A Simulated Annealing Clustering Algorithm Based on Center Perturbation Using Gaussian Mutation

Stephen Merendino and **M. Emre Celebi**
Department of Computer Science,
Louisiana State University in Shreveport,
Shreveport, LA, USA
ecelebi@lsus.edu

Abstract

Clustering, the unsupervised classification of objects into groups, is a widely used technique in exploratory data analysis. The clustering problem is a very complex one, and a popular heuristic for solving it is the Simulated Annealing (SA) algorithm. SA is an approximation algorithm that involves generating a neighborhood solution by perturbing the current solution in a small, yet meaningful way. This new solution is accepted with a probability of 1 if it is quantitatively better than the current solution, and accepted according to the Metropolis criterion otherwise. Cluster quality is measured using the Sum of Squared Error (SSE) criterion. This paper presents an SA algorithm that uses a new type of perturbation to generate solutions. Whereas most SA clustering algorithms perturb data point memberships directly, our algorithm perturbs a randomly chosen center using Gaussian mutation, and then reassigns data points in a nearest neighbor fashion. Experimental results on a diverse collection of data sets demonstrate that our algorithm has comparable effectiveness to other SA algorithms, while being much faster due to its simplicity.

1 Introduction

Clustering is one of the most important tasks in exploratory data analysis (Jain, Murty, and Flynn 1999). In a general sense, the purpose of clustering is to group a set of data points so that the points in any cluster are more similar to each other than to the points in other clusters. Clustering has been used in numerous fields including as statistics, economics, physics, psychology, biology, pattern recognition, engineering, and marketing (Brown and Huntley 1992).

The dissimilarity between data points and the quality of the clustering is commonly quantified using the squared Euclidean distance. The problem of clustering n data points into k clusters is solved by minimizing the following objective function (Selim and Alsultan 1991):

$$J(W, Z) = \sum_{i=1}^{n} \sum_{j=1}^{k} w_{ij} d_{ij}^2 \tag{1}$$

subject to

$$\sum_{j=1}^{k} w_{ij} = 1, \quad 1 \leq i \leq n \tag{2}$$

$$w_{ij} = \begin{cases} 1 & \text{if point } i \text{ is assigned to cluster } j, \\ 0 & \text{otherwise,} \end{cases}$$

$$1 \leq i \leq n, \quad 1 \leq j \leq k$$

where d_{ij} denotes the Euclidean distance between point i and the center of cluster j, and $w_{ij} \in \{0, 1\}$ ensures that points remain exclusive to their respective clusters.

Clustering algorithms can be roughly divided into two categories: hierarchical and partitional. Hierarchical algorithms find nested clusters either in a divisive or agglomerative fashion. Partitional algorithms, on the other hand, find all clusters simultaneously and do not impose a hierarchical structure. Most hierarchical algorithms have quadratic or superquadratic asymptotic time complexity (Jain, Murty, and Flynn 1999) and thus are not suitable for processing large data sets, while partitional algorithms often have linear or loglinear complexity. In this paper, we are concerned with hard-partitional clustering algorithms, where each data point belongs to one and only one cluster. Eq. (3) shows a general formulation of the clustering problem.

General Clustering Problem (3)

Let
    $\mathcal{X}$ be the set of points to be clustered,
    $\mathcal{P}$ be the set of all possible clusterings of $\mathcal{X}$,
    $J : \mathcal{P} \to \mathbb{R}$ be an internal clustering criterion;
Then
    Minimize $J(p)$
Subject To
    $p \in \mathcal{P}$

Clustering is a combinatorial optimization problem. These are typically NP-hard problems (Aloise et al. 2009) concerned with finding the optima of functions of discrete variables (Aarts and van Laarhoven 1989). There are two ways to solve a combinatorial optimization problem: optimization algorithms and approximation algorithms (Rutenbar 1989). Optimization algorithms return the globally optimum solution, whereas approximation algorithms (also called "heuristics") return a solution that is "close" to the global optimum (Aarts and van Laarhoven 1989; Rutenbar 1989). In theory, optimization algorithms seem to be the better choice because they return the best solution, but in practice they can be quite difficult to design and implement, and they are usually problem specific. More importantly, they can be extremely slow on large-scale problems. Approximation algorithms, on the other hand, are often easier to implement, more generic, and faster. SA is a popular approximation algorithm that has been applied to a variety of optimization problems including clustering.

SA is based on the physical annealing process in which a material is heated up in a heating bath allowing the atoms to move freely. After being heated, the material is cooled carefully and slowly so that the atoms can settle into a good crystal or low-energy state (Rutenbar 1989). SA's origins date back to 1953 (Metropolis et al. 1953), when a group of researchers led by Nicholas Metropolis developed an algorithm to model this physical annealing process (Aarts and van Laarhoven 1989). The pseudocode of the Metropolis algorithm is given in Algo. 1 (Metropolis et al. 1953). Here, K_B and t denote the Boltzmann constant and temperature of the solid, respectively.

Algorithm 1 The Metropolis Algorithm

```
1: Select an initial state $i \in S$;
2: Generate a new state by applying a small randomly generated perturbation
3: Calculate the change in energy $\Delta E$
4: **if** $\Delta E < 0$ **then**
5:     The new state is accepted as the starting point for the next move
6: **else if** $\Delta E > 0$ **then**
7:     The new state is accepted with probability $\exp(-K_B \Delta E/t)$
8: **end if**
```

To implement the SA algorithm, four things must be defined (Aarts and van Laarhoven 1989; Rutenbar 1989):

- **Configurations/Solutions:** This is a model for what an acceptable solution to the problem is. It is comparable to the "state" of the solid in the Metropolis algorithm.
- **Perturbation Method:** This is the method that slightly changes the current solution in a small, yet meaningful way, usually by way of a swap or permutation, to generate a neighborhood solution to the current solution. This is comparable to the atoms moving freely in the Metropolis algorithm.
- **Cost Function:** A function that quantifies the quality of a solution. The result is comparable to the "energy" of the solid in the Metropolis Algorithm.
- **Cooling Schedule:** The cooling schedule consists of many components. Its purpose is to determine the initial temperature, determine when and by how much the temperature should be lowered, and when the annealing process is done.

In physical annealing, temperature has a real meaning, but in SA temperature functions as a control parameter (Aarts and van Laarhoven 1989; Rutenbar 1989). The purpose of the temperature is to control the probability that cost increasing moves are accepted. When starting the algorithm, the temperature should be high enough so that the probability of acceptance is close to 1. The temperature parameter is decreased in a monotonic fashion so that the probability of acceptance eventually reaches zero as the algorithm progresses. At each temperature, the system must be allowed to reach "thermal equilibrium". This means that at each temperature enough perturbations must be made to get a good sampling of the neighborhood for that temperature. This helps ensure that by the time the algorithm reaches the final stages it has escaped all local minima and is close to a globally optimum solution. After termination, the final configuration is taken as the solution to the problem. The conventional SA algorithm is given in Algo. 1 (Klein and Dubes 1989).

There are two main approaches to implementing the cooling schedule. The standard way is to implement a cooling schedule that consists of an initial temperature, final temperature, a monotone decreasing function to decrement the temperature, as well as the number of iterations per temperature. This is the multi Markov chain (MMC) approach to the cooling schedule, where each Markov chain corresponds to a certain temperature. The alternative is the single Markov chain (SMC) approach, which has a significantly simpler, yet effective cooling schedule. Instead of stepping through multiple Markov chains, a single temperature parameter is set at the beginning of execution and used throughout the entire course of the algorithm. The main parameter that must be tuned in an SMC cooling schedule is the number of iterations per temperature.

Algorithm 2 Conventional SA Algorithm

```
1: Choose initial temperature $t_0$.
2: Choose final temperature $t_f$.
3: Choose temperature function $f$, such that $f$ is monotonically decreasing.
4: Choose initial random configuration of clusters.
5: **repeat**
6:     **repeat**
7:         perturb configuration $i$ with cost $C_i$ to configuration $j$ with cost $C_j$,
8:         **if** $(\Delta C_{ij} = C_j - C_i) \leq 0$ **then**
9:             accept configuration $j$
10:        **else**
11:            accept configuration $j$ with probability $\exp(-\Delta C_{ij}/t_r)$
12:        **end if**
13:    **until** quasi-equilibrium at $t_r$ is reached
14:    $t_{r+1} = f(t_r)$.
15: **until** $t_{r+1} \leq t_f$ indicating that the system is frozen
```

There are two types of decisions that must be made when implementing an SA algorithm: generic choices and problem specific choices (Dowsland and Thompson 2012). Generic choices concern setting the parameters of the cooling schedule, whereas problem specific choices focus on the cost function, neighborhood structure, and perturbation operator.

This paper presents a new type of SA algorithm for clustering. This algorithm differs from other SA clustering algorithms in how it solves the partitional clustering problem by a specific choice of perturbation method. The rest of the paper is organized as follows. Related work is presented in Section 2. The proposed algorithm is detailed in Section 3. Experimental results are given in Section 4. Finally, conclusions are given in Section 5.

2 Related Work

There are several SA based approaches to the clustering problem. These SA algorithms usually differ in one or more of the categories listed in Section 1. In most cases, the solution space and the cost function will be the same, but the perturbation method and the cooling schedule will differ drastically.

Klein and Dubes (Klein and Dubes 1989) presented an SA algorithm (K & D), where the perturbation operator changes the membership of a single randomly chosen point to a different, randomly chosen cluster. This is a small enough move to ensure that the new solution is in the neighborhood of the current solution. The cooling schedule parameters are set automatically by measuring the initial cost and variance at a high temperature. The only value that must be set manually is ϵ, and its value should be close to zero. The cooling schedule values that are needed are given below:

Markov chain length - the number of accepted moves at one temperature;

t_0 - the initial value of the temperature;

$t_{r+1} = f(t_r)$ - the decrement rule;

t_f - the final value of the control temperature;

δ - a number close to zero that controls the rate of cooling;

ϵ - a number close to zero that controls the freezing point.

The notation $C_i(t_r)$ refers to the cost of configuration i when the temperature is t_r. The term $\overline{C}(t_r)$ is the average cost over n accepted moves (n: number of data points) achieved after the chain has reached equilibrium, that is:

$$\overline{C}(t_r) = \frac{1}{n}\sum_{i=1}^{n} C_i(t_r) \tag{4}$$

$\overline{C^2}(t_r)$ approximates the second moment of cost. It is the same as $\overline{C}(t_k)$, but the cost is squared before summation.

$$\overline{C^2}(t_r) = \frac{1}{n}\sum_{i=1}^{n} C_i^2(t_r) \tag{5}$$

The sample variance of cost is defined as:

$$\sigma^2(t_r) = \overline{C^2}(t_r) - [\overline{C}(t_r)]^2 \tag{6}$$

The initial value of the temperature is given by:

$$t_0 = \sigma(\infty) \tag{7}$$

where $\overline{C^2}(\infty)$ and $\overline{C}(\infty)$ are computed from an initial Markov chain at a very high value of t. The decrement rule f is established as follows:

$$t_{r+1} = t_r \left(1 + \frac{\ln(1+\delta)\cdot t_r}{3\sigma(t_r)}\right)^{-1} \tag{8}$$

The final value of the control parameter t_f is taken to be the first value that satisfies the following equation:

$$\frac{\sigma^2(t_f)}{t_f(\overline{C}(t_0) - \overline{C}(t_f))} < \epsilon \tag{9}$$

Brown and Huntley (Brown and Huntley 1992; 1995) present an algorithm (B & H), where the perturbation operator transfers a randomly chosen point i from its present cluster to another randomly chosen cluster. Most SA algorithms cycle through every point and transfer them with a certain probability, but this algorithm transfers a randomly chosen point with probability of 1 in the exact same way as the K & D algorithm. The B & H algorithm differs from the K & D algorithm by the methods and mathematical formulae used to determine the initial temperature, final temperature, and Markov chain length. The algorithm also accounts for tracking empty clusters versus non-empty clusters, opting to place the randomly chosen point i in an empty cluster preferably, but only if empty clusters exist. The cooling schedule parameters are calculated based on run-time statistics gathered from a fixed number of trial perturbations in a similar fashion to the K & D algorithm. The total number of perturbations tried in any run is MaxIt multiplied by NumTemp, where MaxIt is a fixed multiple of the number of objects to be clustered and NumTemp is a user-defined constant. α is the decrement factor that is used to determine the new temperature, i.e., $t_{r+1} = \alpha t_r$. The initial temperature is given by:

$$t_0 = \frac{\mu^+}{\log\left(\dfrac{m^+}{\chi m^+ - (1-\chi)(\text{MaxIt} - m^+)}\right)} \tag{10}$$

where

m^+ : the number of cost increases in MaxIt trial random perturbations

u^+ : the average cost increase in MaxIt trial random perturbations

χ : the acceptance ratio, a real value in $(0, 1)$

The final temperature is given by:

$$t_f = -\frac{\beta\mu^+}{\log\epsilon} \tag{11}$$

where $0 < \epsilon < 1$ and $0 < \beta < 1$. This represents the algorithm accepting a cost increase of $-\beta\mu^+$ with probability ϵ at the final temperature t_f. Given NumTemp, t_0, and t_f, the calculation of α is straightforward:

$$\alpha = \left(\frac{t_f}{t_0}\right)^{1/\text{NumTemp}} \tag{12}$$

With sufficiently large NumTemp and MaxIt and sufficiently small $(1-\chi)$, β, and ϵ, the annealing schedule ensures slow, steady convergence to a near global optimum solution. The authors recommend values of NumTemp $=$ 200, MaxIt $= 4n$, $\chi = 0.75$, $\beta = 0.125$, and $\epsilon = 0.00000000001$.

Bandyopadhyay *et al.* presented an SA algorithm called SAKM (Bandyopadhyay, Maulik, and Pakhira 2001). Instead of opting for a completely random perturbation method, the SAKM algorithm uses the distance between each data point and its nearest center to calculate the transfer probability of the point. The farther away a data point is from its current cluster center, the more likely it is transferred to another cluster. Data point $\mathbf{x}_i$ in cluster $\mathcal{C}_j$ is transferred to cluster $\mathcal{C}_{\hat{j}}$ with probability:

$$\exp\left(-\frac{|d_{i\hat{j}} - d_{ij}|}{t_r}\right) \tag{13}$$

where $d_{i\hat{j}}$ and d_{ij} represent the distances of $\mathbf{x}_i$ to the centers of clusters $\mathcal{C}_{\hat{j}}$ and $\mathcal{C}_j$, respectively. The SAKM algorithm is advantageous over other SA based clustering algorithms because the same set of parameter values can be used on a myriad of data sets. The temperature values are a non-increasing sequence such that $t_0 \geq t_1 \geq \cdots \geq t_r = 0$. While the use of a logarithmic cooling schedule of the form $t_r = t_0 / \log(1 + r)$ is optimal (Geman and Geman 1984), the SAKM uses a faster geometric schedule, which is identical to that of B & H. The values of t_0 and α are recommended to be set to 100 and 0.95, respectively, and these values are applicable to every data set given in Section 4.

The last related algorithm presented is the SACM algorithm (Boguś, Massone, and Masulli 1999; 2002). In contrast to the aforementioned SA algorithms, the SACM algorithm generates a completely random solution at every step. This means that the new solution is not related to the current solution in any way. The reason that this algorithm is listed as related work is because of the idea behind the SACM perturbation method. Most SA algorithms will perturb the current solution by transferring points in some fashion to other clusters. SACM perturbs the current solution by generating new centers, and then assigning each point to the cluster with the closest center. Due to the unstructured nature of its perturbation method, SACM is not a "true" SA algorithm, but the underlying idea is worth experimenting with. Since the SACM algorithm is not compared against the other methods discussed, its cooling schedule parameters will not be detailed. It does follow a very standard SA format in both its main structure and cooling schedule.

3 Proposed Algorithm

The proposed algorithm is an SA clustering algorithm based on center perturbation using Gaussian mutation, hereafter referred to as the SAGM algorithm. The basic structure of the SAGM algorithm is the same as the standard SA algorithm given in Algo. 1.

Where SAGM differs, like most others, is in how it perturbs the current solution to generate a neighbor solution. In most SA based clustering algorithms, the points are being transferred from one cluster to another based on some perturbation criteria. The SAGM algorithm perturbs the current solution by altering a randomly chosen center in a controlled manner. As opposed to SACM's completely random generation of new centers, the use of Gaussian mutation to slightly alter an existing center complies with the neighborhood structure that is required by a true SA algorithm.

Before applying Gaussian mutation, the attributes of the data set are normalized to the $[0, 1]$ interval using linear scaling (min-max normalization) (Milligan and Cooper 1988). This normalization allows the same mutation factor to be applied to each attribute regardless of their original range. After normalization, Gaussian mutation is applied, that is a normally distributed random value is added to each attribute of the target center. After the perturbation, all points are then assigned to the nearest cluster center. If the new solution has a lower Sum of Squared Error (SSE), than the current solution, then it is accepted with a probability of 1, otherwise it is accepted according to the Metropolis criterion. The pseudocode for the perturbation operator of the SAGM algorithm is given in Algo. 3. The other main component of the SAGM algorithm is the cooling schedule. As discussed in Section 1, an SA algorithm can implement one of two different types of cooling schedules: the multi Markov chain (MMC) cooling schedule and the single Markov chain (SMC) cooling schedule. The proposed SAGM algorithm is implemented using both schedules separately.

Algorithm 3 Perturbation Operator for SAGM

```
1: Let $\mathcal{C}$ be the set of clusters in the current solution.
2: Let c be a center from a randomly chosen cluster from
   the set $\mathcal{C}$.
3: Let $\delta$ be a mutation factor that is close to zero.
4: Let $G$ be a Gaussian random number generator.
5: for Each Attribute $i$ of c do
6:     Set $i = i + \delta G$;
7: end for
8: Reassign all points to the closest cluster whose center
   measures the lowest squared Euclidean distance.
9: Calculate the new SSE.
```

The MMC SAGM algorithm cooling schedule involves the following parameters:

- Markov chain length: number of perturbations at a given temperature
- t_0: initial temperature
- $t_{r+1} = f(t_r)$: temperature function
- t_f: final temperature

In contrast, the SMC SAGM algorithm cooling schedule involves only two parameters: the Markov chain length and t_0.

The temperature function for the MMC cooling schedule is the same as that of SAKM. The basic structures of the SMC and MMC SAGM algorithms differ due to their different cooling schedules. The pseudocode for the SMC procedure is given in Algo. 3.

4 Experimental Results

In this section, experimental results are provided for both variants of the SAGM algorithm as well as the algorithms

Algorithm 4 Modified SA Algorithm with Single Markov Chain Cooling Schedule

```
1: Choose the number of iterations $R$.
2: Choose acceptance temperature $t_0$.
3: Choose initial random configuration of clusters.
4: Let $r = 0$.
5: repeat
6:     perturb configuration $i$ with cost $C_i$ to configuration $j$ with cost $C_j$,
7:     if $(\Delta C_{ij} = C_j - C_i) \leq 0$ then
8:         accept configuration $j$
9:     else
10:        accept configuration $j$ with probability $\exp(-\Delta C_{ij}/t_0)$
11:    end if
12:    $r = r + 1$
13: until $r \geq R$
```

discussed in Section 2. Ten data sets from the UCI Machine Learning Repository (Frank and Asuncion 2013) were used. The data set descriptions are given in Table 1. Note that despite the fact that each algorithm has its own set of cooling schedule parameters, these parameter values were kept unchanged across the data sets. In other words, no algorithm was given unfair advantage by tuning its parameters. For the MMC SAGM algorithm, the initial temperature, final temperature, α, and number of iterations were set to 9, 4, 0.95, and $4n$ (n: # points), respectively. As for the SMC SAGM algorithm, the acceptance temperature and number of iterations were set to 9 and $10n$, respectively. The experiments were conducted on an Intel Core$^{\text{TM}}$ i7-930 2.80GHz machine.

In order to ensure fairness, every algorithm was initialized with the same set of randomly generated initial cluster centers. Tables 2 and 3 give the final SSE and CPU time measurements of the algorithms, respectively.

It can be seen that both SAGM algorithms provide solutions comparable to the other SA algorithms, but the SMC SAGM algorithm generally converges significantly faster than the other algorithms. The SMC SAGM method outperformed every other method on the Yeast data set. The only other SA algorithm that rivals the speed of SMC SAGM is the SAKM algorithm.

The rapid cooling schedule of SMC SAGM makes it preferable to MMC SAGM, because the overall runtime is reduced while converging to a comparable quality solution. This is why the initial and final temperatures are so close in the MMC SAGM algorithm.

The SAGM algorithm is not without its drawbacks. First, the quality of the final solution is dependent on the initial cluster centers. This "cluster center initialization" problem is common to many partitional clustering algorithms and can be addressed using a variety of methods, see for example (Celebi and Kingravi 2012; Celebi, Kingravi, and Vela 2013). Second, as the cluster centers themselves are being perturbed, if an outlier is given as an initial center, it will be very difficult to adjust this center via Gaussian mutation. This problem can be addressed by using either outlier pruning (Zhang and Leung 2003) or an outlier insensitive initialization method such as the one described in (Al Hasan et al. 2009).

Table 1: Descriptions of the Data Sets (n: # points, d: # attributes, k: # classes)

ID	Data Set	n	d	k
1	Ecoli	336	7	8
2	Glass	214	9	6
3	Ionosphere	351	34	2
4	Iris Bezdek	150	4	3
5	Landsat	6,435	36	6
6	Letter Recognition	20,000	16	26
7	Image Segmentation	2,310	19	7
8	Vehicle Silhouettes	846	18	4
9	Wine Quality	6,497	11	7
10	Yeast	1,484	8	10

Table 2: Final SSE for Each SA Clustering Algorithm

ID	K & D	SAKM	B & H	SAGM (MMC)	SAGM (SMC)
1	17.472	17.436	17.437	17.449	$\underline{17.435}$
2	18.260	$\underline{18.241}$	$\underline{18.241}$	18.246	$\underline{18.241}$
3	$\underline{628.89}$	$\underline{628.89}$	$\underline{628.89}$	628.93	628.90
4	$\underline{6.9822}$	$\underline{6.9822}$	$\underline{6.9822}$	$\underline{6.9822}$	$\underline{6.9822}$
5	1742.62	$\underline{1741.59}$	$\underline{1741.59}$	1742.84	1744.61
6	4104.35	$\underline{2747.18}$	2756.86	2767.12	2763.35
7	387.168	$\underline{386.974}$	$\underline{386.974}$	387.540	388.084
8	223.537	$\underline{223.494}$	$\underline{223.494}$	223.986	223.952
9	48.970	$\underline{48.954}$	$\underline{48.954}$	$\underline{48.954}$	49.001
10	69.110	69.011	68.854	64.188	$\underline{59.168}$

Table 3: CPU Time for Each SA Clustering Algorithm ('m': minute, 's': second)

ID	K & D	SAKM	B & H	SAGM (MMC)	SAGM (SMC)
1	15s	3s	3s	$\underline{\text{1s}}$	$\underline{\text{1s}}$
2	4s	2s	1s	$\underline{\text{0s}}$	$\underline{\text{0s}}$
3	20s	7s	7s	4s	$\underline{\text{1s}}$
4	2s	1s	$\underline{\text{0s}}$	$\underline{\text{0s}}$	$\underline{\text{0s}}$
5	3,247m 36s	$\underline{\text{6m35s}}$	145m57s	125m36s	21m55s
6	20,909m 17s	$\underline{\text{23m35s}}$	1,167m 23s	2,701m 19s	470m58s
7	164m1s	$\underline{\text{1m10s}}$	9m7s	6m44s	1m15s
8	7m18s	15s	45s	28s	$\underline{\text{4s}}$
9	3s	2s	1s	$\underline{\text{0s}}$	$\underline{\text{0s}}$
10	15m37s	22s	1m50s	1m24s	$\underline{\text{15s}}$

5 Conclusions

This paper presented a new SA based partitional clustering algorithm called SAGM. This algorithm differs from most existing SA based partitional clustering algorithms in how it perturbs the current solution to generate a neighborhood solution. Instead of swapping data points between clusters, the SAGM algorithm uses Gaussian mutation to perturb a randomly chosen cluster center in a controlled manner. This new method of generating a neighborhood solution leads to dramatically reduced run-times while maintaining convergence to high quality clustering solutions.

Robust clustering ability, simplicity, and speed combined help make the SAGM algorithm a very attractive alternative to the state-of-the-art SA clustering algorithms. Although the algorithm does have some drawbacks, these can be overcome and similar problems exist with other algorithms as well.

6 Acknowledgments

This publication was made possible by a grant from the National Science Foundation (1117457). The authors are grateful to the anonymous reviewers for their insightful suggestions and constructive comments that improved the quality and presentation of this paper.

References

Aarts, E., and van Laarhoven, P. 1989. Simulated Annealing: An Introduction. *Statistica Neerlandica* 43(1):31–52.

Al Hasan, M.; Chaoji, V.; Salem, S.; and Zaki, M. 2009. Robust Partitional Clustering by Outlier and Density Insensitive Seeding. *Pattern Recognition Letters* 30(11):994–1002.

Aloise, D.; Deshpande, A.; Hansen, P.; and Popat, P. 2009. NP-Hardness of Euclidean Sum-of-Squares Clustering. *Machine Learning* 75(2):245–248.

Bandyopadhyay, S.; Maulik, U.; and Pakhira, M. K. 2001. Clustering Using Simulated Annealing with Probabilistic Redistribution. *International Journal of Pattern Recognition and Artificial Intelligence* 15(2):269–285.

Boguś, P.; Massone, A. M.; and Masulli, F. 1999. A Simulated Annealing C-Means Clustering Algorithm. In *Proceedings of the 3rd ICSC Symposia on Intelligent Industrial Automation and Soft Computing*.

Boguś, P.; Massone, A. M.; and Masulli, F. 2002. Simulated Annealing C-means Clustering Algorithm Convergence Proof. *Proceedings of the 6th International Conference on Neural Network and Soft Computing* 590–595.

Brown, D. E., and Huntley, C. L. 1992. A Practical Application of Simulated Annealing to Clustering. *Pattern Recognition* 25(4):401–412.

Brown, D. E., and Huntley, C. L. 1995. Fundamentals of Cluster Analysis Using Simulated Annealing. In Kalivas, J. H., ed., *Adaption of Simulated Annealing to Chemical Optimization Problems*. Elsevier. 133–154.

Celebi, M. E., and Kingravi, H. 2012. Deterministic Initialization of the K-Means Algorithm Using Hierarchical Clustering. *International Journal of Pattern Recognition and Artificial Intelligence* 26(7):1250018.

Celebi, M. E.; Kingravi, H.; and Vela, P. A. 2013. A Comparative Study of Efficient Initialization Methods for the K-Means Clustering Algorithm. *Expert Systems with Applications* 40(1):200–210.

Dowsland, K. A., and Thompson, J. M. 2012. Simulated Annealing. In Rozenberg, G.; Bäck, T.; and Kok, J., eds., *Handbook of Natural Computing*. Springer. 1623–1655.

Frank, A., and Asuncion, A. 2013. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml. University of California, Irvine, School of Information and Computer Sciences.

Geman, S., and Geman, D. 1984. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6(6):721–741.

Jain, A. K.; Murty, M. N.; and Flynn, P. J. 1999. Data Clustering: A Review. *ACM Computing Surveys* 31(3):264–323.

Klein, R. W., and Dubes, R. C. 1989. Experiments in Projection and Clustering by Simulated Annealing. *Pattern Recognition* 22(2):213–220.

Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N.; Teller, A. H.; and Teller, E. 1953. Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics* 21(6):1087–1092.

Milligan, G., and Cooper, M. C. 1988. A Study of Standardization of Variables in Cluster Analysis. *Journal of Classification* 5(2):181–204.

Rutenbar, R. A. 1989. Simulated Annealing Algorithms: An Overview. *IEEE Circuits and Devices Magazine* 5(1):19–26.

Selim, S. Z., and Alsultan, K. 1991. A Simulated Annealing Algorithm for the Clustering Problem. *Pattern Recognition* 24(10):1003–1008.

Zhang, J. S., and Leung, Y.-W. 2003. Robust Clustering by Pruning Outliers. *IEEE Trans. on Systems, Man, and Cybernetics – Part B* 33(6):983–999.