-218X(88)90004-2

[6] K. Schatz, F. Frieß, M. Schäfer, P. C. F. Buchholz, J. Pleiss, T. Ertl, and M. Krone, “Analyzing the similarity of protein domains by clustering Molecular Surface Maps,” *Computers & Graphics*, vol. 99, pp. 114–127, Oct. 2021. https://doi.org/10.1016/j.cag.2021.06.007

[7] K. Mohammadpour, A. Rashki, M. Sciortino, D. G. Kaskaoutis, and A. D. Boloorani, “A statistical approach for identification of dust-AOD hotspots climatology and clustering of dust regimes over Southwest Asia and the Arabian Sea,” Atmospheric Pollution Research, vol. 13, no. 4, Apr. 2022, Art. no. 101395. https://doi.org/10.1016/j.apr.2022.101395

[8] M. Balcilar, A. H. Elsayed, and S. Hammoudeh, “Financial connectedness and risk transmission among MENA countries: Evidence from connectedness network and clustering analysis,” *Journal of International Financial Markets, Institutions and Money*, vol. 82, Jan. 2023, Art. no. 101656. https://doi.org/10.1016/j.intfin.2022.101656

[9] A. M. Dichiarante, N. Langet, R. A. Bauer, B. P. Goertz-Allmann, S. C. Williams-Stroud, D. Kühn, V. Oye, S. E. Greenberg, and B. D. E. Dando, “Identifying geological structures through microseismic cluster and burst analyses complementing active seismic interpretation,” *Tectonophysics*, vol. 820, Dec. 2021, Art. no. 229107. https://doi.org/10.1016/j.tecto.2021.229107

[10] V. V. Romanuke, “Fast-and-smoother uplink power control algorithm based on distance ratios for wireless data transfer systems,” *Studies in Informatics and Control*, vol. 28, no. 2, pp. 147–156, 2019. https://doi.org/10.24846/v28i2y201903

[11] V. V. Romanuke, "An uplink power control routine for quality-of-service equalization in wireless data transfer networks constrained to equidistant power levels," *KPI Science News*, no. 2, pp. 46–56, 2019. https://doi.org/10.20535/kpi-sn.2019.2.160199

[12] Z. Zhang, Q. Feng, J. Huang, and J. Wang, "Improved approximation algorithms for solving the squared metric *k*-facility location problem," *Theoretical Computer Science*, vol. 942, pp. 107–122, Jan. 2023. https://doi.org/10.1016/j.tcs.2022.11.027

[13] S. Li, "A 1.488 approximation algorithm for the uncapacitated facility location problem," in *Automata, Languages and Programming. Lecture Notes in Computer Science*, L. Aceto, M. Henzinger, and J. Sgall, Eds., vol. 6756. Springer, Berlin, Heidelberg, 2011, pp. 77–88. https://doi.org/10.1007/978-3-642-22012-8_5

[14] A. M. Ikotun, A. E. Ezugwu, L. Abualigah, B. Abuhaija, and J. Heming, "*K*-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data," *Information Sciences*, vol. 622, pp. 178–210, Apr. 2023. https://doi.org/10.1016/j.ins.2022.11.139

[15] M. E. Celebi, H. A. Kingravi, and P. A. Vela, "A comparative study of efficient initialization methods for the *k*-means clustering algorithm," *Expert Systems with Applications*, vol. 40, no. 1, pp. 200–210, Jan. 2013. https://doi.org/10.1016/j.eswa.2012.07.021

[16] M. Mahajan, P. Nimbhorkar, and K. Varadarajan, "The planar *k*-means problem is NP-hard," in *WALCOM: Algorithms and Computation. Lecture Notes in Computer Science*, S. Das and R. Uehara, Eds., vol. 5431. Springer, Berlin, Heidelberg, 2009, pp. 274–285. https://doi.org/10.1007/978-3-642-00202-1_24

[17] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu, "A local search approximation algorithm for *k*-means clustering," *Computational Geometry: Theory and Applications*, vol. 28, no. 2–3, pp. 89–112, Jun. 2004. https://doi.org/10.1016/j.comgeo.2004.03.003

[18] P. Fränti and S. Sieranoja, "How much can *k*-means be improved by using better initialization and repeats?" *Pattern Recognition*, vol. 93, pp. 95–112, Sep. 2019. https://doi.org/10.1016/j.patcog.2019.04.014

[19] V. V. Romanuke, "Optimization of a dataset for a machine learning task by clustering and selecting closest-to-the-centroid objects," *Herald of Khmelnytskyi National University. Technical Sciences*, vol. 1, no. 6, pp. 263–265, 2018.

[20] R. Ostrovsky, Y. Rabani, L. J. Schulman, and C. Swamy, "The effectiveness of Lloyd-type methods for the *k*-means problem," in *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, Berkeley, CA, USA, Oct. 2006, pp. 165–174. https://doi.org/10.1109/FOCS.2006.75

[21] H. A. Yehoshyna and V. V. Romanuke, "Constraint-based recommender system for commodity realization," *Journal of Communications Software and Systems*, vol. 17, no. 4, pp. 314–320, Oct. 2021. https://doi.org/10.24138/jcomss-2021-0102

[22] A. Vattani, "*k*-means requires exponentially many iterations even in the plane," *Discrete and Computational Geometry*, vol. 45, no. 4, pp. 596–616, Mar. 2011. https://doi.org/10.1007/s00454-011-9340-1

[23] A. Chakrabarty and D. Swagatam, "On strong consistency of kernel *k*-means: A Rademacher complexity approach," *Statistics & Probability Letters*, vol. 182, Mar. 2022, Art. no. 109291. https://doi.org/10.1016/j.spl.2021.109291

[24] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A *k*-means clustering algorithm," *Journal of the Royal Statistical Society, Series C*, vol. 28, no. 1, pp. 100–108, 1979. https://doi.org/10.2307/2346830

[25] J. Cartensen, "About hexagons," *Mathematical Spectrum*, vol. 33, no. 2, pp. 37–40, 2000–2001.

[26] R. Fletcher, *Practical Methods of Optimization* (2nd ed.). J. Wiley and Sons, Chichester, 1987.

[27] S. A. Vavasis, "Complexity issues in global optimization: A survey," in *Handbook of Global Optimization. Nonconvex Optimization and Its Applications,* R. Horst and P. M. Pardalos, Eds., vol. 2. Springer, Boston, MA, 1995, pp. 27–41. https://doi.org/10.1007/978-1-4615-2025-2_2

Vadim V. Romanuke was born in 1979. He graduated from the Technological University of Podillya in 2001. In 2006, he received the Degree of Candidate of Technical Sciences in Mathematical Modelling and Computational Methods. The degree of Doctor of Technical Sciences in Mathematical Modelling and Computational Methods was received in 2014. In 2016, Vadim Romanuke received the academic status of Full Professor. His current research interests concern wireless communication systems, job scheduling, semantic image segmentation, decision making, game theory, and optimization.
Address for correspondence: Soborna Str. 87, Vinnytsia, Ukraine, 21050.
E-mail: v.romanyuk@vtei.edu.ua
ORCID iD: https://orcid.org/0000-0001-9638-9572

Svitlana V. Merinova was born in 1969. She graduated from the Vinnytsia Polytechnic Institute in 1991. In 1994, she graduated from the Kyiv Institute of Trade and Economics. In 2011, she received the Degree of Candidate of Economical Sciences in Economics and Enterprise Management. The academic status of Associate Professor was received in 2016. Her current research interests encircle business network communication, enterprise digitalization, cluster analysis of 2D and 3D models for systems of computer control and design.
Address for correspondence: Soborna Str. 87, Vinnytsia, Ukraine, 21050.
E-mail: s.merinova@vtei.edu.ua
ORCID iD: https://orcid.org/0000-0001-6563-5320

Hanna A. Yehoshyna was born in 1982. The Master degree was received in 2005 after graduating from the Donetsk State Institute of Artificial Intelligence, Ukraine. She received the Degree of Candidate of Technical Science in Artificial Intelligence Systems and Tools in 2009. The academic status of Associate Professor was received in 2011. Her current research interests focus on machine learning, deep learning, knowledge-based systems, computational linguistics and recommender systems.
Address for correspondence: 10726 106 Ave., Grande Prairie, AB T8V 4C4, Canada.
E-mail: hyehoshyna@nwpolytech.ca
ORCID iD: https://orcid.org/0000-0002-2381-1231