Pattern Recognition 44 (2011) 2786–2799

Contents lists available at ScienceDirect

Pattern Recognition

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

A general stochastic clustering method for automatic cluster discovery

Swee Chuan Tan [a,*], Kai Ming Ting [b], Shyh Wei Teng [b]

[a] *SIM University, 461 Clementi Road, Singapore*
[b] *Monash University, Gippsland School of Information Technology, Australia*

ARTICLE INFO

Article history:
Received 26 March 2010
Received in revised form
1 April 2011
Accepted 2 April 2011
Available online 15 April 2011

Keywords:
Clustering
Ant-based clustering
Automatic cluster detection

ABSTRACT

Finding clusters in data is a challenging problem. Given a dataset, we usually do not know the number of natural clusters hidden in the dataset. The problem is exacerbated when there is little or no additional information except the data itself. This paper proposes a general stochastic clustering method that is a simplification of nature-inspired ant-based clustering approach. It begins with a basic solution and then performs stochastic search to incrementally improve the solution until the underlying clusters emerge, resulting in automatic cluster discovery in datasets. This method differs from several recent methods in that it does not require users to input the number of clusters and it makes no explicit assumption about the underlying distribution of a dataset. Our experimental results show that the proposed method performs better than several existing methods in terms of clustering accuracy and efficiency in majority of the datasets used in this study. Our theoretical analysis shows that the proposed method has linear time and space complexities, and our empirical study shows that it can accurately and efficiently discover clusters in large datasets in which many existing methods fail to run.

1. Introduction

Cluster analysis involves grouping data objects into clusters, having similar objects within a cluster that maximises inter-cluster differences [21]. Many approaches have been proposed over the years. Traditional clustering methods, such as *k*-means [29] and hierarchical clustering approach [24], are well-known. One key problem with many traditional clustering methods is that they require users to input the number of clusters or assume some kind of probability distribution underlying a dataset. This requirement is not practical because users may not have such information prior to cluster analysis.

Recently, a number of nature-inspired data clustering algorithms, namely swarm-based clustering (SBC) methods, have been proposed in the literature [28,32,19,54,4]. The main inspiration of swarm-based approach is to mimic the incredible ability of simple social insects (ants, bees, termites, etc.) to solve complex problems collectively. For example, it has been observed by entomologists that ants are able to construct complicated nests, perform brood-sorting and form cemeteries for dead ants. Although swarm-based systems are made up of simple and autonomous agents, such systems are robust and adaptive to environmental changes, and usually exhibit sophisticated group behaviour [3]. Some interesting examples include clustering algorithms inspired by: (i) the cemetery formation of ants [28,26,32,38,19,54]; (ii) the flocking behaviour of animals [4,12,11]; (iii) the chemical recognition of nest-mates in ant colony [27]; and (iv) aggregated clustering approaches using multi-ant colonies [54]. A comprehensive recent survey is also available [20].

Unlike traditional methods, SBC has a remarkable property—it is able to automatically discover the number of clusters underlying a dataset without requiring users to specify the number of clusters [19]. Therefore, the swarm-based approach is more suitable for data cluster analysis of real-world problems when little or no additional information is known about a dataset.

Like many new clustering methods, SBC has its own weaknesses. For example, it is usually hard to find effective agent behaviours that will improve the overall performance of an SBC algorithm. Furthermore, the algorithmic complexity is hard to derive by theoretical analysis. Recently, Handl et al. have suggested that it may be possible to improve the existing ABC methods using a stochastic algorithm that employs only the key heuristics underlying these systems [19]. This paper is an attempt along this avenue of research.

Here, we propose a general stochastic clustering method (GSCM) that is able to perform automatic cluster discovery. By 'general' we mean there are different ways to extend our basic method presented in this paper. This paper will present a simple yet effective adaptation of the basic method for clustering *real-valued multivariate data*.

Like many SBC methods, our approach employs a stochastic search to incrementally improve a clustering solution until a terminating condition is reached. Unlike SBC, our approach

* Corresponding author.
E-mail address: jamestansc@unisim.edu.sg (S. Chuan Tan).

0031-3203/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2011.04.001

does not employ multiple agents in its formalism. Hence, neither GSCM does employ a colony of ants found in ABC [28], nor does it simulate the behaviours of a flock of flying birds in flocking-based clustering method [4]. As a result, the proposed method is simpler, its computational complexity can be analysed easily, and it retains the power of SBC in automatic cluster discovery.

The structure of this paper is as follows. We first provide a review of some methods related to this work. We then introduce GSCM and describe how GSCM can be adapted to deal with real-world clustering problems. In our experiments, we show that the proposed method performs competitively against three existing methods in terms of clustering accuracy and runtime. Finally, we provide some discussions and conclude this paper.

2. Related works

This section provides a review of some ABC methods that have influenced this work, as well as three other methods that also perform automatic cluster discovery.

2.1. Ant-based clustering methods

Presumably trying to clean their nests, certain species of ants have been found to form cemeteries in their colonies. In the cemetery formation process, ants communicate with each other indirectly via the distribution of dead ants within their nest. Each ant works autonomously and uses only local information (i.e., the proportion of dead ants that it has recently seen). There is no supervisor, or plan, yet the ants appear to work as a team to build the cemeteries.

Inspired by this observation, the computational method, called ant-based clustering (ABC), was originally introduced with ant-like robots that can perform sorting of physical objects [8]. Initially, objects and ant-like agents are scattered over a two-dimensional (2D) space at random locations (c.f. Fig. 1a). These autonomous ant-like agents explore their environment with random movements, and each of them operates with simple rules—an unloaded agent would probabilistically pick up an isolated object and transport it with random movements; a loaded agent would probabilistically drop its object if it comes across an area with many other objects. When the agents repeatedly perform these simple actions, groups of items are gradually created (c.f. Fig. 1b).

2.2. The Lumer and Faieta's model

To extend Deneubourg's model [8] for numerical data analysis, Lumer and Faieta [28] proposed a neighbourhood function to compute the average similarity between an item i and each item (j) in its surroundings:

$$f(i) = \max\left(0.0, \frac{1}{\sigma^2}\sum_j \left(1 - \frac{\delta(i,j)}{\alpha}\right)\right), \tag{1}$$

where α is a distance threshold parameter for the distance measure $\delta(i,j)$ between a currently picked data item i and all the data item j in the neighbourhood of i. $\sigma^2 = (2r+1)^2$ is the size of the ant's perception region (or neighbourhood) for a perceptive radius (r). σ^2 is typically 9 or 25. Intuitively, a high $f(i)$ means that i is surrounded by many similar items, otherwise $f(i)$ is low.

Note that α is a critical and sensitive parameter in ABC. If α is too high, many items will be deemed similar and ABC will return fewer clusters than expected. If α is too low, many items will be deemed dissimilar and ABC will return more clusters than expected.

The probabilities of picking up and dropping an item i are shown in the following two equations respectively:

$$p_{pick}(i) = \left(\frac{k^+}{k^+ + f(i)}\right)^2; \tag{2}$$

and

$$p_{drop}(i) = \begin{cases} 2f(i), & \text{if } f(i) < k^-, \\ 1, & \text{otherwise.} \end{cases} \tag{3}$$

Here, k^+ and k^- are parameters between 0 and 1. These parameters decide how influential $f(i)$ is on $p_{pick}(i)$ and $p_{drop}(i)$. An unloaded ant is likely to pick up an item i if $f(i)$ is small with respect to k^+; this means that the item is either isolated or is surrounded by many dissimilar items. Similarly, a loaded ant is likely to drop an item i if $f(i)$ is large with respect to k^-; this means that the ant encounters many similar items. Consequently, a positive feedback mechanism is developed and it promotes the formation of clusters on the 2D space.

The basic Lumer and Faieta's algorithm, which is partly adapted from the one presented in Handl's thesis [17], is presented in Algorithm 1. Initially, the items are randomly scattered on a 2D grid and each agent is loaded with an item (e.g., Fig. 1a). When the main loop starts, each ant moves randomly and begins to pick up and drop items. The random function $Rand[0, 1]$ generates a random

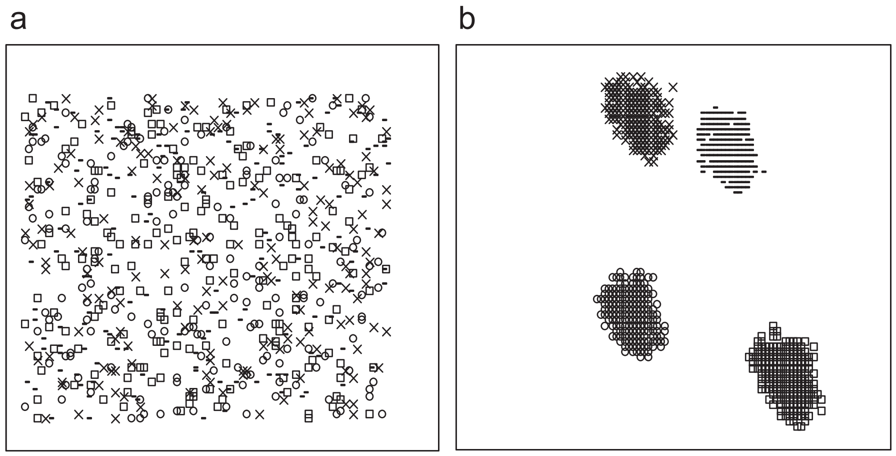

Fig. 1. An example of (a) an initial random solution and (b) four clusters that emerge as a result of ant-based clustering.

value in [0, 1]. At the end of the process, a set of clusters are created on the 2D grid (e.g., Fig. 1b).

Algorithm 1. Basic ant-based clustering.

```
INITIALIZATION
Randomly scatter $n$ items on a 2D toroidal grid
Let $G$ be a population of agents
Each agent in $G$ is randomly assigned (or loaded with) an item
MAIN LOOP
for $iteration = 1$ to $maxIteration$ do
  $g :=$ an agent randomly selected from $G$
  Let the item carried by $g$ be $i$
  $g$ performs a random move on the grid
  if $p_{drop}(i) > Rand[0,1]$ then
    $g$ drops item $i$ at its current location
    $PICK := false$
    while $\neg PICK$ do
    $g$ moves on the grid randomly until it encounters an
    item $q$
    if $p_{pick}(q) > Rand[0,1]$ then
    $PICK := true$
    $g$ is loaded with $q$
    end if
    end while
  end if
end for
```

The Lumer and Faieta's model, has been extended in many subsequent works, many of which produce competitive clustering performance compared to traditional methods [32,17]. Despite this fact, it remains unclear why the method works. To answer this question, Martin et al. [31] proposed a minimal ant-based clustering model (MABC) to study the clustering dynamics of ABC. This model is a highly simplified theoretical version of ABC. In MABC, the 'intelligence' of ants is reduced to a minimum.

The MABC model was tested on a simplistic problem: clustering multiple items of the same type (i.e., one cluster). Initially, the ants move randomly on a 2D toroidal grid where items are randomly scattered. The random movements of the ants and their pick and drop actions lead to random transportation of items from one part of the grid to another. This causes the sizes of the clusters on the grid to fluctuate. Because the ants have a higher chance of encountering big clusters (which occupy larger grid areas) than small ones, when an item is removed from a cluster of any size, the item is more likely to be transported to a big cluster. Hence, big clusters tend to grow and small clusters tend to deplete. A cluster disappears when it becomes empty (i.e., reach a threshold of null size). At the end of the process, only one large cluster remains. The authors called this phenomenon the *fluctuation threshold effect* [31].

One interesting implication of MABC is that there is no collective effect in the clustering process because one ant is able to produce a single cluster—the same as using multiple ants. Intuitively, the items converge into one cluster because of the fluctuation threshold effect. This observation is also confirmed by Gaubert et al. [13], who showed that the items will converge into one cluster *almost surely*. Note that MABC is a simple theoretical model, so it has never been designed or extended to deal with real-world clustering problems. In the following, we will consider a state-of-the-art ABC model that has been developed for clustering real-valued data.

ATTA (adaptive time dependent transporter) is a recent significant improvement over the Lumer and Faieta's model [28]. Proposed by Handl et al. [19], it is a highly cited method which outperforms several traditional clustering methods [17].

One major contribution of ATTA is the automatic estimation method for the critical distance threshold parameter (α) used in the neighbourhood function (e.g., Eq. (1)). During the clustering process, if an ant encounters many *similar* items, this implies that its distance threshold is too loose (i.e., α is too high), and the ant will adjust its α down to tighten the threshold. If an ant encounters many *dissimilar* items, this means that the distance threshold is too tight (i.e., α is too low) and the ant needs to adjust its α up to loosen the threshold. These adjustments are made at regular intervals throughout the entire clustering process. As a result, an automatic determination of distance threshold, which leads to automatic cluster discovery, is achieved.

To demonstrate the automatic cluster discovery capability, Handl et al. [17] showed that ATTA significantly outperforms the state-of-the-art gap statistic [49] (used in conjunction with k-means) with 22 wins and three losses, over 25 datasets. In addition, ATTA was also shown to outperform three traditional clustering methods [17,19], k-means, hierarchical agglomerative clustering with average linkage metric (AvgLink) and self-organizing map (SOM) [25].

2.3. Other methods for automatic cluster discovery

This subsection reviews three existing methods for automatic cluster discovery. The first method is cluster validation. The second method is an expectation maximisation clustering method [53]. The third method is x-means [36], which is an extension of k-means that uses Bayesian information criterion (BIC statistic) [40] to choose the best model.

2.3.1. Cluster validation methods

One of the attempts to address the shortcomings of existing methods is to use a cluster validity index to evaluate a clustering solution. Some examples of popular cluster validity indexes include: (i) Shiluette width [39]; (ii) Dunn index [9]; and (iii) Davies–Bouldin index [6].

One important use of cluster validation is in automatically finding the number of clusters in a dataset. This process involves generating a number of clustering solutions, and then to evaluate each solution using a cluster validity index. The best solution is the one associated with the best validity index. This brute-force approach [23] is known to be computationally expensive. In addition, many cluster validity indexes are known to work only when their assumptions about the cluster structures are not violated. In practice, it is difficult to choose the right index for a particular problem.

2.3.2. Expectation maximisation clustering (EMC)

Given a dataset that contains a set of clusters; supposed each cluster is generated from a Gaussian distribution with unknown parameters [30]. The goal of EMC is to use the maximum likelihood approach [7] to estimate the parameters in each cluster so that each of the estimated clusters best fits an observed distribution of values.

EMC starts with a solution based on a minimum number of clusters specified by the user. It then incrementally steps up the number of clusters, and evaluates the solution for each possible number of clusters using *v-fold cross-validation*. The average likelihood for the data in the testing samples is computed. Using the optimum result, EMC automatically stops at the most probable cluster structure in a dataset.

EMC is known to have problem converging to the correct clustering especially when its assumptions (e.g., Gaussian distributions of items in clusters) are violated. Its runtime performance is also an issue when data size is large. Its clustering performance is also known to be sensitive to the kind of pre-processing treatment done on the data [22].

2.3.3. x-Means

Another method that automatically derives the number of clusters is the widely cited *x*-means method introduced by Pelleg and Moore [36]. The method extends *k*-means by an *improve-structure* mechanism which attempts to split each cluster centre into two if a better fit of data can be achieved. The decision to split a centre is based on the BIC statistic [40]. Hence *x*-means chooses the model that has the best BIC statistic. Pelleg and Moore have demonstrated that *x*-means is more efficient than the original *k*-means algorithm with the additional advantage of deriving the number of clusters based on a lower bound and an upper bound of *the number of clusters* provided by users.

3. General stochastic clustering method

GSCM is a conceptual framework that consists of two key components: (i) workspace is the space in which data items are placed and moved around during the clustering process, and (ii) elementary operations perform the clustering tasks of random item selections, item comparisons and move operations.

Algorithm 2 presents the GSCM conceptual framework. The main loop consists of a two-step procedure: (i) randomly select two items *i* and *j* from the dataset (*D*) and (ii) moves item *i* to the location of item *j* if *i* and *j* are similar. This procedure is repeated until a termination condition is reached.

Algorithm 2. GSCM.

```
Input: a dataset of $n$ items, $D$
Output: a set of clusters
Initialise Workspace
repeat
  $i := \text{randomSelect}(D)$
  $j := \text{randomSelect}(D)$, where $i \neq j$
  if $similar(i, j)$ then
    $move(i, j)$ // move $i$ to location of $j$
  end if
until termination condition is reached
Return a set of clusters
```

GSCM can be considered as a stochastic two-step procedure that aims to improve the clustering results by modifying the solution locally and incrementally. Each modification involves searching for two similar items and grouping them together. As a result, groups of similar items (i.e., clusters) emerge as the two-step procedure is repeated many times.

At this point, GSCM does not assume any specific kind of workspace. The workspace may be implemented as a two-dimensional (2D) grid, or simply be a set of item bins. The elementary operations are procedures that group the items within the workspace. Hence, GSCM is a conceptual model that can be adapted in different ways.

3.1. Simple GSCM

To demonstrate the working principle of GSCM, we adapt the conceptual GSCM to simple GSCM (*s* GSCM) and then perform some simple clustering tasks that were previously studied using minimal ant-based clustering (MABC) [31].

s GSCM is presented in Algorithm 3. We define the workspace as a set of item bins, where each bin initially stores one item. As in [31], we begin by considering a simple problem: the clustering of items of the same type (i.e., all items belong to one cluster). In a strict sense, this is not a 'problem', rather, it is an effective toy example to examine how a *s* GSCM model works. The similarity measure is trivial because there is no distinction between items.

Given a dataset *D* with *n* items, we begin by allocating each item to one bin. Then, we repeat the main loop of *s* GSCM for a fixed number of iterations. In each loop, two different items *i* and *j* are randomly selected from *D*. Let *i* be an item from a source bin, and *j* be an item from a destination bin. At the end of each iteration, *i* is always assigned to the bin of *j*.

Algorithm 3. Simple GSCM.

```
Input: a dataset of $n$ items, $D$
Output: a set of clusters
Initialise $n$ bins by allocating each item in $D$ to a bin
for $iteration = 1$ to $maxIteration$ do
  $i := \text{randomSelect}(D)$
  $j := \text{randomSelect}(D)$, where $i \neq j$
  $move(i, j)$
end for
Return all non-empty bins as a set of final clusters
```

Initially, each bin is randomly allocated with one item. In subsequent iterations, the random assignments of items among the bins cause the bin sizes to fluctuate, resulting in some bigger and some smaller bins. Once an item is selected during an iteration, the probability of adding this item to a bin b is $|b|/n$, where n is the total number of items and $|b|$ is the number of items in bin b. If $|b|$ is larger than those in other bins, then bin b will tend to grow; if $|b|$ is smaller than other bins, then bin b will tend to vanish. Hence, the algorithm is biased towards forming a single large bin (i.e., cluster).

To cluster two or more distinct types of items, Algorithm 3 can be modified by replacing the statement *move*(*i*, *j*) with:

```
if type($i$) equals type($j$) then
  $move(i, j)$
end if
```

3.2. Speeding up the convergence for s GSCM

We can improve the convergence rate of *s* GSCM using a heuristic. Let $c(x)$ be a function that computes the *level of support* for item x in its currently assigned bin. If $c(x)$ is low, then x has a low level of support from similar items within the bin it is currently in; thus it shall be moved to another bin. If $c(x)$ is high, then x has a high level of support from similar items in its bin; so it shall be retained in its current bin. More specifically, whenever two separate items i and j are compared and deemed similar, the heuristic is to move i into j's bin, if $c(i) < c(j)$; or to move j into i's bin, if $c(i) \geq c(j)$.

Formally, the level of support for an item x is defined as:

Definition 1. *Level of Support:* Let S_x be a set of *all* items similar to item x in D, excluding x. Let X be the set of items within x's bin. Then, $c(x) = |A_x|/|S_x|$, where $A_x = S_x \cap X$.

Literally, $|A_x|$ is the number of items similar to x *within its current bin*, excluding x. Let $B_x = S_x \setminus A_x$, then $|B_x|$ is the number of items similar to x *outside of its current bin*. If $|A_x| \gg |B_x|$, then $c(x)$ is close to one, indicating that x has a high level of support in its current bin and x is likely to attract similar items from other bins. On the other hand, if $|A_x| \ll |B_x|$, then $c(x)$ is close to zero, indicating that x has a low level of support and x is likely to move to other bins which contain similar items.

To compute $c(x)$, item x must be compared with all the other items in the dataset. For n items, the time complexity to compute the levels of support for all items becomes $O(n^2)$, which is expensive. Fortunately, $c(\cdot)$ can be estimated during the clustering process. Recall that *s* GSCM involves a series of comparisons where each comparison involves two separate items that are randomly selected, the idea is to store the outcomes of the last *b recorded* comparisons made with each item, and estimate $c(\cdot)$ based on these outcomes.

The outcomes are stored using a first-in-first-out buffer, denoted as V_x. A comparison is *recorded* in V_x if x is compared to another item y that is deemed similar to x. V_x has a fixed size of b, in which each element stores one bit, representing one of the two possible outcomes: (i) when y is in the same bin as x, '1' is recorded; or (ii) when y is not in the same bin as x, '0' is recorded. Finally, $c(x)$ at current time t is estimated as:

$$\hat{c}_t(x) = \frac{1}{b}\sum_{k=0}^{b-1} V_x[t-k], \tag{4}$$

where $V_x[t-k]$ is the comparison outcome recorded at a time that is k time steps before the current time t. Hence, $\hat{c}_t(x)$ is the proportion of ones in V_x at time t.

After the first b comparisons, $\hat{c}_t(x)$ can be computed without going through all the b elements in the buffer. This is because $\hat{c}_t(x)$ can be written as:

$$\hat{c}_t(x) = \hat{c}_{t-1}(x) + \frac{V_x[t] - V_x[t-b]}{b}.$$

Hence, $\hat{c}_t(x)$ can be updated in constant time regardless of the buffer size b.

The above mechanism can be incorporated in *s* GSCM (Algorithm 3) by replacing the statement *move* (i, j) with the following procedure:

```
if $acceptance(i, j)$ then
  store the comparison outcome in $V_i$ and $V_j$
  $(\hat{c}(i) < \hat{c}(j))?move(i,j) : move(j,i)$
end if
```

Here, *acceptance* (i, j) simply checks if type (i) equals type (j).

It is important to ensure that the selection of items for comparisons in *s* GSCM is purely random, so that $\hat{c}(x)$ is a point estimate of the proportion $c(x)$, based on a simple random sample of size b. This estimation eliminates the need to compute $c(x)$ using the entire dataset.

Based on the Wilson's estimate for binomial proportion [52], and by using the most conservative value of 0.5 for the unknown proportion, the sample size derived by Piegorsch [37] is given as:

$$b = \frac{1}{4}\left(\frac{z}{m}\right)^2 - z^2,$$

where m is the margin of error required for the estimation. As we only require a rough estimation of $c(\cdot)$ for our clustering heuristic, we set $m=0.15$. We use $z=1.65$, which corresponds to a 90% confidence interval estimation. Hence, the sample size b is 27.5 and then rounded up to 30.

3.3. Practical GSCM

In the previous subsection, we show the basic properties of *s* GSCM using some toy problem. Here, we show that *s* GSCM can be further adapted to cluster multivariate real-world data. We call this algorithm practical GSCM (*p* GSCM) as shown in Algorithm 4.

Algorithm 4. Practical GSCM.

```
Input: dataset of $n$ items, $D$
Output: a set of clusters
Estimate the dissimilarity thresholds for $n$ items
Initialise $n$ bins by allocating each item in $D$ to a bin
for $iteration = 1$ to $maxIteration$ do
  $i :=$ randomSelect($D$)
  $j :=$ randomSelect($D$), where $i \neq j$
  if $acceptance(i, j)$ then
    Store the comparison outcome in $V_i$ and $V_j$
    $(\hat{c}(i) < \hat{c}(j))?move(i,j) : move(j,i)$
  end if
end for
Return all non-empty bins as a set of final clusters
```

Despite the fact that *p* GSCM is required to handle real-world clustering problems, surprisingly few changes are required in order to change from *s* GSCM to *p* GSCM. By using the *s* GSCM algorithm with the convergence heuristic, the only additional change required is to use an *acceptance* predicate that is able to compare numerical data. A common *acceptance* predicate that has been used in SBC's literature (e.g., [27]) is, $acceptance(i,j) = sim(i,j) > max(S_i,S_j)$, where $sim(i,j) = 1-\delta(i,j)$ is the similarity between i and j, $\delta(i,j)$ is the *normalised* distance scaled to [0, 1] and S_x is the similarity threshold for x. Here, we avoid the need to compute normalised distance by using unnormalised distance directly in the following equation:

$$acceptance(i,j) = d(i,j) < min(T_i,T_j), \tag{5}$$

where $d(i,j)$ is an (unnormalised) 1-norm distance metric between i and j. The dissimilarity threshold (denoted as T_x) for item x is defined as:

$$T_x = \omega \cdot meanDist(x,\cdot) + (1-\omega) \cdot minDist(x,\cdot), \tag{6}$$

where *meanDist* and *minDist* are respectively the mean and minimum distances between an item x and all the other items in D, and ω controls the weights of *meanDist* and *minDist*. Intuitively, if ω is high, more weight is allocated to *meanDist* than *minDist*, resulting in high T_x and the clustering process may generate fewer clusters than expected. On the other hand, a low ω causes a reverse effect, and the clustering process may generate more clusters than expected. Our experience indicates that a sensible range of ω is [0.2, 0.4], for it gives more weights to *minDist* and tends to create cohesive clusters. We use a fixed setting of $\omega = 0.3$ for the 44 datasets used in our experiments.

Note that each data item has its own unique dissimilarity threshold. Let p be an item of a dense cluster (C_p) and q be an item of a sparse cluster (C_q). Intuitively, $T_p < T_q$. This means that p will probably not be deemed similar to q because it is likely that $d(q,\cdot) > T_p$. This happens even when C_p and C_q are close to each other. This property is important because it allows *p* GSCM to detect clusters of different densities, which is one of the hardest problems in cluster analysis [10].

In Eq. (6), *meanDist* and *minDist* involve distance evaluation between an item and each of all the other items in the dataset. This means that the cost for computing the thresholds of n items is $O(n^2)$. We address this problem by estimating the dissimilarity threshold of each data item before the clustering process [27]. Each estimation involves comparing an item with ψ other items randomly selected from the dataset. For n items, there are a total of $\psi \cdot n$ comparisons required for learning the thresholds. This learning process provides an estimation of the dissimilarity threshold for each item.

In our experiments, ψ is set at 150 for all the small and large datasets used, where the largest dataset contains 72 dimensions and 200 000 instances. The key is to ensure that the ψ instances are randomly selected (without bias) during the estimation process, so that each estimation is based on a representative sample of the dataset. For extremely large datasets, one can examine whether the ψ setting is sufficient by making sure that changes in the estimates are negligible after the estimation process. As will be shown later, increasing ψ multiple-folds does not change the clustering accuracy significantly.

The key advantage of using the acceptance predicate is that it requires only one distance computation in order to determine if two items are similar or not. Thus p GSCM is expected to run faster than other clustering methods (such as ABC and flocking-based clustering [11]) that perform multiple distance computations within each grid-neighbourhood.

At the end of the maximum number of iterations, each item that belongs to a 'small bin' is reassigned to a larger bin which has a centroid that is most similar to the item. In p GSCM, a 'small bin' is one with bin size less than $min(50,n/20)$. The threshold of $n/20$ is based on the criterion used by Fabien et al. [11]. When n is large, we limit the bin size threshold to 50.

Note that the threshold of $n/20$ is the same as assuming that the smallest clusters would be no less than 5% of the total data size n. This is reasonable because a larger threshold percentage might cause small but genuine clusters to be mistakenly removed from the final solution. As will be shown in Section 5.1, any threshold between 1% and 10% does not affect the clustering performance.

3.4. *Complexity analysis for p GSCM*

This subsection analyses about the time and space complexities of p GSCM.

3.4.1. *Time complexity of p GSCM*

In p GSCM, the initialisation involves allocating each item to a bin. For n items, the time complexity of this step is $O(n)$. As mentioned earlier, there are $150 \cdot n$ iterations required to estimate the dissimilarity thresholds for n items. Since 150 is a constant and most datasets are a lot bigger than 150, the cost of this step is $O(n)$.

In the main loop, the constant-time operations are (i) random selections of items i and j; (ii) storing the outcome of a comparison between vectors V_i and V_j; (iii) the *move* operation; and (iv) computing $\hat{c}(\cdot)$.

Apart from these constant-time operations, each iteration in the main loop involves mainly evaluating distance between two data items in d-dimensional space, which costs $O(d)$. For a fixed number of iterations $k \cdot n$ in the main loop, where k is a constant, the overall complexity for p GSCM is $O(k \cdot n \cdot d)$. Intuitively, k controls the (expected) number of times each item compares with other items during the clustering process. We set k=2000 for all the datasets used in our experiments. For small datasets, this setting does not affect the practical runtime because n is small. For large datasets that contain thousands of instances, the overall time complexity tends towards $O(n)$ when $k \ll n$ and $d \ll n$.

3.4.2. *Space complexity of p GSCM*

Note that p GSCM does not require a distance matrix. Each data item requires a buffer of b bits. Together with all the n data items to be stored in the memory, the space complexity is $O(b \cdot n)$. Since b is a small constant fixed at 30, the resulting space complexity is $O(n)$.

4. *p* GSCM versus ATTA, EMC and *x*-means

The main aim of this section is to conduct a detailed comparison of p GSCM and ATTA, which is the state-of-the-art method in SBC. To the best of our knowledge, there is no other direct competitor of p GSCM that works better than ATTA.

We also include EMC and x-means to illustrate the differences between p GSCM and other approaches. EMC is a model-based clustering method that employs the principle of maximum likelihood. x-means is a partitioning method that uses BIC statistic. All these methods represent different approaches for automatic cluster discovery.

This section is organised as follows. We first describe the experimental setup. Next, we analyse the results of the three methods on 32 datasets. These analyses will be organised into three separate subsections, where each subsection discusses the performance of the methods based on one specific measure. To highlight the practical benefits of p GSCM, we also illustrate how p GSCM can be used to cluster three large datasets. This will be presented in the final subsection.

4.1. *Experimental setup*

4.1.1. *Experimental data*

To compare the performance differences between p GSCM and each of its competitors, this study uses a variety of datasets for the experimental evaluations. We organise the data into three groups.

The first group of data contains 21 synthetic datasets previously used by Handl et al. to test ATTA against traditional methods. These datasets are the squares, sizes series which are overlapping clusters and unevenly sized clusters respectively. There is also the xDyC series of datasets, where each contains different number of dimensions (denoted by x) and different number of clusters (denoted by y). Characteristics of these datasets are shown in Fig. 2, and more details of these datasets can be found in [18,17].

The second group contains real-world datasets, which can be divided into two subgroups. The first subgroup contains seven datasets taken from the UCI machine learning repository [2].

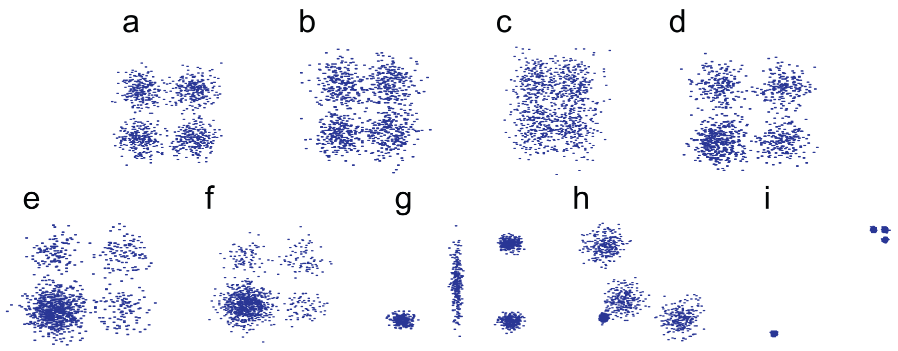

Fig. 2. Each synthetic dataset contains 1000 instances and four bivariate Gaussian clusters. Square1 (a), Square3 (b) and Square5 (c) contain increasingly overlapping clusters. Sizes1 (d), Sizes3 (e) and Sizes5 (f) contain clusters with increasingly uneven sizes. Triangle (g) contains clusters with different shapes; VaryDensity (h) contains clusters with different densities; and Skewed (i) contains clusters with different distances between their centroids.

The second subgroup contains four datasets that have been used in clustering studies in Bioinformatics. A summary of these datasets is given in Table 1. The number of clusters is the actual number of classes in each dataset.

The first subgroup of real-world data contains wisconsin, iris, wine, dermatology, zoo, yeast and digits. Details of these datasets are given in the UCI machine learning repository [2]. Note that most datasets from UCI are originally meant for classification problems, so they generally have more instances than the number of features.

To introduce datasets with other properties, such as having more features than instances, or of temporal nature, we use a second subgroup of data from the Bioinformatics domain. Two datasets are associated with the problem of finding groups of samples associated with certain phenotypes. The leukaemia [15] dataset has 38 samples (11 acute myeloid leukaemia (AML), eight T-lineage acute lymphoblastic leukaemia (ALL) and 19 B-lineage ALL). The novartis dataset [41] consists of tissue samples from four cancer types: 26 breast, 26 prostate, 28 lung, and 23 colon samples. Both leukaemia and novartis contain a small number of instances described by the expression levels of several thousand genes associated with the cancer types. Although a pre-processing step has been carried out to reduce the high number of features in these two datasets [33], their number of features are still substantially higher than the number of instances. The last two datasets are associated with finding groups of genes. The yeast cell cycle data [5] contains a subset of 384 genes [55] in the original data, and there are five gene groups and 17 conditions. Finally, the rat CNS datasets [51] contains 112 genes, 17 conditions and six gene groups; more details of this dataset are described in the work of [56].

The third and final group of data consists of three large datasets, which are described as follows:

PenDigits: This dataset is obtained from the UCI machine learning repository [2]. It has 16 attributes and 10 clusters representing digits from '0' to '9'. This dataset has 10 992 instances.

Animals: This dataset can be produced using the 'animals.c' program obtained from the UCI machine learning repository. It is officially known as the Quadruped Mammals Dataset [14]. It has 72 dimensions and four classes (each class represents one type of animal). We create a series of variants of animals datasets, each with a different number of instances (from 1250 to 200 000 instances) in order to test the runtime performance of the methods when data size grows.

OneBig: This dataset has been used to test data sampling techniques for clustering large data [35]. The dataset described here is taken from Appel et al. [1]. This dataset has 20 features and nine clusters. The biggest cluster has 50 000 instances. The other eight clusters are small, each containing 1000 instances. The remaining 10 000 points are noise randomly distributed in the feature space. The total number of instances for this dataset is 68 000.

Table 1
Summary of real-world and large datasets. N is the number of instances in the dataset; N_i is the number of instances for cluster i; Q is the number of features and C is the number of clusters.

Dataset	N	N_i	Q	C
UCI domains				
Wisconsin	699	458, 241	9	2
Dermatology	366	112, 61, 72, 49, 52, 20	34	6
Zoo	101	41, 20, 5, 13, 4, 8, 10	16	7
Wine	178	59, 71, 48	13	3
Iris	150	50, 50, 50	4	3
Digits	3498	363, 364, 364, 336, 364, 335, 336, 364, 336, 336	16	10
Yeast	1484	463, 429, 244, 163, 51, 44, 37, 30, 20, 5	8	10
Bioinformatics domains				
Leukaemia	38	11, 8, 19	999	3
Novartis	103	26, 26, 28, 23	1000	4
Yeast cell cycle	384	67, 135, 75, 52, 55	17	5
Rat CNS	112	27, 20, 21, 17, 21, 6	17	6
Large datasets				
PenDigits	10 992	1143, 1143, 1144, 1055, 1144, 1055, 1056, 1142, 1055, 1055	16	10
Animals	200000	50 000, 50 000, 50 000, 50 000	72	4
OneBig	68000	50 000, 1000, 1000, 1000, 1000 1000, 1000, 1000, 1000. noise: 10 000.	20	9

4.1.2. Algorithms and settings

For x-means and EMC, we use the implementations that are available from the Weka toolkit [53]. All the algorithms are implemented in Java and tested on the same machine (Pentium 3, 3 GHz with 1 GB RAM). This allows a head-to-head comparison of the runtime of the four algorithms. The results for each dataset are obtained based on 50 independent runs, and finally evaluated using the number of clusters detected, runtime taken, and *F-measure* [50]. All results are compared using two-sample t-test with 5% level of significance.

The *F-measure* is a broadly used external measure to quantify how well a set of generated clusters match the actual classes in the dataset. Let n_i (and n_j) be the number of items in class i (and cluster j respectively), then n_{ij} is the number of items in class i within cluster j. The F-measure is then defined as:

$$F(i,j) = \frac{2 \cdot prec(i,j) \cdot rec(i,j)}{prec(i,j) + rec(i,j)};$$

for each class i and cluster j, $prec(i,j) = n_{ij}/n_j$ and $rec(i,j) = n_{ij}/n_i$ are defined as the precision and the recall respectively. Note that this objective measure treats clustering like an information retrieval problem [50]. The overall F-measure can be computed as:

$$F = \sum_i \frac{n_i}{n} \max_j \{F(i,j)\}, \tag{7}$$

where n is the total number of items in a dataset. F-measure is in the interval [0, 1] and equals 1 for perfect clustering.

In clustering problems, we do not know the class labels or the cluster structure underlying a dataset in real-world scenarios; hence there is usually no information available for tuning the parameters of clustering algorithms. In our experiments, we use the default parameter settings for all algorithms. The only exception is x-means, where we have to set the upper and lower bounds for the number of clusters. Since the number of clusters in all the datasets used in this study is within [2, 10], we set the range for the putative number of clusters to be [1, 11].

4.2. Analysis of results in terms of F-measure

Table 2 presents the results of p GSCM, ATTA, EMC and x-means in terms of F-measure. On the whole, p GSCM performs very well in majority of the datasets.

Square and sizes series: ATTA performs well when clusters are relatively well separated (from Square1 to Square4) or when clusters are unevenly sized (i.e., sizes series), but its performance starts to degrade on highly overlapping clusters (Square5 onwards). EMC performs well on the Square series; but is weak in finding clusters with different sizes (especially Sizes2 and Sizes5). x-means is the worst performing method because its performance is sensitive to the range of *the number of clusters* input by users. Finally, p GSCM has the most consistent performance across the square and sizes series datasets.

Triangle, VaryDensity and Skewed series: For the triangle and varydensity datasets, p GSCM and EMC outperform ATTA. ATTA performs poorly by splitting the elongated cluster in triangle [45]

Table 2
***F*-measure** (mean $\pm$ standard deviation) of *p* GSCM, ATTA, EMC and *x*-means. Results that are significantly better than *p* GSCM are printed in boldface. Results that are significantly worst than *p* GSCM are underlined. Results are compared using two-sample *t*-test with 5% level of significance.

Dataset	*p* GSCM	ATTA	EMC	*x*-Means
Increasingly overlapping clusters				
Square1	0.99 ± 0.00	0.98 ± 0.01	0.99 ± 0.01	0.40 ± 0.00
Square2	0.97 ± 0.00	0.96 ± 0.01	0.97 ± 0.00	0.40 ± 0.00
Square3	0.94 ± 0.00	0.93 ± 0.01	**0.95 ± 0.00**	0.40 ± 0.00
Square4	0.90 ± 0.03	0.89 ± 0.02	**0.91 ± 0.00**	0.40 ± 0.00
Square5	0.85 ± 0.01	0.81 ± 0.04	**0.86 ± 0.00**	0.40 ± 0.00
Square6	0.68 ± 0.04	0.63 ± 0.11	**0.74 ± 0.04**	0.40 ± 0.00
Square7	0.62 ± 0.05	0.43 ± 0.06	**0.64 ± 0.02**	0.40 ± 0.00
Increasingly uneven cluster sizes				
Sizes1	0.98 ± 0.00	0.98 ± 0.00	**0.99 ± 0.00**	0.40 ± 0.00
Sizes2	0.99 ± 0.00	0.98 ± 0.00	0.86 ± 0.07	0.40 ± 0.00
Sizes3	0.98 ± 0.00	0.98 ± 0.00	**0.99 ± 0.00**	0.40 ± 0.00
Sizes4	0.99 ± 0.00	0.98 ± 0.00	0.99 ± 0.00	0.40 ± 0.00
Sizes5	0.99 ± 0.00	0.96 ± 0.06	0.89 ± 0.11	0.40 ± 0.00
Different cluster shapes/densities/distances				
Triangle	1.00 ± 0.00	0.91 ± 0.05	1.00 ± 0.00	0.56 ± 0.10
VaryDensity	0.98 ± 0.02	0.83 ± 0.01	0.98 ± 0.03	0.40 ± 0.00
Skewed	0.63 ± 0.00	**0.99 ± 0.00**	**1.00 ± 0.00**	**0.87 ± 0.18**
***x*D*y*C datasets**				
2D4C	0.98 ± 0.04	0.97 ± 0.04	0.99 ± 0.05	0.78 ± 0.28
2D10C	0.87 ± 0.07	**0.95 ± 0.04**	0.89 ± 0.09	0.21 ± 0.01
10D4C	1.00 ± 0.00	0.99 ± 0.00	0.99 ± 0.04	0.94 ± 0.10
10D10C	1.00 ± 0.00	1.00 ± 0.00	0.91 ± 0.09	0.55 ± 0.39
100D4C	1.00 ± 0.00	0.99 ± 0.00	0.99 ± 0.02	0.96 ± 0.07
100D10C	1.00 ± 0.00	1.00 ± 0.00	0.89 ± 0.04	1.00 ± 0.00
UCI domains				
Wisconsin	0.96 ± 0.02	**0.97 ± 0.00**	0.68 ± 0.10	0.67 ± 0.00
Wine	0.95 ± 0.01	0.86 ± 0.02	0.85 ± 0.04	0.95 ± 0.00
Dermatology	0.89 ± 0.00	0.82 ± 0.03	0.81 ± 0.08	0.89 ± 0.02
Zoo	0.92 ± 0.01	0.69 ± 0.04	0.73 ± 0.03	0.62 ± 0.10
Yeast	0.46 ± 0.01	0.44 ± 0.04	0.43 ± 0.04	0.35 ± 0.00
Digits	0.77 ± 0.03	0.53 ± 0.02	0.68 ± 0.12	0.70 ± 0.05
Iris	0.91 ± 0.08	0.79 ± 0.02	0.70 ± 0.03	0.78 ± 0.00
Bioinformatics domains				
Leukaemia	0.79 ± 0.04	0.54 ± 0.00	0.80 ± 0.01	0.54 ± 0.00
Novartis	0.96 ± 0.02	0.91 ± 0.03	0.97 ± 0.07	0.97 ± 0.05
Yeast cell cycle	0.67 ± 0.02	**0.68 ± 0.01**	0.59 ± 0.04	**0.68 ± 0.01**
Rat CNS	0.65 ± 0.05	0.55 ± 0.08	**0.76 ± 0.05**	0.41 ± 0.18

and tends to overestimate α in varydensity. *p* GSCM works well because each data item is assigned a unique dissimilarity threshold and this allows the method to differentiate clusters with different densities. For the skewed dataset, *p* GSCM fuses the three nearby clusters into one cluster. To address this problem, we can apply *p* GSCM twice: after the first application, it is applied on the data associated with the big cluster containing the three nearby clusters. This will extract all the clusters required.

***x*D*y*C series**: *p* GSCM and ATTA are the top two performing methods. *p* GSCM frequently ranks first; the only dataset that *p* GSCM ranks third is the 2D10C dataset. This is because the dissimilarity threshold is more difficult to define in the 'congested' 2D space, and becomes more well-defined in a higher dimensional space where data is sparse. This is why its performance quickly caught up when tested on datasets with 10 and 100 dimensions.

UCI and Bioinformatics domains: Table 2 shows that *p* GSCM is the best performing method with its *F*-measure usually ranked either first or second. For the yeast dataset, both ATTA and *p* GSCM perform poorly; a previous study [17] has suggested that this is because the cluster structure of the yeast dataset is not easily noticeable.

ATTA does not perform well on the leukaemia dataset that contains more features than instances. While EMC performs very well on most of the synthetic data, its performance starts to degrade when dealing with real-world data. This is not surprising because all the synthetic datasets are generated based on Gaussians, which match the assumptions held by EMC. As for *x*-Means, it performs quite poorly on synthetic data, but its performance starts to improve on some of the real-world data.

4.3. Analysing results on the number of clusters detected

Table 3 shows that the results on the number of clusters detected; they are well correlated with the *F*-measure results in Table 2. This is because both of these measures evaluate the clustering accuracy. The key difference is that *F*-measure contains more information because it evaluates how well the generated clusters match the actual classes in each dataset, whereas the number of clusters detected does not evaluate the accuracy of cluster assignments.

For all the synthetic data (i.e., squares, sizes, different cluster shapes/densities/distances and *x*D*y*C), *p* GSCM is the only method that most frequently identifies the number of clusters that is closest to the actual number of clusters. EMC performs better than ATTA on the square series datasets; but loses to ATTA on the *x*D*y*C datasets. The results of *x*-means are poor, which agree with the *F*-measure results in Table 2.

In terms of real-world data from the UCI and Bioinformatics domains, *p* GSCM is either ranked first or second in all but one (i.e., yeast) dataset. As for ATTA, EMC and *x*-means, their results are mixed, having a big swing from one dataset to another.

4.4. Analysing results on the runtime performance

Table 4 shows that *p* GSCM runs faster than ATTA, EMC and *x*-means.

ATTA and EMC take increasingly longer time to find clusters that are increasingly overlapped in the square series datasets. ATTA has this problem because it requires an unpredictable duration to search for items [44]. As for EMC, it requires more time to converge because the underlying cluster structures become vague when the clusters are highly overlapped. Unlike ATTA and EMC, the runtime of *p* GSCM remains constant for each of the square series datasets.

For the runtime performance of the four methods on the *x*D*y*C datasets, Table 4 shows that *p* GSCM is the fastest. *x*-means and ATTA are ranked second and third respectively. EMC is the slowest method—its runtime is unrealistically long when the dimensionality is high in the 100D10C dataset.

ATTA is slower than *p* GSCM because every movement of an item involves multiple distance computations. In contrast, the movement of each data item in *p* GSCM is decided based on only a single distance computation.

x-Means is fast; but its relatively poor clustering accuracy suggests that the method perhaps converge too early.

4.5. Clustering large data

In this section, we assess the performance of *p* GSCM versus ATTA and *x*-means for large data. Since EMC is confirmed to be slow, we will exclude it from the experiments reported here.

The experimental setup is the same as that described in Section 4.1.2, the only exception is that we reduce the number of independent runs (for each dataset), from 50 to 10 because of large data size.

p GSCM versus ATTA and *x*-means:

Table 5 presents the clustering performance of *p* GSCM, ATTA and *x*-means on datasets with different sizes. The aim of the experiment is to examine whether the algorithms can work with different data sizes under the constraint of a standard computer configuration (Pentium 3, 3 GHz and 1 GB RAM).

Table 3
Number of clusters (mean $\pm$ standard deviation) detected by *p* GSCM, ATTA, EMC and *x*-means. ***K*** is the actual number of clusters. Averaged results that are the *least deviated* from ***K*** are printed in boldface. Averaged results that are *most deviated* from ***K*** are underlined.

Dataset	*K*	*p* GSCM	ATTA	EMC	*x*-Means
Increasingly overlapping clusters					
Square1	**4**	**4.00 ± 0.00**	4.02 ± 0.14	4.16 ± 0.37	1.00 ± 0.00
Square2	**4**	**4.00 ± 0.00**	**4.00 ± 0.00**	4.06 ± 0.31	1.00 ± 0.00
Square3	**4**	**4.00 ± 0.00**	**4.00 ± 0.00**	**4.00 ± 0.00**	1.00 ± 0.00
Square4	**4**	3.96 ± 0.20	**4.00 ± 0.00**	**4.00 ± 0.00**	1.00 ± 0.00
Square5	**4**	**4.00 ± 0.00**	3.94 ± 0.24	**4.00 ± 0.00**	1.00 ± 0.00
Square6	**4**	3.70 ± 0.76	3.00 ± 0.99	**3.72 ± 0.45**	1.00 ± 0.00
Square7	**4**	**3.68 ± 0.62**	1.32 ± 0.65	3.66 ± 0.56	1.00 ± 0.00
Increasingly uneven cluster sizes					
Sizes1	**4**	**4.00 ± 0.00**	4.02 ± 0.14	**4.00 ± 0.00**	1.00 ± 0.00
Sizes2	**4**	**4.00 ± 0.00**	**4.00 ± 0.00**	4.80 ± 0.40	1.00 ± 0.00
Sizes3	**4**	**4.00 ± 0.00**	**4.00 ± 0.00**	**4.00 ± 0.00**	1.00 ± 0.00
Sizes4	**4**	**4.00 ± 0.00**	4.04 ± 0.20	**4.00 ± 0.00**	1.00 ± 0.00
Sizes5	**4**	**3.98 ± 0.14**	3.86 ± 0.70	4.42 ± 0.50	1.00 ± 0.00
Different cluster shapes/densities/distances					
Triangle	**4**	**4.00 ± 0.00**	5.38 ± 1.09	**4.00 ± 0.00**	1.72 ± 0.45
VaryDensity	**4**	**3.98** ± 0.14	3.20 ± 0.40	4.22 ± 0.42	1.00 ± 0.00
Skewed	**4**	2.00 ± 0.00	4.06 ± 0.24	**4.00 ± 0.00**	3.32 ± 0.96
***x*D*y*C datasets**					
2D4C	**4**	3.84 ± 0.37	**4.14 ± 0.53**	4.16 ± 0.42	2.86 ± 1.51
2D10C	**10**	7.64 ± 1.14	**10.98 ± 1.41**	12.26 ± 2.84	1.00 ± 0.00
10D4C	**4**	**4.00 ± 0.00**	4.02 ± 0.14	4.26 ± 0.66	4.58 ± 1.07
10D10C	**10**	**9.98 ± 0.14**	**10.02 ± 0.14**	13.14 ± 2.13	5.02 ± 4.59
100D4C	**4**	**4.00 ± 0.00**	**4.00 ± 0.00**	4.12 ± 0.33	4.36 ± 0.63
100D10C	**10**	**10.00 ± 0.00**	**10.00 ± 0.00**	13.51 ± 1.28	**10.00 ± 0.00**
UCI domains					
Wisconsin	**2**	2.06 ± 0.24	**2.00 ± 0.00**	6.02 ± 1.76	6.00 ± 0.00
Wine	**3**	3.02 ± 0.14	3.02 ± 0.14	4.50 ± 0.74	**3.00 ± 0.00**
Dermatology	**6**	5.00 ± 0.00	4.12 ± 0.39	**5.32 ± 1.24**	4.86 ± 0.35
Zoo	**7**	**5.96 ± 0.20**	3.18 ± 0.66	3.36 ± 1.03	2.18 ± 0.72
Yeast	**10**	3.24 ± 0.48	**5.12 ± 1.33**	4.32 ± 1.60	1.00 ± 0.00
Digits	**10**	11.58 ± 0.57	5.92 ± 0.57	13.28 ± 5.39	**11.00 ± 0.00**
Iris	**3**	2.78 ± 0.42	**3.04 ± 0.20**	5.76 ± 0.82	2.00 ± 0.00
Bioinformatics domains					
Leukaemia	**3**	4.94 ± 0.59	1.06 ± 0.24	**2.00 ± 0.00**	1.00 ± 0.00
Novartis	**4**	**4.06 ± 0.02**	**4.06 ± 0.37**	**3.94 ± 0.42**	3.92 ± 0.27
Yeast cell cycle	**5**	**4.32 ± 0.51**	4.02 ± 0.14	9.26 ± 1.60	4.10 ± 0.30
Rat CNS	**6**	**5.04 ± 0.88**	3.34 ± 0.80	4.92 ± 0.78	1.66 ± 1.26

First, we observe that *p* GSCM is better at dealing with large datasets compared to ATTA. For example, our results show that *p* GSCM works well in detecting the clusters in both digits 3.5k and digits 11k, whereas ATTA runs out of memory space when working on digits 11k. This result shows that *p* GSCM has a lower memory requirement compared to ATTA, and *p* GSCM is likely to work well on larger datasets without the need to increase the memory capacity of our computer.

Second, when the data size of animals doubles from 1.25k to 2.5k (i.e., 2500 instances), ATTA triples its runtime, whereas *p* GSCM only doubles its runtime. It is interesting to note that ATTA also uses a parameter setting $k \cdot n$ to control the maximum number of iterations. However, each iteration in ATTA also contains sentinel loops[1] forming part of the clustering process. This explains why its runtime does not increase linearly with data size.

As the data size grows to 5000 instances and above, ATTA begins to encounter memory capacity problem, whereas *p* GSCM continues to run well under the same computer configuration. When doubling the size of the animals dataset from 50 000 to 100 000 instances; we observe that the runtime of *p* GSCM increases 2.18 times. When doubling the size from 100 000 to 200 000 instances, the runtime again grows at about the same rate, which is 2.13 times. Thus, we do not expect the runtime of *p* GSCM to increase substantially as the input data size grows. It is interesting to note that *x*-means works very well initially for 2500 instances or less, but its *F*-measure drops significantly when the number of instances in the Animal dataset increases to 5000 instances and beyond. This suggests that *x*-means is not robust against different data characteristics. In this case, a simple increase in data size is enough to cripple the method. Furthermore, *x*-means runs out of memory space for the Animal dataset with 200 000 instances.

Finally, we present the results of *p* GSCM on the onebig dataset. The original onebig dataset contains 68 000 instances. We create two smaller versions of the onebig dataset using simple random samples of sizes 3688 and 6754 instances respectively (which are about 4% and 10% of the actual data size respectively). Table 5 shows that ATTA can only run on the smallest version of the datasets, whereas *p* GSCM performs well with a significantly faster runtime (11 versus 163 s) and a better *F*-measure on the same dataset. Furthermore, *p* GSCM also runs well on the larger onebig datasets. As for *x*-means, it fails to detect the smaller clusters in these datasets, as indicated by the relatively small *F*-measures.

On the whole, *p* GSCM outperforms ATTA and *x*-means on the large datasets used in this study, in terms of clustering quality (i.e., *F*-measure and the number of clusters detected) and it requires significantly less memory space. In terms of runtime,

[1] A sentinel loop is a kind of 'while loop' which has an indefinite loop count.

Table 4
Runtime (mean $\pm$ standard deviation) of *p* GSCM, ATTA, EMC and *x*-means. The runtime is measured in seconds. Runtime results that are significantly faster than *p* GSCM are printed in boldface. Runtime results that are significantly slower than *p* GSCM are underlined. Results are compared using two-sample *t*-test with 5% level of significance.

Dataset	*p* GSCM	ATTA	EMC	*x*-Means
Increasingly overlapping Clusters				
Square1	1.87 ± 0.01	10.39 ± 1.04	28.82 ± 4.90	2.52 ± 0.04
Square2	1.98 ± 0.17	10.56 ± 0.94	26.95 ± 4.64	2.52 ± 0.02
Square3	1.85 ± 0.02	11.13 ± 0.96	30.17 ± 0.66	2.53 ± 0.06
Square4	1.95 ± 0.05	12.63 ± 1.46	30.62 ± 0.97	2.52 ± 0.01
Square5	1.85 ± 0.03	13.16 ± 0.93	29.91 ± 1.05	2.65 ± 0.31
Square6	1.97 ± 0.12	15.63 ± 2.92	40.05 ± 6.44	2.52 ± 0.01
Square7	1.85 ± 0.02	21.12 ± 10.63	41.39 ± 6.84	2.57 ± 0.06
Increasingly uneven cluster sizes				
Sizes1	1.87 ± 0.02	10.42 ± 0.95	29.35 ± 0.66	2.53 ± 0.05
Sizes2	1.96 ± 0.04	11.71 ± 1.62	38.65 ± 7.58	2.53 ± 0.01
Sizes3	1.87 ± 0.02	11.57 ± 1.29	23.87 ± 0.68	2.55 ± 0.07
Sizes4	1.94 ± 0.02	14.47 ± 2.54	27.2 ± 1.27	2.53 ± 0.01
Sizes5	1.86 ± 0.02	18.30 ± 5.34	38.61 ± 6.54	2.53 ± 0.04
Different cluster shapes/densities/distances				
Triangle	1.93 ± 0.01	11.92 ± 0.72	29.73 ± 0.92	2.54 ± 0.04
VaryDensity	1.91 ± 0.03	15.22 ± 4.47	21.57 ± 5.76	2.53 ± 0.04
Skewed	2.16 ± 0.16	14.36 ± 0.50	21.24 ± 1.15	2.55 ± 0.02
x**D*****y*****C datasets**				
2D4C	2.04 ± 0.44	13.69 ± 5.65	22.07 ± 6.62	2.46 ± 0.12
2D10C	4.75 ± 0.56	18.74 ± 3.41	319.88 ± 143.19	$\mathbf{2.66 \pm 0.09}$
10D4C	2.22 ± 0.49	12.25 ± 3.27	59.9 ± 34.71	2.78 ± 0.13
10D10C	5.50 ± 0.94	18.89 ± 3.46	1145.92 ± 386.14	$\mathbf{3.54 \pm 0.36}$
100D4C	4.06 ± 1.05	10.00 ± 1.86	425.14 ± 200.69	6.04 ± 0.90
100D10C	12.23 ± 1.84	17.93 ± 2.67	11711.46 ± 3570.80	14.17 ± 1.81
UCI domains				
Wisconsin	1.49 ± 0.06	12.58 ± 0.39	55.92 ± 19.87	2.85 ± 0.03
Wine	0.66 ± 0.01	4.44 ± 0.16	24.05 ± 3.40	2.47 ± 0.04
Dermatology	0.92 ± 0.01	5.77 ± 0.43	50.72 ± 13.56	2.74 ± 0.03
Zoo	0.66 ± 0.04	4.67 ± 0.33	14.49 ± 1.79	2.43 ± 0.05
Yeast	2.97 ± 0.07	10.42 ± 0.16	93.25 ± 45.93	$\mathbf{2.78 \pm 0.05}$
Digits	8.85 ± 0.47	35.58 ± 0.61	2349.05 ± 1320.4	$\mathbf{5.82 \pm 0.35}$
Iris	0.60 ± 0.01	5.51 ± 0.19	18.50 ± 1.83	2.41 ± 0.03
Bioinformatics domains				
Leukaemia	7.15 ± 1.56	$\mathbf{6.14} \pm 0.23$	32.96 ± 0.98	$\mathbf{3.06 \pm 0.07}$
Novartis	6.39 ± 0.02	$\mathbf{6.03} \pm 0.45$	208.28 ± 47.26	$\mathbf{5.48 \pm 0.31}$
Yeast cell cycle	0.90 ± 0.02	5.13 ± 0.48	147.56 ± 40.13	2.79 ± 0.33
Rat CNS	0.69 ± 0.01	3.87 ± 0.19	15.59 ± 2.45	2.36 ± 0.07

x-means is the fastest. However, this could be because *x*-means converges too early, which results in poor clustering quality (e.g., see its results on 'animals 5k' and onebig datasets.)

5. Discussions

This section discusses the effects of different parameter settings on the performance of *p* GSCM. It also explains why *p* GSCM performs well and gives a conceptual comparison of *p* GSCM with existing SBC and traditional methods.

5.1. Effects of parameter settings on p GSCM

Here, we analyse the effects of the key parameter settings (ω, k, ψ, b and s) on the overall performance of *p* GSCM. Parameter ω controls the dissimilarity threshold. Parameter k is a coefficient that sets the number of iterations and controls the terminating condition of *p* GSCM. Parameter ψ is the sample size used to estimate the initial dissimilarity threshold. Parameter b is the sample size estimating the level of support. Parameter s is the smallest bin size threshold whereby bins with size less than s will be dissolved. If *p* GSCM is robust in performance, its average clustering accuracy should not change drastically with respect to small changes in the parameter settings.

As shown in Table 6, each parameter value is stepped from its minimum value (*minV*) to its maximum value (*maxV*) using a fixed step size (*step*). For each step, *p* GSCM is executed 10 times on each of the 32 datasets shown in Table 2. The overall performance is measured using the average *F*-measure of 320 clustering results. Table 6 shows that the biggest range of average *F*-measures obtained from these experiments is very small, only within [0.84, 0.88]. The results indicate that ω within its sensible range produces good results in a wide range of datasets. The results also show that, once the final clusters emerge, there is little or no change in the clustering solution with increased number of iterations (k). Furthermore, increasing ψ and b simply increase estimation accuracies and do not degrade the clustering performance of *p* GSCM.

Apart from the parameters, we also investigate the effects of using a different distance metric. We change the distance metric from 1-norm (i.e., the default distance metric) to 2-norm (or Euclidean), and the average *F*-measures are found to be 0.88 and 0.87 respectively, indicating that the change is practically small.

It is our experience that many existing methods would not perform well if they were to use a fixed set of parameter settings for the datasets used in our experiments. For example, users of *k*-means will need to change *k* whenever the number of clusters is different from the default settings. One might argue that the use of cluster validity indexes could help to alleviate this issue, but this approach normally requires multiple runs of clustering algorithms.

Table 5
Results of p GSCM, ATTA and x-means on a range of small and large datasets. Each entry is the mean $\pm$ standard deviation. ***K*** is the actual number of classes in each dataset. **OM** means a method runs out of memory space.

Dataset	*K*	*p* GSCM	ATTA	*x*-Means
***F*-measure**				
Digits 3.5k		0.76 ± 0.03	0.52 ± 0.02	0.71 ± 0.02
Digits 11k		0.72 ± 0.03	**OM**	**0.74 ± 0.02**
Animals 1.25k		1.00 ± 0.00	0.66 ± 0.00	1.00 ± 0.00
Animals 2.5k		1.00 ± 0.00	0.67 ± 0.00	1.00 ± 0.02
Animals 5k		1.00 ± 0.00	**OM**	0.60 ± 0.02
Animals 10k		1.00 ± 0.00	**OM**	0.60 ± 0.01
Animals 50k		0.98 ± 0.05	**OM**	0.60 ± 0.02
Animals 100k		1.00 ± 0.00	**OM**	0.59 ± 0.01
Animals 200k		1.00 ± 0.00	**OM**	**OM**
OneBig 3.7k		0.96 ± 0.01	0.85 ± 0.00	0.60 ± 0.05
OneBig 6.8k		0.94 ± 0.19	**OM**	0.64 ± 0.00
OneBig 68k		0.94 ± 0.01	**OM**	0.63 ± 0.03
Number of clusters detected				
Digits 3.5k	**10**	11.50 ± 0.53	5.70 ± 0.48	**11.00 ± 0.00**
Digits 11k	**10**	**9.00 ± 1.56**	**OM**	**11.00 ± 0.00**
Animals 1.25k	**4**	**4.00 ± 0.00**	2.00 ± 0.00	**4.00 ± 0.00**
Animals 2.5k	**4**	**4.00 ± 0.00**	2.00 ± 0.00	**4.00 ± 0.00**
Animals 5k	**4**	**4.00 ± 0.00**	**OM**	11.00 ± 0.00
Animals 10k	**4**	**4.00 ± 0.00**	**OM**	11.00 ± 0.00
Animals 50k	**4**	**3.90 ± 0.32**	**OM**	11.00 ± 0.00
Animals 100k	**4**	**4.00 ± 0.00**	**OM**	11.00 ± 0.00
Animals 200k	**4**	**4.00 ± 0.00**	**OM**	**OM**
OneBig 3.7k	**9**	**9.20 ± 0.42**	2.10 ± 0.32	11.00 ± 0.00
OneBig 6.8k	**9**	**9.00 ± 0.00**	**OM**	11.00 ± 0.00
OneBig 68k	**9**	**9.10 ± 0.32**	**OM**	11.00 ± 0.00
Runtime (seconds)				
Digits 3.5k		8.82 ± 0.26	35.62 ± 0.45	**5.88 ± 0.86**
Digits 11k		34.84 ± 0.21	**OM**	**15.42 ± 1.30**
Animals 1.25k		4.42 ± 0.07	17.30 ± 0.35	4.53 ± 0.34
Animals 2.5k		10.67 ± 0.15	60.80 ± 1.07	**6.80 ± 0.37**
Animals 5k		23.59 ± 0.65	**OM**	**16.14 ± 1.55**
Animals 10k		51.48 ± 0.23	**OM**	**36.88 ± 2.71**
Animals 50k		265.78 ± 13.42	**OM**	**240.14 ± 16.10**
Animals 100k		578.64 ± 14.86	**OM**	**473.66 ± 35.12**
Animals 200k		**1231.75** ± 1.93	**OM**	**OM**
OneBig 3.7k		11.02 ± 0.09	163.29 ± 4.80	**6.30 ± 0.54**
OneBig 6.8k		23.33 ± 0.18	**OM**	**11.09 ± 1.10**
OneBig 68k		286.98 ± 0.24	**OM**	**148.25 ± 32.75**

Table 6
The effects of key parameters on the clustering performance of p GSCM evaluated using the average F-measure over 32 datasets.

Parameter	Setting			Average *F*-measure	
	minV	*maxV*	*step*	*min*	*max*
ω	0.2	0.4	0.05	0.84	0.88
k	1000	10 000	1000	0.88	0.89
ψ	150	900	150	0.88	0.89
b	30	150	30	0.88	0.88
s	1%	10%	1%	0.87	0.88

Furthermore, many cluster validity indexes are known to make assumptions about the cluster structure of datasets and such methods will work well only when the assumptions are not violated.

5.2. *Why does p GSCM work well?*

There are two properties of p GSCM that contribute to its good clustering accuracy. The first property is that each cluster is formed using core points (i.e., data points around centroids), before incorporating the fringe points (i.e., data points near fringes of clusters). This occurs because there are usually *more* core points populated around centroids and *fewer* fringe points scattered around cluster fringes. With the random sampling scheme used in p GSCM, core points are more likely to be selected than fringe points during the clustering process. In addition, there are more neighbouring core points with small dissimilarity thresholds that are deemed similar compared to fewer fringe points with larger dissimilarity thresholds. Since core points are higher in quantity and density, initial cluster are likely to form using core points. We illustrate this using a connected graph $G(V, E)$, where each data point is a node $v \in V$ and two similar items i and j (which passes the acceptance test in Eq. (5)) forms an edge $e_{ij} \in E$. Each edge has a weight which records the number of times two randomly selected items are deemed similar since the beginning of the clustering process. Fig. 3 shows that, during the initial clustering process, the cluster centroids in each dataset are more heavily connected around the core points than the fringe points. This means that the initial clusters formation are using the core points. As a result, p GSCM can accurately detect clusters with dense points around centroids and it makes no explicit assumption about the degrees of overlaps, sizes, and densities in clusters.

Because noise points are less well connected, the above property of p GSCM does not include them in the bins and eventually treat them rightly as noise or outliers. This is why p GSCM works well even when 10 000 noise points are added to the onebig dataset. However, it is possible that our method may fail if there exists some extreme outliers that are very far from majority of the points. These extreme outliers can cause the mean dissimilarity component (of Eq. (5)) to be increased substantially. This, however, is not a major concern because extreme outliers are usually obvious and they can be eliminated during the data cleaning stage of data mining process.

The second property is that p GSCM is able to take advantage of each item's unique dissimilarity threshold to detect clusters of different densities. We have showed that p GSCM can detect clusters of different densities in the varydensity dataset. Fig. 3c shows that there is a low connectivity between the dense cluster (C1) and sparse cluster (C2). This is because most of the points in C1 have smaller dissimilarity threshold than the points in C2.

To illustrate the usefulness of these properties, we give another example in the Fig. 3d. This dataset consists of four Gaussian clusters; two of which are dense clusters (C3 and C4) situated in between two overlapping but sparse clusters (C1 and C2). It shows that cluster C3 is not linearly separable from C1, C2 and C4; and likewise for C4 with respect to C1, C2 and C3. p GSCM detects four clusters in this dataset in 49 out of 50 runs, and gives an average F-measure of 0.86. This is significantly better than x-means, which gives an average F-measure of 0.81 despite that it has been given the exact number of clusters (i.e., four) as input. ATTA's performance is the worst; it detects only one cluster. The average F-measure of EMC is 0.92 because C1–C4 are Gaussian clusters which match the assumption of EMC. For real-world datasets, EMC may not perform better than p GSCM. For example, the iris dataset is known to have two clusters that are not linearly separable, and Table 2 shows that p GSCM actually outperforms EMC on this dataset. p GSCM works well in this case because it does not attempt to model the non-linear boundary directly. Instead, it simply starts building clusters from their core points.

Our experimental results also show that p GSCM is efficient. This is largely attributed to the *level of support* heuristic described in Section 3.2, which speeds up the assignment of data points to appropriate clusters. In fact, the simple GSCM (without this heuristic) will work quite inefficiently, taking up hundreds of iterations just to complete clustering a dataset with 1000 items. The heuristic is able to reduce the number of iterations by 40-fold. Interested readers can refer to [48] for details.

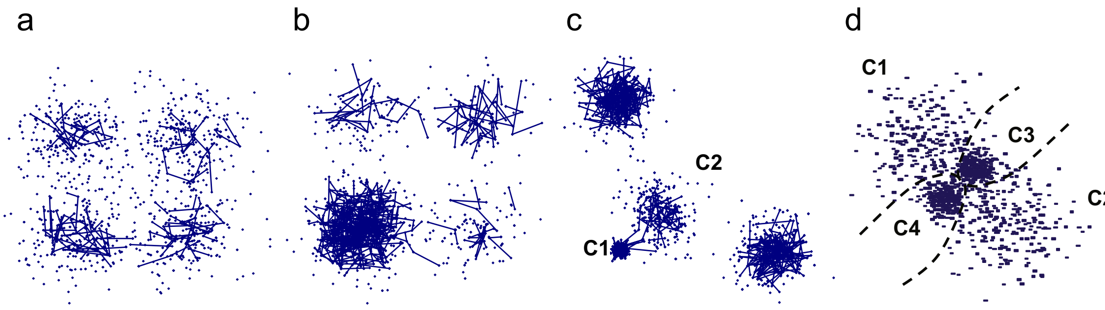

Fig. 3. (a)–(c) Connected graphs showing the initial cluster formation after the first 20*n* iterations. For clarity, we show 80% of the most heavily weighted edges constructed during the clustering process. (d) Significantly overlapping clusters that are hard to separate: (a) Square3, (b) Sizes3, (c) VaryDensity and (d) significant overlap.

5.3. *Relation to SBC methods*

Although *p* GSCM is inspired from SBC, it differs from existing SBC methods. First, ATTA requires a 2D grid whereas *p* GSCM does not. This means that ATTA requires an additional algorithm to obtain an explicit partitioning from the grid. This additional step has a quadratic time and space complexities.

Second, ATTA requires a precomputed distance matrix in order to work efficiently. The time and space complexities for obtaining the precomputed distance matrix are both $O(n^2)$. This is the reason why it runs out of memory when data size becomes large. As shown earlier, *p* GSCM works without a distance matrix, and its time and space complexities are linear.

Third, ATTA uses a global dissimilarity threshold value (α) for an entire dataset can cause problems when there are clusters with different densities. The algorithm will fail to detect either dense clusters (if the α-value is too large) or the sparse clusters (if the α-value is too small). In contrast, *p* GSCM uses a unique similarity threshold for each data item and is able to detect clusters with overlaps, uneven sizes and varying densities.

In summary, the superior performance of *p* GSCM over ATTA is a direct result of algorithmic simplicity and more target operations.

Note that *p* GSCM has also been shown to outperform two other SBC methods, namely flocking-based clustering (i.e., FClust [11]) and ant-chemical recognition method (i.e., AntClust [27]). To ensure fair comparisons, we use the datasets and putatively the best possible results originally published in FClust's and AntClust's papers. Comparing with FClust, *p* GSCM performs better with 11 wins and one loss. In addition, *p* GSCM performs better than AntClust with eight wins and three losses. Readers can refer to [48] for details.

5.4. *Relation to traditional clustering methods*

In Section 2.1, we have given a detailed account of a previous experimental study [17], where the authors used 25 (out of 32) datasets in Table 2 and showed that ATTA performs competitively against the well-known gap statistic used in conjunction with *k*-means, as well as traditional clustering methods such as *k*-means, hierarchical clustering using average-link (AvgLink) and self-organizing map (SOM). In this paper, we has shown that *p* GSCM outperforms ATTA on the exact same group of data. Hence we can conclude that *p* GSCM also outperforms these traditional methods.

Unlike *k*-means, *p* GSCM automatically forms its cluster using core points and does not required user to specify the number of clusters and the centroids.

During each agglomerative operation, hierarchical clustering employs a *deterministic search* of two 'nearest' clusters (which is defined by the linkage metric) and merges the two clusters. Instead of searching for clusters deterministically, *p* GSCM *randomly* finds two similar items and moves one item from its cluster to the other cluster that has a higher level of support. Furthermore, Hierarchical clustering often entails the computation of similarity matrix in order to speed up distance computations. This process has a time and space complexities of $O(n^2)$. In contrast, *p* GSCM does not require similarity matrix, and we have shown that it has a linear time complexity.

The key limitations of SOM are that users must choose appropriate settings of parameters; and the method tends to split or combine clusters when there are varying sizes, shapes and densities [43]. *p* GSCM differs from SOM in that it does not use a grid and therefore does not require a neighbourhood function or define the grid type. Furthermore, SOM requires users to choose the number of centroids, whereas *p* GSCM discovers the clusters (and therefore the centroids) automatically. Another issue is that SOM generates its clusters on a grid and it is difficult to quantify the clustering quality and consequently hard to compare different SOM clustering results [43]. In contrast, *p* GSCM generates an explicit partitioning of data that can be evaluated and compared.

Fig. 4 presents the average *F*-measure results of *p* GSCM, *k*-means, AvgLink and SOM. Unlike *p* GSCM, the traditional methods are unable to perform consistently well throughout all the datasets. For example, *k*-means produces the highest and lowest *F*-measure on dermatology and zoo respectively. Similarly, SOM performs well on the iris dataset; but gives the worst result on the dermatology dataset. AvgLink also pales in comparison—its results are the lowest in three of the seven datasets.

Out of the seven real-world datasets, *p* GSCM produces the highest *F*-measure on five of them; these datasets include yeast, digits, iris, wine and zoo. As for the two remaining datasets, *p* GSCM scores marginally below its rivals. Due to space limitation, we cannot provide details of other results; but interested readers can refer to our paper [47] for more information.

5.5. *p GSCM versus EMC*

The basic difference between *p* GSCM and EMC is that EMC attempts to model the actual cluster structure, whereas *p* GSCM performs a stochastic search with the aim of grouping similar items together. Thus, EMC assumes that each cluster in a dataset can be characterised by a certain probability distribution, whereas *p* GSCM does not make any explicit assumption about the distribution of each cluster. For each data point, EMC computes its probability of cluster membership to decide whether to assign the item to a cluster. *p* GSCM, on the other hand, heuristically assigns an item *i* to a bin only when that bin contains many items similar to *i*.

Another difference between *p* GSCM and EMC is in the way the algorithms determine when to terminate the iterative process. *p* GSCM terminates its process when the maximum number of iteration is reached, which is denoted by $k\cdot n$, where *k* is a positive integer constant and *n* is number of instances. As for EMC, the iteration usually stops when the change in overall

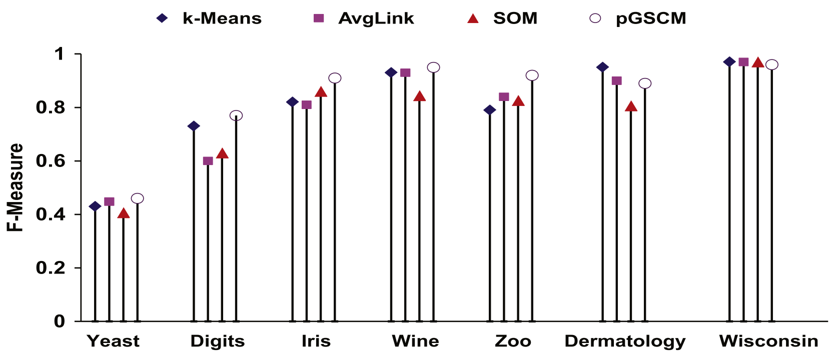

Fig. 4. Average *F*-measure of four clustering methods on seven real-world datasets.

likelihood of the model (which represents the clustering quality) becomes negligible. Since it can be difficult to know in advance when the iterative process terminates, the runtime of EMC is not predictable. Furthermore, there is no complexity analysis associated with the EMC technique in Weka [53].

p GSCM and EMC are similar in terms of their capability to perform automatic cluster discovery. However, the approaches used by the two methods to achieve this common goal are fundamentally different. While EMC explicitly estimates the number of clusters, *p* GSCM does it implicitly—the clusters so formed is a direct result of the stochastic process.

5.6. *p* GSCM versus *x*-means

The key difference between *p* GSCM and *x*-means is that *x*-means attempts to find the model that produces the best BIC score, whereas *p* GSCM makes no such estimation. In fact, *p* GSCM is not aware of the overall quality of the model throughout the clustering process. Since *x*-means relies considerably on BIC Statistic, it is likely to fail when the data characteristics do not match the BIC Statistic. One such example is non-spherical data, where *x*-Means tends to overfit by choosing too many cluster centres [16].

Although *x*-means does not require users to specify the exact number of clusters in a dataset, it still requires users to input the upper and lower bounds of the number of clusters to be estimated in the dataset [36]. This requires much guesswork when domain knowledge is not available. Unlike *x*-means, *p* GSCM derives the number of clusters without requiring such bounds.

In terms of runtime, both *p* GSCM and *x*-means are efficient. While we show that *p* GSCM has $O(n)$ time and space complexities, only empirical runtime analysis of *x*-means is available [36].

The most similar aspect between *p* GSCM and *x*-means is that they are both partitioning methods. Once new cluster centres are derived, *x*-means assign each point to the cluster that has the nearest centroid. As for *p* GSCM, the items are assigned to each bin based on some heuristics.

6. Concluding remarks

We have presented a practical version of general stochastic clustering method (GSCM) for automatic cluster discovery in datasets, which has the following characteristics:

- It is relatively simple compared to the state-of-the-art ant-based clustering method, ATTA.
- It does not assume any data distribution and hence is suitable in situations when little is known about a dataset.
- It does not require users to input the number of clusters and is able to automatically derive the number of clusters in a dataset.

We have demonstrated that the practical version of GSCM (*p* GSCM) works well with real-world clustering problems. *p* GSCM performs better than a number of existing methods in terms of clustering accuracy and runtime performance. These existing methods represent different approaches to cluster analysis. When tested using a diverse range of datasets, these methods tend to produce mixed results—they excel or fail under different conditions. In contrast, *p* GSCM performs consistently well across most of the datasets.

Our theoretical analysis showed that *p* GSCM has linear time and space complexities. This analysis is confirmed by our experiments using large datasets, where we have demonstrated that *p* GSCM scales up well on large data. Our results also showed that three existing methods either took too long to run or ran out of memory space when confronted with large data.

We have shown that the overall clustering performance of *p* GSCM is not overly sensitive to changes in the key parameter settings. Note that many existing methods would not perform well if they were to use a fixed set of parameter settings for the 44 datasets used in our experiments.

So far we have shown only one practical way to implement GSCM; and there are other possible ways to implement GSCM. In the near future, we would like to investigate the use of a 2D grid as the workspace so as to offer cluster visualisation on the grid.

Acknowledgements

The authors thank Julia Handl for providing the source codes of ATTA and the initial research idea that leads to this work. The first author was partially supported by Monash Postgraduate Publication Award.

References

[1] A.P. Appel, A.A. Paterlini1, E.P.M. de Sousa, A.J.M. Traina, J.C. Traina, A density-biased sampling technique to improve cluster representativeness, Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases, Lecture Notes in Computer Science, vol. 4702, Springer-Verlag, 2007, pp. 366–373.
[2] A. Asuncion, D.J. Newman, UCI Machine Learning Repository, School of Information and Computer Science, University of California, Irvine, 2007.
[3] E. Bonabeau, M. Dorigo, G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, New York, 1999.
[4] X. Cui, J. Gao, T.E. Potok, A flocking based algorithm for document clustering analysis, Journal of Systems Architecture 52 (2006) 505–515.
[5] R.J. Cho, M.J. Campbell, E.A. Winzeler, L. Steinmetz, A. Conway, L. Wodicka, T.G. Wolfsberg, A.E. Gabrielian, D. Landsman, D.J. Lockhart, R.W. Davis, A genome-wide transcriptional analysis of the mitotic cell cycle, Molecular Cell 2 (1) (1998) 65–73.
[6] D.L. Davies, D.W. Bouldin, A cluster separation measure, IEEE Transaction on Pattern Analysis and Machine Intelligence 1 (4) (2000) 224–227.
[7] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, Journal of the Royal Statistical Society Series B 39 (1) (1977) 1–38.

[8] J.-L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, L. Chrétien, The dynamics of collective sorting: robot-like ants and ant-like robots, Proceedings of the First International Conference on Simulation of Adaptive Behaviour: From Animals to Animats, vol. 1, MIT Press, Cambridge, 1991, pp. 356–363.
[9] J. Dunn, Well separated clusters and optimal fuzzy partitions, Journal of Cybernetics 4 (1975) 95–104.
[10] L. Ertoz, M. Steinbach, V. Kumar, Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data, in: Proceedings of the Second SIAM International Conference on Data Mining, San Francisco, CA, USA, 2003.
[11] P. Fabien, A. Hanene, V. Gilles, G. Christiane, A new approach of data clustering using a flock of agents, Evolutionary Computation 15 (3) (2007) 345–367.
[12] G. Folino, G. Spezzano, An adaptive flocking algorithm for spatial clustering, Proceedings of Seventh International Conference on Parallel Problem Solving from Nature. Lecture Notes in Computer Science, vol. 2439, Springer-Verlag, 2002, pp. 924–933.
[13] L. Gaubert, P. Redoua, P. Harroueta, J. Tisseaua, A first mathematical model of brood sorting by ants: functional self-organization without swarm-intelligence, Ecological Complexity 4 (4) (2007) 234–241.
[14] J.H. Gennari, P. Langley, D. Fisher, Models of incremental concept formation, Journal of Artificial Intelligence 40 (1989) 11–61.
[15] T.R. Golub, D.K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J.P. Mesirov, H. Coller, M.L. Loh, J.R. Downing, M.A. Caligiuri, C.D. Bloomfield, E.S. Lander, Molecular classification of cancer: class discovery and class prediction by gene expression monitoring, Science 286 (1999) 531–537.
[16] G. Hamerly, C. Elkan, Learning the *k* in *k*-means, Advances in Neural Information Processing Systems 16 (2004).
[17] J. Handl, Ant-based methods for tasks of clustering and topographic mapping: extensions, analysis and comparison with alternative methods. Master's Thesis, University of Erlangen-Nuremberg, Germany, 2003.
[18] J. Handl, J. Knowles, An evolutionary approach to multiobjective clustering, IEEE Transactions on Evolutionary Computation 11 (1) (2007) 56–76.
[19] J. Handl, J. Knowles, M. Dorigo, Ant-based clustering and topographic mapping, Artificial Life 12 (1) (2006) 35–61.
[20] J. Handl, B. Meyer, Ant-based and swarm-based clustering, Swarm Intelligence 1 (2007) 95–113.
[21] A.K. Jain, M.N. Murty, P.J. Flynn, Data clustering: a review, ACM Computing Surveys 31 (3) (1999) 263–323.
[22] D. Jiang, C. Tang, A. Zhang, Cluster analysis for gene expression data: a survey, IEEE Transaction on Knowledge and Data Engineering 16:11 (2004) 1370–1386.
[23] I. Kärkkäinen, P. Fränti, Dynamic local search for clustering with unknown number of clusters, in: Proceedings of the International Conference on Pattern Recognition, IEEE Computer Society, 2002, pp. 240–243.
[24] L. Kaufman, P.J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, John Wiley and Sons, 1990.
[25] T. Kohonen, Self-Organization and Associative Memory, Springer-Verlag, Berlin, 1984.
[26] P. Kuntz, D. Snyers, P. Layzell, A stochastic heuristic for visualizing graph clusters in a bi-dimensional space prior to partitioning, Journal of Heuristics 5 (3) (1998) 327–351.
[27] N. Labroche, C. Guinot, G. Venturini, Fast Unsupervised Clustering with Artificial Ants, Proceedings of Parallel Problem Solving from Nature VIII, Lecture Notes in Computer Science, vol. 3242, Springer-Verlag, 2004, pp. 1143–1152.
[28] E. Lumer, B. Faieta, Diversity and adaptation in populations of clustering ants, Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3, vol. 1, MIT Press, Cambridge, 1994, pp. 501–508.
[29] L. MacQueen, Some methods for classification and analysis of multivariate observations, Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, University of California Press, Berkeley, 1967, pp. 281–297.
[30] M.A.T. Figueiredo, A.K. Jain, Unsupervised learning of finite mixture models, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (2000) 381–396.
[31] M. Martin, B. Chopard, P. Albuquerque, Formation of an ant cemetery: swarm intelligence or statistical accident? Future Generation Computer Systems 18 (7) (2002) 951–959.
[32] N. Monmarché, M. Slimane, G. Venturini, On improving clustering in numerical databases with artificial ants, Proceedings of the Fifth European Conference on Artificial Life (ECAL'99). Lecture Notes in Artificial Intelligence, vol. 1674, Springer-Verlag, 1999, pp. 626–635.
[33] S. Monti, P. Tamayo, J. Mesirov, T. Golub, Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data, Machine Learning 52 (1–2) (2003) 91–118.
[35] A. Nanopoulos, Y. Theodoridis, Y. Manolopoulos, Indexed-based density biased sampling for clustering applications, IEEE Transaction on Data and Knowledge Engineering 57 (1) (2006) 37–63.
[36] D. Pelleg, A.W. Moore, *x*-means: extending *k*-means with efficient estimation of the number of clusters, in: Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann, 2000, pp. 727–734.
[37] W.W. Piegorsch, Sample sizes for improved binomial confidence intervals, Computational Statistics and Data Analysis 46 (2004) 309–316.
[38] V. Ramos, A. Abraham, Evolving a stigmergic self-organized data mining, in: Proceedings of the Fourth International Conference on Intelligent Systems, Design and Applications. Budapest, Hungary, 2004, pp. 725–730.
[39] P.J. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, Journal of Computational and Applied Mathematics 20 (1987) 53–65.
[40] G.E. Schwarz, Estimating the dimension of a model, Annals of Statistics 6 (2) (1978) 461–464.
[41] A.I. Su, M.P. Cooke, K.A. Ching, Y. Hakak, J.R. Walker, T. Wiltshire, A.P. Orth, R.G. Vega, L.M. Sapinoso, A. Moqrich, A. Patapoutian, G.M. Hampton, P.G. Schultz, J.B. Hogenesch, Large-scale Analysis of the Human and Mouse Transcriptomes, Proceedings of the National Academy of Sciences of the United States of America 99 (7) (2002) 4465–4470.
[43] P.N. Tan, M. Steinbach, V. Kumar, Introduction to Data Mining, Addison-Wesley Longman Publishing Co. Inc., Boston, USA, 2005.
[44] S.C. Tan, K.M. Ting, S.W. Teng, Reproducing the results of ant-based clustering without using ants, in: Proceedings of Congress on Evolutionary Computation 2006, IEEE Press, 2006, pp. 1760–1767.
[45] S.C. Tan, K.M. Ting, S.W. Teng, Examining Dissimilarity Scaling in Ant Colony Approaches to Data Clustering, Proceedings of The Third Australian Conference in Artificial Life, Lecture Notes in Artificial Intelligence, vol. 4828, Springer-Verlag, 2007, pp. 270–281.
[47] S.C. Tan, K.M. Ting, S.W. Teng, A practical stochastic clustering method. GSIT Technical Report Series (TR2009/3), Monash University, Australia, 2009.
[48] S.C. Tan, Approaches to simplify and improve swarm-based clustering. Ph.D. Thesis, Monash University, Australia, 2009.
[49] R. Tibshirani, G. Walther, T. Hastie, Estimating the number of clusters in a dataset via the Gap statistic, Technical Report 208, Department of Statistics, Standford University, CA.
[50] C. van Rijsbergen, Information Retrieval, second ed., Butterworths, London, UK, 1979.
[51] X. Wen, S. Fuhrman, G.S. Michaels, D.B. Carr, S. Smith, J.L. Barker, R. Somogyi, Large-scale temporal gene expression mapping of central nervous system development, Proceedings of the National Academy of Sciences of the United States of America 95 (1) (1998) 334–339.
[52] E.B. Wilson, Probable inference the law of succession and statistical inference, Journal of the American Statistical Association 22 (158) (1927) 209–212.
[53] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools with Java Implementations, Morgan Kaufmann, San Francisco, 2000 Available from ⟨http://www.cs.waikato.ac.nz/ml/weka⟩.
[54] Y. Yang, S.M. Kamel, An aggregated clustering approach using multi-ant colonies algorithms, Pattern Recognition 39 (7) (2006) 1278–1289.
[55] K.Y. Yeung, Cluster analysis of gene expression data. Ph.D. Dissertation, Computer Science Department, University of Washington, 2001.
[56] K.Y. Yeung, D.R. Haynor, W.L. Ruzzo, Validating clustering for gene expression data, Bioinformatics 17 (2001) 309–318.

Swee Chuan Tan received his Ph.D degree from Monash University in May 2009. He is currently a Lecturer in SIM University and his research focuses on swarm-based clustering and anomaly detection.

Kai Ming Ting received his Ph.D degree from Sydney University. He is currently an Associate Professor in Monash University and his research focuses on data mining, machine learning techniques and content-based image retrievals.

Shyh Wei Teng received his Ph.D degree from Monash University. He is currently a Senior Lecturer in Monash University and his research focuses on data mining, machine learning techniques and content-based image retrievals.