Pattern Recognition 112 (2021) 107713

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

A simulated annealing algorithm with a dual perturbation method for clustering

Julian Lee[a,*], David Perkins[b]

[a] The Pingry School, 131 Martinsville Road, Basking Ridge, NJ 07920, United States
[b] Taylor Science Building, Hamilton College, 198 College Hill Road, Clinton, NY 13323, United States

ARTICLE INFO

Article history:
Received 2 February 2020
Revised 11 October 2020
Accepted 21 October 2020
Available online 22 October 2020

Keywords:
Partitional clustering
Simulated annealing
Sum of squared error criterion
K-means

ABSTRACT

Clustering is a powerful tool in exploratory data analysis that partitions a set of objects into clusters with the goal of maximizing the similarity of objects within each cluster. Due to the tendency of clustering algorithms to find suboptimal partitions of data, the approximation method Simulated Annealing (SA) has been used to search for near-optimal partitions. However, existing SA-based partitional clustering algorithms still settle to local optima. We propose a new SA-based clustering algorithm, the Simulated Annealing with Gaussian Mutation and Distortion Equalization algorithm (SAGMDE), which uses two perturbation methods to allow for both large and small perturbations in solutions. Our experiments on a diverse collection of data sets show that SAGMDE performs more consistently and yields better results than existing SA clustering algorithms in terms of cluster quality while maintaining a reasonable runtime. Finally, we use generative art as a visualization tool to compare various partitional clustering algorithms.

1. Introduction

The ever-growing abundance of data prompts further research into data mining. An important and well-studied technique within the field of data mining is clustering, a type of unsupervised machine learning that involves grouping data based on predefined attributes. Clustering is applicable to a vast array of fields ranging from statistics to engineering to psychology [1]. For example, this tool has applications in medical fields through image segmentation [2] as well as business through market analysis [3].

Clustering is an NP-hard problem [4] that is generally solved by assigning *N*data points in *D*dimensions to *K*clusters with the goal of optimizing these clusters based on a given criterion. There are two main categories of clustering: hierarchical and partitional [5]. In hierarchical clustering, objects in the data set are grouped into a nested hierarchy of clusters. Partitional clustering involves dividing the objects into groups simultaneously without creating a nested structure.

Researchers have also investigated a third category of clustering, density-based clustering, which builds clusters based on the density of data points in a given region of the *D*-dimensional space. This class of algorithms has the benefit of reduced sensitivity to noise and cluster shape, but this strategy generally does not perform well for high-dimensional data [6]. Examples of algorithms that fully or partially employ a density-based strategy include a critical distance-based algorithm [7] and a gravity-center algorithm [8].

This paper will focus on partitional clustering algorithms. The most popular approach to partitional clustering is center-based clustering. Center-based, iterative clustering algorithms follow these two general steps:

1. Centers are initialized
2. Until convergence, every point's membership to each center and every point's weight are recomputed and used to recalculate the centers

The *K*-means algorithm (KM) is the best-known center-based clustering algorithm. It finds *K*centers in *D*dimensions with the goal of maximizing the similarity of the elements within each cluster based upon the provided attributes of each object. To accomplish this goal, the algorithm attempts to minimize the sum of the distances between each point and its closest center, which is the Sum of Squares Error (SSE). Given that $X = \{x_i | i = 1, \ldots, N\}$represents the *N*data points and $M = \{m_l | l = 1, \ldots, K\}$represents the current *K*centers, the performance function for *K*-means is shown below:

$$Perf_{KM}(X, M) = \sum_{i=1}^{N} MIN\{\|x_i - m_l\|^2 |\ l = 1, \ldots, K\} \quad (1)$$

* Corresponding author.
E-mail addresses: jlee2021@pingry.org (J. Lee), dperkins@hamilton.edu (D. Perkins).

https://doi.org/10.1016/j.patcog.2020.107713
0031-3203/

The distance-based approach of KM requires quantitative data, but researchers have produced variants of KM to perform clustering on mixed numerical and categorical data [9]. KM has a hard membership function, meaning that each point belongs to only one cluster (in this case its nearest cluster) as opposed to soft membership, where a proportion of each point belongs to multiple centers [10]. Additionally, every point in KM has the same weight. In KM, only the points close to a given center are considered in the recalculation of that center's position; as a result, KM is sensitive to initialization and converges upon a local minimum.

There are also studies aimed at finding the optimal value of $K$for center-based clustering algorithms such as Pelleg and Moore's study [11]. Guan et al. [12] proposed the K-means+ algorithm that automatically finds a near-optimal K.

Various initialization methods have been investigated to improve the clusters found by KM. Two popular algorithms include Forgy's method [13], in which the initial centers are assigned as Krandom observations in the data set, and MacQueen's method (the Random Partition method) [14], in which objects are randomly assigned to clusters and the initial centers are calculated as the centroids of these clusters. However, in a comparative study of initialization methods for K-means, Celebi et al. [15] showed there are alternative initialization methods that outperform these popular algorithms. Additionally, Fränti and Sieranoja [16] found that data sets with well-separated clusters still pose a challenge for all tested existing initialization methods.

Beyond experimenting with initialization methods, researchers have studied the use of different performance functions to improve cluster quality. The Fuzzy K-Means (FKM) algorithm, proposed by Dunn [17], as well as the K-Harmonic Means algorithm, proposed by Zhang et al. [18], use new performance functions that account for soft membership such that every point influences the location of each center. The K-Harmonic Means algorithm applies varying weights to points to further optimize the output, but KHM still converges upon a local minimum rather than the global minimum [18].

Metaheuristics serve as a different method to solve this local optima problem [19]. Some authors [20,21] used the local search heuristic while some others [22,23] applied the Tabu search method [24], and some others [25,26] used genetic algorithms. Nature-based metaheuristic methods have also been explored [27]. Additionally, competitive learning algorithms [28,29] have been proposed, which use an abundance of initial centers (seeds) that "compete," "cooperate," or both to eventually yield a given number of cluster centers; these algorithms have the benefit of automatically determine the number of cluster centers. This paper will specifically focus on the use of the simulated annealing (SA) algorithm [30].

This paper will describe the simulated annealing heuristic and its current applications to the clustering problem in Section 2. Section 3 describes our proposed algorithm. Experimental results that compare the proposed algorithm with other clustering algorithms using the simulated annealing heuristic are detailed in Section 4. Section 5 covers a comparison of center-based, partitional clustering algorithms through generative art.

2. Simulated annealing with *K*-means

The SA heuristic is modeled after the physical annealing process, in which a solid is slowly cooled until its atoms are repositioned to form a crystal with a low-energy state [31]. Metropolis et al. described the key components of the algorithm in 1953 [30], and the full algorithm was first presented by Kirkpatrick et al. [32], who also proposed the algorithms use as an optimization tool for combinatorial problems that attempt to minimize a cost function. In addition to accepting modified solutions that decrease the cost function, the algorithm sometimes accepts solutions that increase the cost function, which allows the algorithm to eventually converge upon a better local minimum.

The algorithm begins with a random solution. It then generates a neighboring solution by slightly modifying the current solution through the use of a perturbation method (which simulates atoms moving randomly within the solid). The algorithm accepts the new solution over the prior if the quality (energy) of the new solution, as determined by the cost function, is lower. Otherwise, the solution is accepted with a probability of $e^{\Delta E/T}$,where $\Delta E$is the change in energy between the solutions and $T$is the solutions temperature. Temperature acts as a control parameter in the algorithm [31]. Once enough perturbations to the solutions are made at a given temperature to allow the energy to settle to a thermal equilibrium, the temperature is decreased until a minimum temperature is reached. Reaching a thermal equilibrium at each temperature allows for a minimum near the global minimum to be found regardless of the initial solution [5].

When implementing the SA algorithm, there are two categories of considerations: problem-specific choices and generic choices [33]. Problem-specific choices involve implementing the various simulated annealing functions, namely the cost function, perturbation method, and the structure of the solution, based upon the problem of interest. Generic choices, such as setting the parameters for cooling, are not directly related to the specific problem a SA algorithm aims to solve.

2.1. Prior research

In the context of partitional data clustering, simulated annealing is used to minimize a cost function that relates to the homogeneity of objects within each cluster. In the following implementations of SA as well as in our own SAGMDE algorithm, a given solution consists of Kcenter locations, and the cost function is based upon the performance function for K-means, the Sum of Squares Error. However, research has also been conducted regarding other cost functions such as the KHM performance function [10].

Several prior SA algorithms for K-means [1,34] have used a perturbation method whereby the calculation of a neighboring solution involves the movement of a single point chosen at random from its current cluster to a randomly chosen cluster. All clusters and points have an equal probability of being chosen. These proposed algorithms differ in their calculation of the cooling parameters.

This paper will use the parameters detailed by Brown and Huntley [1]. The Brown and Huntley algorithm (BH) calculates the initial and final temperatures after a trial run of $MaxIt$iterations, where $MaxIt$is the number of iterations to be run at every given temperature. During this trial run, two values are calculated: $m^+$and μ^+. The variable m^+represents the number of times the cost of the solution increased during the trial iterations, while the variable μ^+represents the average cost increase in the trial iterations. Then, the initial temperature, T_0,is calculated with a user-defined acceptance ratio, χ,using the formula:

$$T_0 = \frac{\mu^+}{\log\left(\frac{m^+}{\chi\mu^+ - (1-\chi)(MaxIt-\mu^+)}\right)} \tag{2}$$

The final temperature, T_f,is calculated using $\beta$and ϵsuch that the probability of accepting a cost increase of $\beta\mu^+$at the final temperature is ϵ:

$$T_f = \frac{-\beta\mu^+}{\log(\epsilon)} \tag{3}$$

Finally, given the user-defined variable $NumTemp$ that specifies the number of temperatures to be run by the algorithm, α,which

is the decrement variable for the temperature such that $T_{n+1} = \alpha T_n$,can be calculated as follows:

$$\alpha = \left(\frac{T_f}{T_0}\right)^{\frac{1}{NumTemp}} \tag{4}$$

Brown and Huntley used $NumTemp = 200, MaxIt = 4n, \chi = 0.25, \beta = 0.125$,and $\epsilon = 0.00000000001$for the simulations in their paper, where $n$is the number of data points.

Bandyopadhyay et al. created another SA-based clustering algorithm, the SAKM-clustering algorithm [35], which moves multiple points between clusters to produce neighboring solutions. Points farther away from their current cluster center are more likely to be redistributed to another cluster. Additionally, the probability of moving any given point decreases as the overall temperature decreases since large deviations in neighboring solutions are unlikely to be accepted at low temperatures. The probability that a point $i$is transferred from its current cluster $j$to cluster $k$is given by

$$\exp\left(\frac{-\max(D_{ik} - D_{ij}, 0)}{T}\right) \tag{5}$$

where $D_{ik}$is the Euclidean distance between point $i$and center k,and $T$is the current temperature. The values used by Bandyopadhyay et al. are $T_0 = 100$and $\alpha = 0.95$,and the algorithm ends when either the current configuration of clusters has no more possible perturbations (which occurs at small values of T) or when the algorithm has gone through 200 temperatures.

Merendino and Celebi [5] propose the SAGM algorithm. Rather than transferring points between clusters as seen in the other algorithms, the SAGM perturbation method slightly moves each centers position based on a Gaussian distribution. In the perturbation method, $\delta G$is added to each attribute of every center, where $\delta$is an extremely small constant and $G$is a randomly generated Gaussian number. As the algorithm uses the same mutation factor regardless of each attribute's range, the data is normalized using min-max normalization such that all values of each attribute are scaled linearly to the range [0,1]. Then, points are reassigned to the closest cluster. Despite its use of SA, the algorithm is sensitive to initialization as the positions of the centers have a minimal change in neighboring solutions. The values used in their paper for δ, T_0, T_f, α,and $MaxIt$are 0.001, 0.002, 0.00001, 0.95, and $2n$respectively, where $n$is the number of points.

3. Our proposed algorithm: SAGMDE

Our Simulated Annealing with Gaussian Mutation and Distortion Equalization algorithm differs from the previously mentioned algorithms in its combination of a small-perturbation method that allows for small shifts in the cluster centers with a large-perturbation method that allows for bigger, strategic shifts in centers. Specifically, the algorithm uses the Gaussian perturbation method (seen in SAGM) that provides these smaller shifts in conjunction with the distortion equalization and utility procedure introduced by PatanĘ and Russo [36], which generates larger shifts in centers.

The distortion equalization and utility procedure begins by calculating each cluster's *distortion*, which is given by the sum of the distances between a center and all the points within its cluster. The distortion of the ith cluster, S_i,is given by

$$D_i = \sum_{x \in S_i} d(x, y_i) \tag{6}$$

in which $y$is the set of centers, and $d(x, y_i)$is the Euclidean distance between point $x$and center y_i. The *utility*of cluster $i$is calculated as

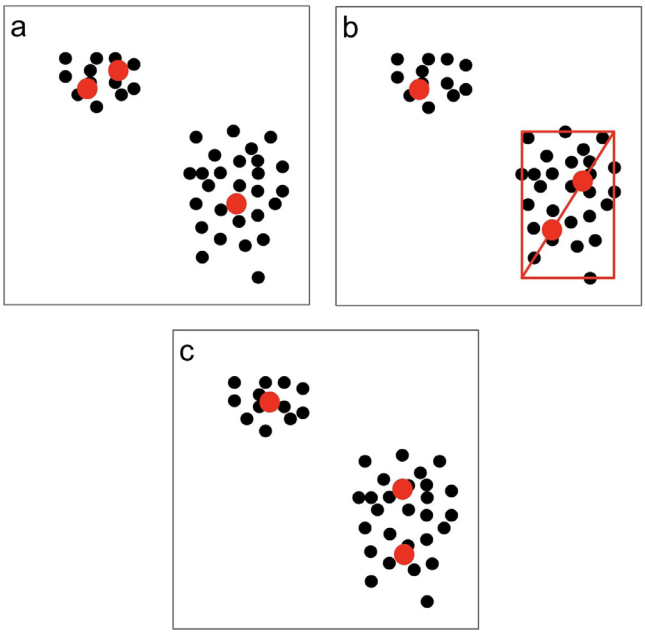

Fig. 1. (Color necessary) A shift of centers using the distortion equalization and utility procedure. **a** The original center positions (shown in red). **b** Shift of two centers to the principal diagonal of the minimum volume hyper-box. **c** Refinement of centers. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

the cluster's distortion divided by the mean distortion of the clusters:

$$U_i = \frac{D_i}{D_{mean}} \tag{7}$$

As all distortions are divided by the same value, the equalization of distortions is equivalent to the equalization of utilities. If all clusters had the same distortion, every cluster's utility would be 1, so a low utility is defined as a utility less than 1 and a high utility is defined as a utility greater than 1. The function then attempts to equalize the utilities by shifting each center whose cluster has a low utility towards a center whose cluster has a high utility. In each shift, the low utility and high utility centers are moved to the principal diagonal of the K-dimensional hyper-box, which is the minimum volume hyper-box that encompasses all the points originally belonging to the high utility center, such that they divide this diagonal into thirds. The locations of these two centers are refined based on the points originally belonging to the high utility center to reach a local minimum for SSE. The points originally belonging to the low utility-shifted cluster are reassigned to the center nearest to the low utility-shifted cluster, and this nearest center is recalculated as the mean of the points in its cluster. The new solution of centers after each shift is evaluated against the prior solution, and the new solution replaces the prior solution if it reduces the mean distortion.

The algorithm helps create a more even distribution of the centers' distortions, which often leads to a better solution of centers. Specifically, the use of the distortion equalization and utility procedure prevents the algorithm from being caught in a local minimum in which points of a smaller cluster are partitioned among multiple centers while points in a larger or more dispersed cluster are partitioned among fewer centers. This can be seen in the example shown in Fig. 1. The small Gaussian perturbations in SAGM would be unable to shift one of the top left centers in Fig. 1a to its ideal position in the bottom right cluster as seen in Fig. 1c.

In the distortion equalization perturbation, a randomly chosen center is replaced by a randomly generated center, and the centers

are then shifted based on the distortion equalization and utility procedure. This distortion equalization perturbation has been used previously in a KHM-based simulated-annealing algorithm [10], but it was the sole perturbation method.

While SAGMDE uses the Gaussian perturbation *MaxIter*times for every temperature, a larger shift from the distortion equalization perturbation is only attempted $\frac{MaxIter}{20}$ times per temperature. Different cooling schedules and starting temperatures are used for the two perturbation methods. The current best solution and its corresponding temperature for the Gaussian perturbation are stored. This allows for a slight optimization of the best solution by a short second cooling that applies SA with the Gaussian perturbation starting at the temperature at which the best solution was found.

The pseudo-code for SAGMDE is provided (Algorithm 1). The

Algorithm 1: Pseudo-code for SAGMDE.

```
Select the maximum temperatures, T and T_distort;
Select the final temperature, T_f;
Select the cooling values, α and α_distort;
Select the number of iterations, MaxIter;
Generate a random trial solution S_0 of centers;
while T > T_f do
    for i ← 0 to MaxIter do
        if C_i < C_best then
            C_best ← C_i;
            S_best ← S_i;
            T_best ← T;
        end
        Generate a trial solution S_t with cost C_t from S_i
        with cost C_i using Gaussian perturbation;
        if C_t ≤ C_i or exp((C_i − C_t)/T) > Rand[0,1] then
            S_i ← S_t;
            C_i ← C_t;
        end
        if i%20 == 0 then
            Generate a trial solution S_t with cost C_t from S_i
            with cost C_i using Distortion Equalization;
            if C_t ≤ C_i or exp((C_i − C_t)/T_distort) > Rand[0,1] then
                S_i ← S_t;
                C_i ← C_t;
            end
        end
    end
    T_distort = T_distort * α_distort;
    T = T * α;
end
T ← T_best;
S_i ← S_best;
while T > T_f do
    for i ← 0 to MaxIter do
        Generate a trial solution S_t from S_i using Gaussian
        perturbation;
        if C_t ≤ C_i or exp((C_i − C_t)/T) > Rand[0,1] then
            S_i ← S_t;
            C_i ← C_t;
        end
    end
    T = T * α;
end
return S_i
```

Table 1
Descriptions of real data sets.

Data set	# Points	# Attributes	# Classes (K)
Breast Cancer Wisconsin (Original)	683	10	2
Ecoli	336	7	8
Glass Identification	214	9	6
Iris	150	4	3
Wine	178	13	3
Yeast	1484	8	10

following abbreviations were used:

T: Initial temperature for the Gaussian perturbation

$T_{distort}$: Initial temperature for the Distortion Equalization perturbation

T_f: Final temperature for the Gaussian perturbation

α: Cooling value for the Gaussian perturbation

$\alpha_{distort}$: Cooling value for the Distortion Equalization perturbation

MaxIter: Number of iterations

For the Gaussian perturbation, we used $\delta = \sqrt{T}/10$to allow larger shifts at higher temperatures. To use SAGMDE without min-max scaling, the Gaussian perturbation method should be scaled for each attribute by adding the value $a\delta G$instead of δG,where $a$is the difference in the maximum and minimum values for a specific attribute across all points.

4. Experimental study

The experimental results below compare our algorithm, SAGMDE, with some of the aforementioned algorithms, namely KM, BH, SAKM, and SAGM. As all of these algorithms are center-based clustering algorithms, they should not be applied to highly nonlinear data sets such as concentric rings. We ran the algorithms 20 times on a diverse set of 9 data sets: 6 of these were taken from the Machine Learning Laboratory [37] (details on the real data sets used can be found in Table 1), and we created 3 synthetic data sets [38–40] that had a greater number of clusters as seen in Fig. 2. The synthetic data sets have 600 points in 2 dimensions, and the offset between a given point and its true center in each dimension is determined by the following expression where $G$is a random Gaussian number:

$$Rand[0.02, 0.04] * G \tag{8}$$

The experiments were conducted using a 2.8 GHz, 16 GB RAM computer. The Random Partition initialization method was used to generate the initial positions of the centers for the runs testing the SA algorithms. The Forgy initialization was used for KM. Parameters for SAGMDE were set as follows: $MaxIter = 2n, T = 0.0015, T_f = 0.000001, \alpha = 0.98, T_{distort} = 6, \alpha_{distort} = 0.985 (T_{distort\_f} = 0.025)$. Note $n$is the number of points in the data set. Cooling parameters for the other SA algorithms are specified in Section 2.1. Prior to each run, all data was normalized based on min-max normalization as described in Section 2.

To measure each algorithm's performance over the 20 runs for each data set, the CPU time and the quality of the centers based on the Sum of Squares Error were recorded for each run. Results are detailed in Tables 2-10, each corresponding to a data set. The minimum, mean, maximum, and standard deviation of performance based on SSE as well as the average CPU time are listed for each algorithm, and best results regarding the quality of the clusters across the algorithms for each data set are shown in bold.

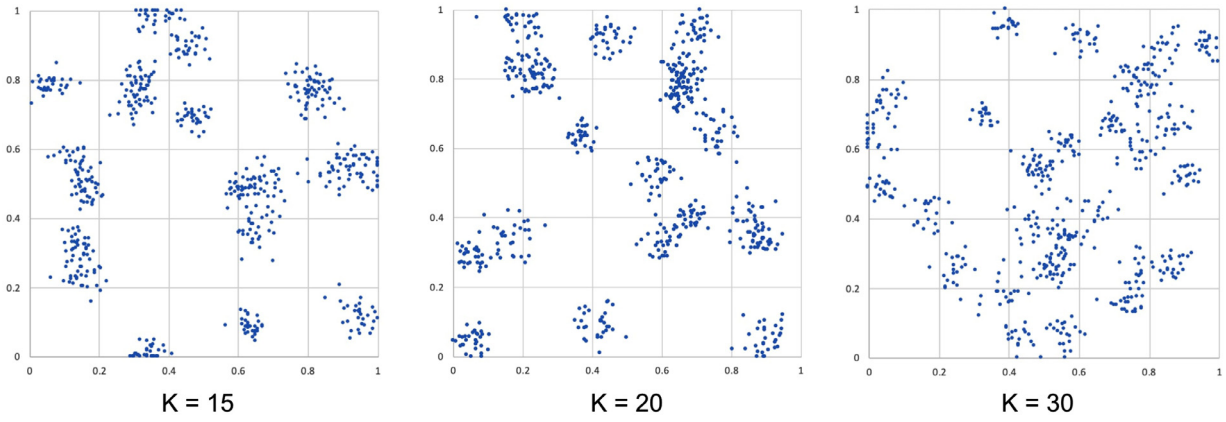

Fig. 2. (Color necessary) Synthetic data sets. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 2
Experiments with breast cancer wisconsin data set.

	KM	BH	SAKM	SAGM	SAGMDE
Min SSE	253.403	253.441	**253.397**	253.398	**253.397**
Mean SSE	253.404	253.476	253.409	253.399	**253.397**
Max SSE	253.404	253.539	253.471	253.399	**253.397**
SD in SSE	0.0004	0.0269	0.0169	0.0004	**0.0000**
Mean Time (ms)	0.65	38,964	13,817	3738	45,421

Table 3
Experiments with Ecoli data set.

	KM	BH	SAKM	SAGM	SAGMDE
Min SSE	17.443	18.409	**17.406**	17.408	**17.406**
Mean SSE	18.048	18.454	17.411	17.657	**17.409**
Max SSE	18.83	18.713	17.439	18.733	**17.437**
SD in SSE	0.4883	0.082	0.0102	0.4679	**0.0084**
Mean Time (ms)	1.15	12,460	5416	1756	29,716

Table 4
Experiments with glass identification data set.

	KM	BH	SAKM	SAGM	SAGMDE
Min SSE	18.592	18.253	**18.241**	18.349	**18.241**
Mean SSE	19.791	19.697	**18.284**	19.859	18.298
Max SSE	24.827	21.971	19.049	22.732	**18.382**
SD in SSE	1.4269	0.8361	0.1801	1.0585	**0.0564**
Mean Time (ms)	0.7	5116	2395	724	9909

Table 5
Experiments with Iris data set.

	KM	BH	SAKM	SAGM	SAGMDE
Min SSE	**6.998**	7.005	**6.998**	**6.998**	**6.998**
Mean SSE	8.314	7.015	6.999	6.999	**6.998**
Max SSE	12.144	7.027	7.003	6.999	**6.998**
SD in SSE	2.1376	0.0061	0.0014	0.0002	**0.0000**
Mean Time (ms)	0.25	764	533	104	1103

Table 6
Experiments with wine data set.

	KM	BH	SAKM	SAGM	SAGMDE
Min SSE	48.961	**48.954**	**48.954**	48.956	**48.954**
Mean SSE	48.983	48.96	48.961	48.966	**48.954**
Max SSE	49.015	48.973	48.992	48.988	**48.954**
SD in SSE	0.014	0.0068	0.0098	0.0148	**0.0000**
Mean Time (ms)	0.45	3090	1465	456	7051

Table 7
Experiments with Yeast data set.

	KM	BH	SAKM	SAGM	SAGMDE
Min SSE	58.359	59.006	68.382	58.289	**58.276**
Mean SSE	65.391	67.722	68.395	61.54	**58.276**
Max SSE	78.252	69.66	68.408	69.041	**58.276**
SD in SSE	5.9024	3.7599	0.0088	4.756	**0.0001**
Mean Time (ms)	8	353,116	137,627	48,742	460,449

Table 8
Experiments with synthetic data set ($K = 15$).

	KM	BH	SAKM	SAGM	SAGMDE
Min SSE	7.2	3.17	1.46	1.612	**1.446**
Mean SSE	10.817	3.388	1.475	3.708	**1.446**
Max SSE	16.29	4.666	1.57	6.772	**1.446**
SD in SSE	2.2576	0.477	0.0315	1.1394	**0.0001**
Mean Time (ms)	0.9	18,518	11,472	2722	33,267

Table 9
Experiments with synthetic data set ($K = 20$).

	KM	BH	SAKM	SAGM	SAGMDE
Min SSE	2.823	1.35	1.075	1.268	**1.071**
Mean SSE	4.988	1.699	1.076	1.663	**1.072**
Max SSE	7.543	2.037	1.08	2.151	**1.079**
SD in SSE	1.104	0.2179	**0.0009**	0.2838	0.0027
Mean Time (ms)	1.1	23,816	13,204	3530	27,083

4.1. Discussion

The SA algorithms tested attempt to minimize SSE, and our algorithm proved most successful in accomplishing this task. SAGMDE had the best mean performance compared to the other four algorithms for every tested data set except the Glass data set (Tables 2-10). Also, SAGMDE had the lowest minimum and maximum performance across all data sets; this demonstrates the utility of our algorithm's dual perturbation method. SAGMDE performed particularly well on the Yeast data set and the synthetic data set with $K = 15$. On these data sets, the maximum performance reached by the SAGMDE algorithm across the 20 runs was lower than the minimum performance reached by any of the other algorithms.

SAGMDE performs very consistently with a very low deviation in SSE across all runs for each data set. For 5 of the 9 data sets, the minimum and maximum SSE for the SAGMDE were the same, meaning SAGMDE found solutions with the same SSE across all 20 trials. No other algorithm had the same minimum and maximum SSE for any one data set. SAGMDE had the lowest standard deviation among all algorithms in every data set except the synthetic

Table 10
Experiments with synthetic data set ($K = 30$).

	KM	BH	SAKM	SAGM	SAGMDE
Min SSE	2.44	1.317	0.846	1.133	**0.844**
Mean SSE	3.363	1.683	0.85	1.61	**0.848**
Max SSE	4.198	2.104	0.862	2.095	**0.854**
SD in SSE	0.547	0.2066	0.0043	0.2667	**0.0029**
Mean Time (ms)	1.4	35,202	16,697	5717	47,351

Table 11
Descriptions of higher dimensional data sets.

Data set	# Points	# Attributes	# Classes (K)
Synthetic Control	600	60	6
Mice Protein	1077	68	8
LSVT Voice Rehabilitation	126	309	2

Table 12
Average sum of squares error across algorithms for higher dimensional data.

	KM	BH	SAKM	SAGM	SAGMDE
Synthetic Control	562.777	568.419	576.037	563.451	**539.075**
Mice Protein	821.057	813.888	812.589	822.859	**812.386**
LSVT Voice Rehabilitation	778.472	773.181	773.241	778.963	**769.863**

data set with $K = 20$,where it had the second lowest standard deviation.

While SAGMDE performs slower than the other algorithms, the algorithm's consistency between runs combined with its ability to find the better solutions compared to the other algorithms allows SAGMDE to yield better results with fewer runs.

4.2. Higher dimensional data

Following the same experimental procedure, we also collected experimental data on 3 data sets with a higher number of attributes from the Machine Learning Laboratory [37]. We adapted the parameters specified previously in Section 4 to ensure steady cooling by setting $\alpha$to 0.99 and setting $\alpha_{distort}$to 0.9925. A description of the data sets as well as the average SSE for the SA algorithms for each data set over 20 trials is shown below:

SAGMDE was the most successful algorithm in minimizing SSE. While BH and SAKM produced similar average SSE values compared to our algorithm in the Mice Protein data set, our algorithm found clusters that had a significantly lower average SSE compared to all the other algorithms for the Synthetic Control and Voice Rehabilitation data sets (Table 12).

4.3. Other metrics

SA-based partitional clustering algorithms have been exclusively evaluated using the Sum of Squares Error metric in existing literature. However, to provide a comprehensive performance evaluation, we have experimented with two other popular metrics: Normalized Mutual Information (NMI) and Adjusted Randomized Index (ARI). These metrics offer a measure of the correlation between the predicted labels and the true labels for a given data set. Experiments were run on the real data sets specified in Table 1 and Table 11 as well as the synthetic data sets specified in Fig. 2. The voice data set was excluded due to its highly non-linear nature. Results are detailed below.

In terms of average NMI and ARI, SAGMDE did not clearly outperform the other algorithms; however, our algorithm had the advantage of performing more consistently across runs (Tables 13-16). Our algorithm found clusters with the best average NMI for 4 out of 8 of the real data sets, and it found clusters with the best

Table 13
Average NMI across algorithms for various data sets.

	KM	BH	SAKM	SAGM	SAGMDE
Breast Cancer	**0.753**	0.75	0.751	0.749	0.748
Ecoli	0.598	0.561	0.6	0.594	**0.602**
Glass	0.334	0.342	0.33	**0.349**	0.33
Iris	0.697	**0.742**	**0.742**	**0.742**	**0.742**
Mice	0.316	**0.35**	0.327	0.33	0.322
Synthetic Control	0.749	**0.783**	0.737	0.748	0.756
Wine	0.83	0.852	0.852	0.85	**0.853**
Yeast	0.28	0.279	0.274	0.278	**0.284**
Synthetic ($K = 15$)	0.56	0.705	0.729	0.691	**0.744**
Synthetic ($K = 20$)	0.836	0.89	0.916	0.892	**0.926**
Synthetic ($K = 30$)	0.837	0.874	0.915	0.874	**0.918**

Table 14
Standard deviation for NMI across algorithms for various data sets.

	KM	BH	SAKM	SAGM	SAGMDE
Breast Cancer	0.0028	0.003	0.0034	0.0025	**0**
Ecoli	0.038	0.0127	0.0057	0.0249	**0.0038**
Glass	0.0299	0.014	0.0103	0.0222	**0.0077**
Iris	0.0477	**0**	**0**	**0**	**0**
Mice	0.0251	0.0018	**0.0015**	0.0308	0.0164
Synthetic Control	0.0096	0.0058	0.0103	0.012	**0**
Wine	0.0117	0.0025	0.0101	0.0067	**0**
Yeast	0.0063	0.0029	0.0018	0.0059	**0.0007**
Synthetic ($K = 15$)	0.0578	0.0101	0.0022	0.02	**0.0005**
Synthetic ($K = 20$)	0.0234	0.0105	0.0012	0.0102	**0.0009**
Synthetic ($K = 30$)	0.0129	0.0049	0.0034	0.0099	**0.0024**

Table 15
Average ARI across algorithms for various data sets.

	KM	BH	SAKM	SAGM	SAGMDE
Breast Cancer	**0.851**	0.848	0.849	0.848	0.847
Ecoli	0.437	0.366	**0.464**	0.425	0.46
Glass	0.199	0.206	0.185	**0.21**	0.187
Iris	0.655	**0.724**	**0.724**	**0.724**	**0.724**
Mice	0.185	**0.208**	0.183	0.199	0.193
Synthetic Control	0.597	**0.681**	0.623	0.587	0.59
Wine	0.848	**0.871**	0.87	0.868	**0.871**
Yeast	0.159	0.151	0.151	0.155	**0.161**
Synthetic ($K = 15$)	0.295	0.457	0.499	0.438	**0.511**
Synthetic ($K = 20$)	0.542	0.693	0.837	0.689	**0.839**
Synthetic ($K = 30$)	0.587	0.681	**0.825**	0.685	0.816

Table 16
Standard deviation for ARI across algorithms for various data sets.

	KM	BH	SAKM	SAGM	SAGMDE
Breast Cancer	0.0022	0.0024	0.0028	0.002	**0**
Ecoli	0.0575	0.0342	**0.0008**	0.0425	0.0065
Glass	0.0194	0.0092	0.0052	0.0249	**0.0048**
Iris	0.1014	**0**	**0**	**0**	**0**
Mice	0.0188	0.001	**0.0007**	0.0225	0.0152
Synthetic Control	0.0328	0.0339	0.0343	0.0281	**0**
Wine	0.0098	0.0033	0.0125	0.0077	**0**
Yeast	0.0079	0.0066	0.0009	0.0064	**0.0008**
Synthetic ($K = 15$)	0.0566	0.0158	0.0023	0.0272	**0.0007**
Synthetic ($K = 20$)	0.0402	0.0303	0.0028	0.0418	**0.0016**
Synthetic ($K = 30$)	0.0308	0.0119	0.0097	0.0257	**0.0077**

average ARI for 3 out of 8 of these data sets. SAGMDE still performed well in cases where it did not produce the best average NMI or ARI: the difference in average NMI and ARI between SAGMDE and the best performing algorithm for any of the tested data sets was less than 0.03 with the exception of ARI performance for the synthetic control data set.

Our algorithm proved most effective in minimizing standard deviation across the 20 runs for both metrics. SAGMDE had the lowest NMI standard deviation for 7 of the 8 real data sets as well

as the lowest ARI standard deviation for 6 of the 8 real data sets. SAGMDE reached a standard deviation of 0 for 4 of these data sets. Therefore, the results once again highlight the consistency of the SAGMDE algorithm across runs as well as its ability to return quality solutions.

It is also worth noting that SAGMDE clearly outperformed the other algorithms for the 3 synthetic data sets. These data sets contain linearly separable groups of overlapping clusters, so our algorithm's better performance on these data sets reflects its strength of evenly distributing centers among groups of overlapping clusters. This strength stems from the distortion equalization component of SAGMDE as seen in Fig. 1, so the results further display the usefulness of our algorithm's dual perturbation method.

4.4. Parameter setting

Parameters specified earlier in this section were chosen conservatively to produce quality clusters for a wide range of data sets. If the user wishes to modify the provided parameters, we recommend one of the following changes:

1. Increasing αor $MaxIter$can lead to more thorough cooling and better clustering.
2. An increase in Tallows for greater cost increases to be accepted at the beginning of the algorithm, which can help the algorithm escape deeper local minima.
3. A decrease in $T_f$increases the precision of the convergence to the best minimum found by the algorithm, and thus the final SSE value.

Note all these changes come at the expense of runtime. A decrease in α,$MaxIter$,or T,or an increase in T_f,would improve runtime.

When modifying α,T,or T_f,the corresponding cooling parameter for the distortion equalization perturbation should be modified as described below to prevent disrupting the relative cooling schedules between the two perturbation methods.

First, note that we can calculate the number of iterations of the primary loop containing both perturbation methods, which we will denote as $numIt$. We can write a relationship between T,α,T_f,and $numIt$since the current temperature, which starts at T,is multiplied by αduring every iteration of the loop until $T_f$is reached. From this relationship, we can solve for $numIt$as seen in Eq. (9).

$$T_f = \alpha^{numIt} T \quad \rightarrow \quad numIt = \frac{ln(T_f) - ln(T)}{ln(\alpha)} \tag{9}$$

Then, upon modification of α,$numIt$should be recalculated using Eq. (9), and the new value for $\alpha_{distort}$can be determined using Eq. (10):

$$\alpha_{distort} = \left(\frac{T_{distort\_f}}{T_{distort}} \right)^{1/numIt} \tag{10}$$

If the user chooses to modify the initial temperatures, the new initial temperatures ($T_{new}$and $T_{distort\_new}$) should be computed pairwise as described below, where $a$is a user-defined value:

$$T_{new} = \alpha^a T \tag{11}$$

$$T_{distort\_new} = \alpha_{distort}^a T_{distort} \tag{12}$$

Since $\alpha < 1$,choosing a positive $a$will decrease the initial temperature while choosing a negative $a$will increase the initial temperature. Similarly, to modify the final temperatures, the following formulas should be used to calculate the new final temperatures ($T_{f_new}$and $T_{distort\_f\_new}$), where $b$is a user-defined value:

$$T_{f\_new} = \alpha^b T_f \tag{13}$$

$$T_{distort\_f\_new} = \alpha_{distort}^b T_{distort\_f} \tag{14}$$

A positive value of $b$decreases the final temperature while a negative value of $b$increases the final temperature.

We ran experiments on the Ecoli data set from the Machine Learning Laboratory [37] (a description of the data set is provided in Table 1) to determine the impact of modifying the parameters (T,T_f,α,$MaxIter$) on the runtime of SAGMDE. Corresponding cooling parameters for the distortion equalization perturbation were adjusted accordingly as specified above. The results shown in Fig. 3 indicate that $\log(T)$,$\log(T_f)$,and $MaxIter$have a linear relationship with runtime. In the graph illustrating the relationship between runtime and α,the trendline has the equation $t = -\frac{c}{\ln(\alpha)}$where $t$is the runtime and $c$is a constant, so runtime is proportional to $-\frac{1}{\ln(\alpha)}$.

4.5. Convergence

We experimentally show that SAGMDE converges to a given cost, where cost is calculated by the SSE performance metric. SAGMDE was run for the first 250 iterations of the main loop using the recommended parameters for the Ecoli, Glass, Wine, and Yeast data sets from the Machine Learning Laboratory [37] (details on these data sets used can be found in Table 1). From Fig. 4, we see that SAGMDE converges within these first 250 iterations of the main loop for all four data sets. With our recommended parameters, 362 iterations of the loop are executed, which is sufficient for our algorithm to converge.

5. Generative art comparison

Generative art uses an autonomous system, which is almost always a computer, to create art. By introducing a degree of randomness into the creation of the artwork, the same set of instructions can produce similar but unique pieces of art. In this paper, generative art offers a route to further explore and compare partitional clustering algorithms.

Our generative art allows for comparison between any two center-based, partitional clustering algorithms. To produce the art, both algorithms to be compared are run 20 times using the same user-chosen data set, and information captured during these runs affect various components of the artwork. Below, we used our code to generate art comparing K-means with SAGMDE (as seen in Fig. 5) as well as SAGM with SAGMDE (as seen in Fig. 6). Both artworks shown below were produced using the synthetic data set ($K = 20$) described in Section 4. Unique pieces of art can be produced using the same algorithms and data set due to the variation in the initial location of cluster centers by the Random Partition method.

The art highlights a variety of features from both algorithms through its three main components: the large concentric background circles, the line segments directed towards the center of the circle, and the smaller outlined shapes scattered about the image.

The background circles provide information regarding the runtime of the algorithms. Specifically, we associate a shade value for each given radius from the center, and this value corresponds to the percentage of runs such that the ratio between the current runtime to the maximum runtime is less than the ratio between the current radius and the maximum radius. Therefore, the ratio of runtimes between the fastest run and the slowest run is equal to the ratio between the radius of the innermost, darkest circle and the maximum radius. In Fig. 5, we see that the background circles for KM (shown on the left of the image) gradually become lighter

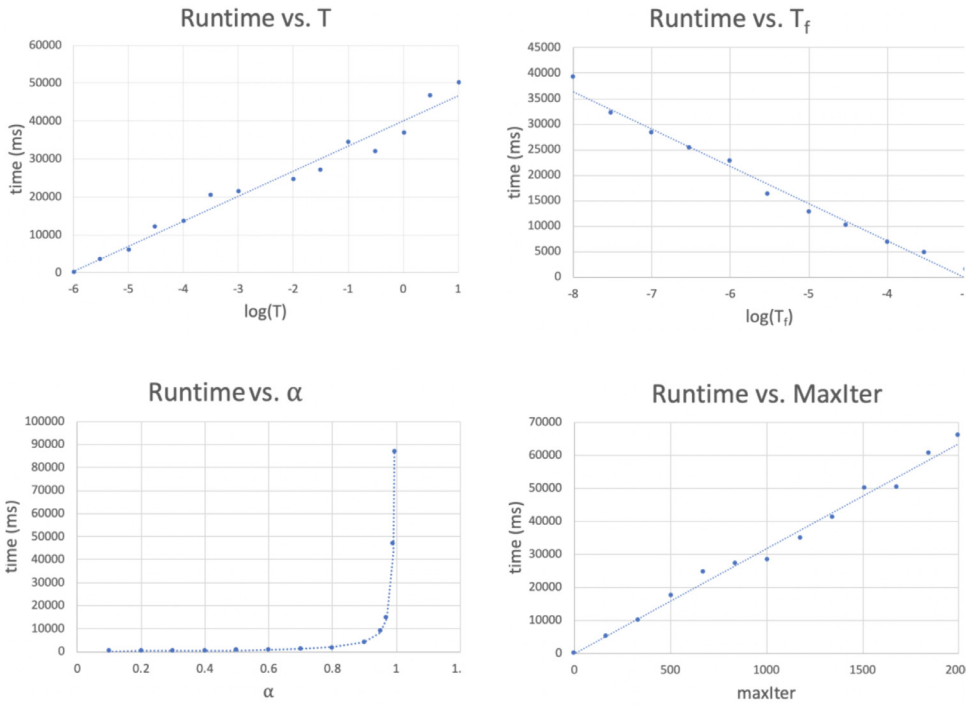

Fig. 3. Impact of SAGMDE parameters on runtime. The mean runtime of 5 trials was used to generate each data point. The trendline is shown for each graph.

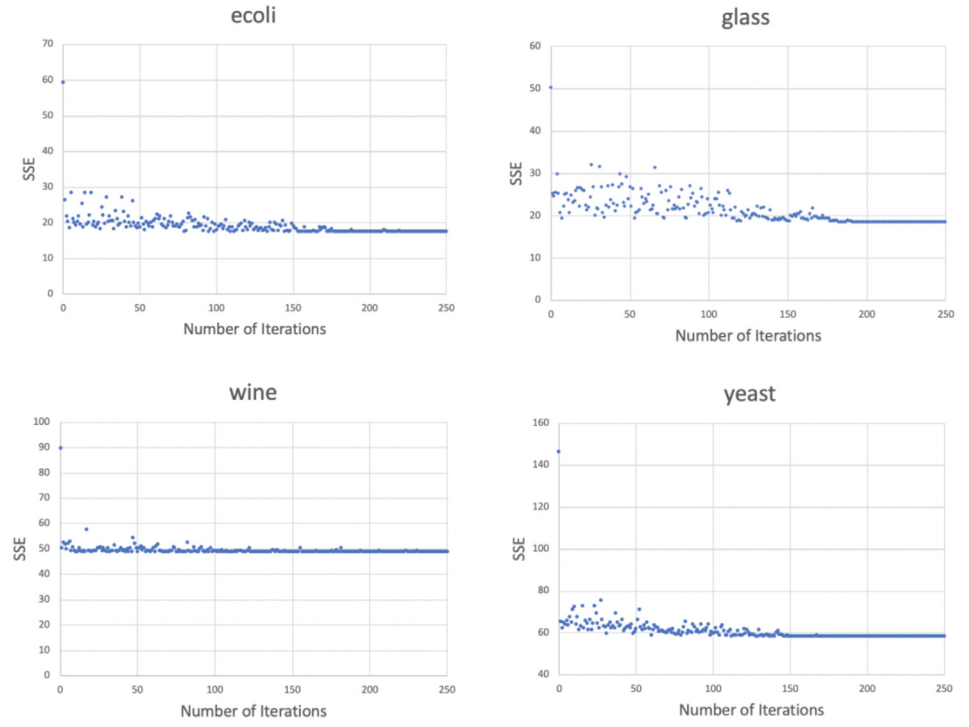

Fig. 4. Current SSE performance for various real data sets after a given number of iterations of the SAGMDE algorithm.

as the radius increases, and this steady gradient indicates a large deviation in runtime. However, the innermost background circle for SAGMDE (shown on the right of the image) has a radius close to the maximum radius, so SAGMDE had a relatively consistent runtime as the ratio of runtimes between the fastest and slowest runs is close to 1. For the same reason, it can be seen in Fig. 6 that SAGM also had a relatively consistent runtime.

While the shade presents the deviations in runtimes of a specific algorithm, the blue tint can be used to compare absolute runtimes between the two algorithms. The blue pixel value of the background is multiplied by a value proportional to the square of the log of the median runtime, so there will be a more prominent blue tint if the algorithm had a longer runtime for the given data set. As SAGMDE appears to have a stronger blue tint in both

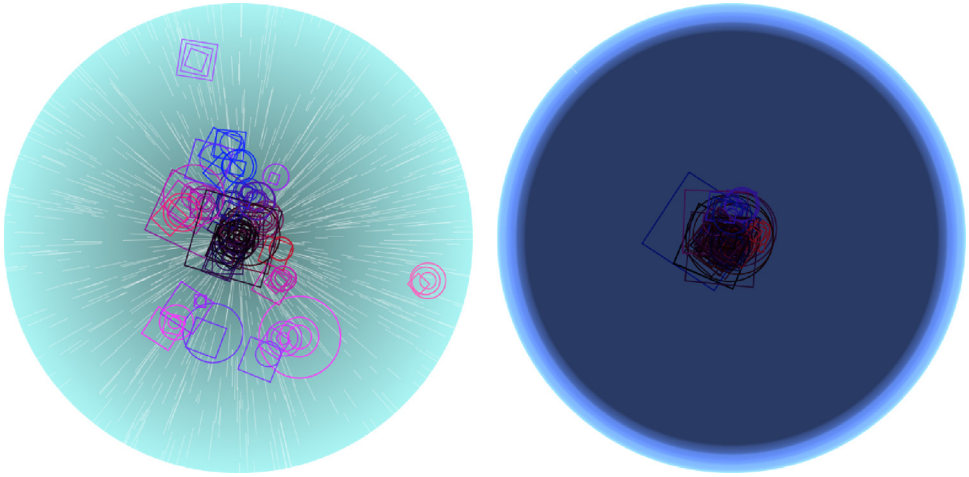

Fig. 5. (Color necessary) A generative art comparison between KM (left) and SAGMDE (right). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

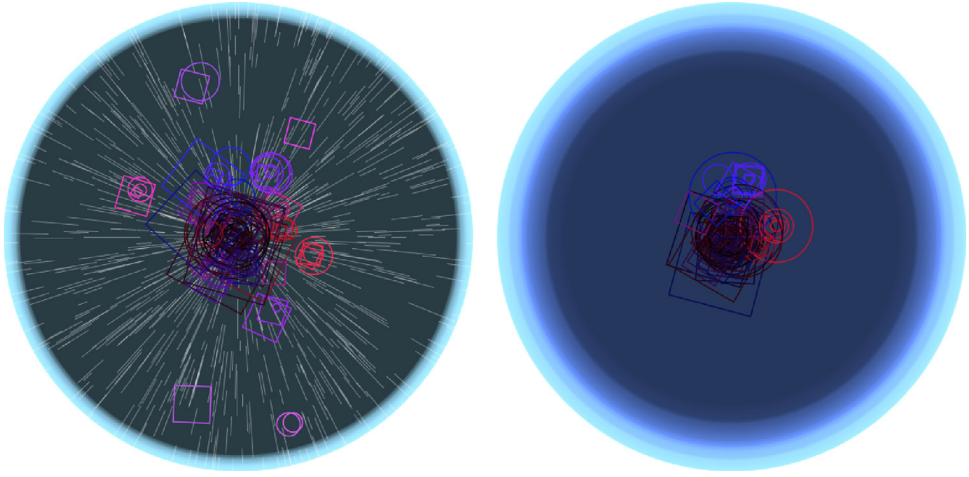

Fig. 6. (Color necessary) A generative art comparison between SAGM (left) and SAGMDE (right). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Figs. 5 and 6, SAGMDE had a longer median runtime than both KM and SAGM.

The line segments in the art highlight deviations in the performance of the algorithms, which is based upon the Sum of Squares Error. The number of line segments is directly proportional to the standard deviation of the algorithms performance across the 20 runs. In Figs. 5 and 6, the circle in the left of the image, representing KM and SAGM respectively, has significantly more line segments than the circle in the right of the image, representing SAGMDE, which shows that both KM and SAGM had a much higher deviation in performances than SAGMDE for the provided data set.

The outlined shapes illustrate the consistency of each algorithm in finding the same local optima in multiple runs. The displacement of the center of each shape from the focal point of the background circles is directly proportional to the offset of the centers found by the algorithm from the closest true center in the data set. The displacements of the centers are scaled for both sides of the image based on the displacements found from the first algorithm run such that the maximum displacement is near the largest radius of a background circle. Both images are scaled by the same factor, so the average distance of the centers of the shapes from the corresponding focal point can be used to compare the algorithms. If the algorithm was able to successfully identify all the true centers, every shapes center would be located at the focal point of the background circles, but note that the set of centers that minimizes Sum of Squares Error is not necessarily the set of true centers. However, the optimal solution should still generally have centers that are closer to the focal point. In Figs. 5 and 6, the centers in the right of the image are significantly closer to the focal point compared to the centers on the left of the image, demonstrating that SAGMDE performed better than both KM and SAGM. The shapes were generated based on the first six runs of each algorithm. If the algorithm were to find the global optimum each time, then the outlined shapes would form groups of six which share identical centers. In Fig. 5, on the left of the image, note the high number of groups with between two and four outlined shapes, which shows that KM is finding different local optima in different runs. This is also evident on the left of the image for Fig. 6, where the six centers with the largest displacements are either isolated or form groups of two, which shows SAGM also finds different local optima. In both figures, the outlined shapes on the right of the image tend to form groups of six, suggesting SAGMDE can more consistently find the same optima.

6. Conclusion

This paper describes a new SA-based partitional clustering algorithm, SAGMDE, which yields better clusters and performs more

consistently than similar existing algorithms. SAGMDE takes advantage of two different perturbation methods—one based on a Gaussian mutation and the other based on the equalization of center distortions—to allow for both small shifts for converging upon the best nearby local minimum and large shifts to find better local minimums elsewhere. The algorithm utilizes complementary cooling between two perturbation methods with different cooling schedules, and we demonstrate how this dual cooling schedule can be implemented without an increase in the number of parameters. Although slower than the SAGM algorithm that only uses the Gaussian perturbation method, this dual strategy proved effective in consistently producing higher quality clusters for a variety of data sets compared to SAGM and other preexisting SA algorithms. We aim to reduce the runtime of SAGMDE in the future while maintaining the quality of the solutions.

The paper also introduces generative art as a new creative tool to analyze the various performance benchmarks of the partitional clustering algorithms. The generative art shown in this paper changes based upon each algorithm's performance over multiple runs such that two of these algorithms can be compared side by side through analysis of the art. Generative art can become a new method for comparing optimizations of any given algorithm in the future.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to thank Dr. Marie-Pierre Jolly and Mingxiao Song for their helpful suggestions in the revision process. The authors would also like to thank the Pioneer Academics Team for their support during the paper writing process.

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.patcog.2020.107713 .

References

[1] D.E. Brown, C.L. Huntley, A practical application of simulated annealing to clustering, Pattern Recognit. 25 (4) (1992) 401–412, doi:10.1016/0031-3203(92)90088-Z.
[2] B. Beddad, K. Hachemi, J. Postaire, F. Jabloncik, O. Messai, An improvement of spatial fuzzy c-means clustering method for noisy medical image analysis, in: 2019 6th Int. Conf. on Image and Signal Process. and their Appl. (ISPA), IEEE, 2019, pp. 1–5.
[3] S.H. Shihab, S. Afroge, S.Z. Mishu, RFM Based market segmentation approach using advanced k-means and agglomerative clustering: a comparative study, in: 2019 Int. Conf. on Electr., Comput. and Commun. Eng. (ECCE), IEEE, 2019, pp. 1–4.
[4] D. Aloise, A. Deshpande, P. Hansen, P. Popat, NP-hardness of euclidean sum-of-squares clustering, Mach. Learn. 75 (2) (2009) 245–248, doi:10.1007/s10994-009-5103-0.
[5] S. Merendino, M.E. Celebi, A simulated annealing clustering algorithm based on center perturbation using gaussian mutation, in: C. Boonthum-Denecke, G.M. Youngblood (Eds.), Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference, The AAAI Press, 2013, pp. 456–461.
[6] M. Verma, M. Srivastava, N. Chack, A. Diswar, N. Gputa, A comparative study of various clustering algorithms, Data Min., Int. J. Eng. Res. Appl. (IJERA) (2012) 1379–1384.
[7] F. Kuwil, F. Shaar, A. Topcu, F. Murtagh, A new data clustering algorithm based on critical distance methodology, Expert Syst. Appl. 129 (2019) 296–310, doi:10.1016/j.eswa.2019.03.051.
[8] F. Kuwil, U. Atila, R. Abu-Issa, F. Murtagh, A novel data clustering algorithm based on gravity center methodology, Expert Syst. Appl. 156 (2020), doi:10.1016/j.eswa.2020.113435.
[9] M. Ahmed, R. Seraj, S.M.S. Islam, The k-means algorithm: a comprehensive survey and performance evaluation, Electronics 9 (8) (2020) 1295, doi:10.3390/electronics9081295.
[10] Z. Güngör, A. Ünler, K-harmonic means data clustering with simulated annealing heuristic, Appl. Math. Comput. 184 (2) (2007) 199–209, doi:10.1016/j.amc.2006.05.166.
[11] D. Pelleg, A. Moore, X-means: extending k-means with efficient estimation of the number of clusters, in: P. Langley (Ed.), Proceedings of the 17th International Conference on Machine Learning, Morgan Kaufmann, 2000, pp. 727–734.
[12] Y. Guan, A.A. Ghorbani, N. Belacel, K-means+: An Autonomous Clustering Algorithm, Technical Report TR04-164, UNB Faculty of Computer Science, Canada, 2004, doi:10.13140/RG.2.1.1113.7365.
[13] E.W. Forgy, Cluster analysis of multivariate data: efficiency vs. interpretability of classifications, Biometrics 21 (3) (1965) 768–769.
[14] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: L.M.L. Cam, J. Neyman (Eds.), Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, University of California Press, 1967, pp. 281–297.
[15] M.E. Celebi, H. A.Kingravi, P.A. Vela, A comparative study of efficient initialization methods for the k-means clustering algorithm, Expert Syst. Appl. 40 (1) (2013) 200–210, doi:10.1016/j.eswa.2012.07.021.
[16] P. Fränti, S. Sieranoja, How much can k-means be improved by using better initialization and repeats? Pattern Recognit. 93 (2019) 95–112, doi:10.1016/j.patcog.2019.04.014.
[17] J.C. Dunn, A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters, J. Cybern. 3 (3) (1973) 32–57, doi:10.1080/01969727308546046.
[18] B. Zhang, M. Hsu, U. Dayal, K-Harmonic Means - A Data Clustering Algorithm, Technical Report HPL-1999-124, Hewlett-Packard Labs, 1999.
[19] A.M. Bagirov, N. Karmitsa, S. Taheri, Metaheuristic clustering algorithms, in: Partitional Clustering via Nonsmooth Optimization, Unsupervised and Semi-Supervised Learning, Springer, 2020, pp. 165–183, doi:10.1007/978-3-030-37826-4.
[20] P. Hansen, N. Mladenović, J-Means: a new local search heuristic for minimum sum of squares clustering, Pattern Recognit. 34 (2) (2001) 405–413, doi:10.1016/S0031-3203(99)00216-2.
[21] D. Zhang, C. Hao, C. Wu, D. Xu, Z. Zhang, Local search approximation algorithms for the k-means problem with penalties, J. Comb. Optim. 37 (2019) 439–453, doi:10.1007/s10878-018-0278-6.
[22] K.S. Al-Sultan, A tabu search approach to the clustering problem, Pattern Recognit. 28 (9) (1995) 1443–1451, doi:10.1016/0031-3203(95)00022-R.
[23] Y. Lu, B. Cao, C. Rego, F. Glover, A tabu search based clustering algorithm and its parallel implementation on spark, Appl. Soft Comput. 63 (2018) 97–109, doi:10.1016/j.asoc.2017.11.038.
[24] F. Glover, M. Laguna, Tabu search, in: D.Z. Du, P.M. Pardalos (Eds.), Handbook of Combinatorial Optimization, Springer, Boston, MA, 1998, pp. 2093–2229.
[25] K. Krishna, M.N. Murty, Genetic k-means algorithm, IEEE Trans. Syst. Man Cybern. B Cybern. 29 (3) (1999) 433–439, doi:10.1109/3477.764879.
[26] M.Z. Islam, V. Estivill-Castro, M.A. Rahman, T. Bossomaier, Combining k-means and a genetic algorithm through a novel arrangement of genetic operators for high quality clustering, Expert Syst. Appl. 91 (2018) 402–417, doi:10.1016/j.eswa.2017.09.005.
[27] S.J. Nanda, G. Panda, A survey on nature inspired metaheuristic algorithms for partitional clustering, Swarm Evol. Comput. 16 (2014) 1–18, doi:10.1016/j.swevo.2013.11.003.
[28] Y. ming Cheung, On rival penalization controlled competitive learning for clustering with automatic cluster number selection, IEEE Trans. Knowl. Data Eng. 17 (11) (2005) 1583–1588, doi:10.1109/TKDE.2005.184.
[29] H. Jia, Y. ming Cheung, J. Liu, Cooperative and penalized competitive learning with application to kernel-based clustering, Pattern Recognit. 47 (2014) 3060??3069, doi:10.1016/j.patcog.2014.03.010.
[30] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equations of state calculations by fast computing machines, J. Chem. Phys. 21 (6) (1953) 1087–1092, doi:10.1063/1.1699114.
[31] R.A. Rutenbar, Simulated annealing algorithms: an overview, IEEE Circuits Devices Mag. 5 (1) (1989) 19–26, doi:10.1109/101.17235.
[32] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Sci. 220 (4598) (1983) 671–680, doi:10.1126/science.220.4598.671.
[33] D. Henderson, S.H. Jacobson, A.W. Johnson, The theory and practice of simulated annealing, in: F.W. Glover, G.A. Kochenberger (Eds.), Handbook of Metaheuristics, International Series in Operations Research and Management Science, vol. 57, Springer, Boston, MA, 2003, pp. 110–127, doi:10.1007/0-306-48056-5_10.
[34] R.W. Klein, R.C. Dubes, Experiments in projection and clustering by simulated annealing, Pattern Recognit. 22 (2) (1989) 213–220, doi:10.1016/0031-3203(89)90067-8.
[35] S. Bandyopadhyay, U. Maulik, M.K. Pakhira, Clustering using simulated annealing with probabilistic redistribution, Int. J. Pattern Recognit. Artif. Intell. 15 (2) (2001) 269–285, doi:10.1142/S0218001401000927.
[36] G. Patané, M. Russo, The enhanced LBG algorithm, Neural Netw. 14 (9) (2001) 1219–1237, doi:10.1016/S0893-6080(01)00104-6.
[37] D. Dua, C. Graff, UCI machine learning repository, 2019.

[38] Dataset, J. Lee, D. Perkins, Synthetic clustering dataset (k = 15), 2020a, (Mendeley Dataa). 10.17632/phj2x27rjj.2
[39] Dataset, J. Lee, D. Perkins, Synthetic clustering dataset (k = 20), 2020b, (Mendeley Datab). 10.17632/fgsx9hn8zh.2
[40] Dataset, J. Lee, D. Perkins, Synthetic clustering dataset (k = 30), 2020c, (Mendeley Datac). 10.17632/b6pyfcwn4r.2

Julian Lee is a student at The Pingry School graduating in 2021. His current research interests include machine learning and image processing.

David Perkins received his Ph.D. in Mathematics from the University of Montana and now teaches in the computer science department at Hamilton College in central New York. His research interests are primarily related to artificial intelligence.