Sādhanā (2020)45:66
https://doi.org/10.1007/s12046-020-1297-4

Accelerated Single-Linkage algorithm using triangle inequality

PAYEL BANERJEE[1,*], AMLAN CHAKRABARTI[2] and TAPAS KUMAR BALLABH[1]

[1]Department of Physics, Jadavpur University, Kolkata, India
[2]Calcutta University, Kolkata, India
e-mail: payelbanerjee54@gmail.com; achakra12@yahoo.com; tkb@phys.jdvu.ac.in

MS received 21 February 2019; revised 24 August 2019; accepted 10 January 2020

Abstract. Single-Linkage algorithm is a distance-based Hierarchical clustering method that can find arbitrary-shaped clusters but is most unsuitable for large datasets because of its high time complexity. The paper proposes an efficient accelerated technique for the algorithm with a merging threshold. It is a two-stage algorithm with the first one as an incremental pre-clustering step that uses the triangle inequality method to eliminate the unnecessary distance computations. The incremental approach makes it suitable for partial clustering of streaming data along with the collection. The second step using the property of the Single-Linkage algorithm itself takes a clustering decision without comparing all the patterns. This method shows how the neighbourhood between the input patterns can be used as a tool to accelerate the algorithm without hampering the cluster quality. Experiments are conducted with various standard and large real datasets and the result confirms its effectiveness for large datasets.

Keywords. Single-Linkage clustering; merging threshold; Leader algorithm; triangle inequality; large datasets.

1. Introduction

Dealing with gigantic datasets has always been a tough job for scientists, researchers and other experts in the field of Computer Science. However, after an era of handling the challenges of data collection, nowadays, a new problem has arisen about how to process this huge amount of data after collection. To extract information from this avalanche of data, it is important to use a powerful data mining tool for knowledge discovery.

Clustering is considered as one of these strong data mining tools for this purpose. It aggregates or groups a particular set of objects such that the interconnection and bonding are strong among the objects of the same cluster and is weak among the objects of different clusters [1]. In other words, good clusters must attain high intra-cluster resemblance and low-inter cluster similarity. Clustering is not a specific algorithm but can be achieved by various methods that differ significantly in the definition of similarity. It is a vital task of data mining that finds applications in various fields like machine learning, pattern recognition, image processing, spatial data analysis, bioinformatics, information retrieval, data compression, etc. [2–4]. While handling a large dataset, clustering and classification techniques help in better data processing as now, rather than dealing with all the patterns in the dataset, only clusters are addressed, which reduces the complexity in handling high data volume. However, some clustering algorithms suffer from high computational complexity and the convergence time of the algorithm consequently increases with an increase in data size. Thus, from the very beginning, researchers are trying to handle the complexity and computational cost of the clustering algorithms and consequently to increase their scalability and the speed. Emersion of big data has added more challenges to this problem. Clustering can be broadly identified into two main types, called Partitional and Hierarchical. Unlike Hierarchical clustering, Partitioned clustering technique usually needs an a priori specification of the cluster number, and an improper specification of the cluster number can lead to an erroneous result. For analysing biological data, Hierarchical clustering algorithms are widely used in various biological applications like phylogeny reconstruction [5], protein structure prediction [6], analysis of genomic data [7], etc.

Single-Linkage (SL) algorithm is one such Hierarchical clustering method that suffers from high time complexity, which makes it unsuitable for clustering large data volume. It also requires the entire dataset in advance to initiate the algorithm. Distance computation between input patterns is the most computationally expensive part of any clustering algorithm, which is likely to increase with an increase in the size of the input dataset. ‘Triangle inequality’ is a widely used technique to avoid redundant distance computations without affecting the result of the algorithm. Triangle

*For correspondence

Published online: 12 March 2020

inequality has been previously used in speeding up various other popular algorithms like K means [8], DBSCAN [9], Average Linkage Hierarchical algorithm [10], etc. Hence, our paper proposes a suitable technique of applying triangular inequality in SL clustering algorithm to make it more suitable for high data volume. We have utilized the concept of the accelerated version of the Leader algorithm [10] as a pre-clustering step and have proposed a final merging step, which without computing the pair-wise distance between all the input patterns takes a clustering decision. The paper reveals the dependence of the speed of the algorithm on the proximity of the data points by utilizing the property of the SL algorithm itself to accelerate the method.

Novelty of the proposed algorithm:

The algorithm is an improved version of the SL algorithm, which does not require the entire dataset in advance unlike some of its well-known variants.

- It does not require computing the distance between all the data points to take a clustering decision and hence saves a lot of time by avoiding the computation of distance matrix, unlike the other methods as discussed in section 2.
- The proposed algorithm has employed the triangular inequality method in the pre-clustering stage to avoid redundant distance computations, thereby speeding up the algorithm.
- The pre-clustering stage compresses the large input data and the second stage applies the clustering algorithm on these compressed data, thereby making the process easier and faster.
- The final stage also avoids the pair-wise distance computations between all data points to take a clustering decision, unlike the original SL algorithm.
- The algorithm does not compromise the quality of clusters under any circumstances although speeds up the convergence to a larger extent as confirmed by experimental results.

Thus, SL algorithm, which is unsuitable for clustering large datasets, can regain its suitability by using this modified method without compromising the cluster quality.

The paper is organized as follows. Section 2 gives the background and the related work of this study, section 3 explains the methodology of the proposed algorithm, section 4 gives the time and space complexity of the algorithm, section 5 explains the advantages of the algorithm, section 6 shows the experimental results and section 7 finally presents the conclusions and future scopes of the research.

2. Related work

Hierarchical clustering algorithm, also called connectivity-based algorithm, merges 'objects' to 'clusters' in the hierarchical form on the basis of the distance between them [11]. Depending on how the clusters are created, there are two types of Hierarchical clustering algorithms, Agglomerative Hierarchical clustering algorithm also called AGNES (agglomerative nesting) and Divisive Hierarchical clustering algorithm also called DIANA (divisive analysis) [12].

The distances between each pair of clusters are computed to find the pair of clusters with the most similarity. Now there are various standard alternatives to measure the inter-cluster distances. Different linkage methods [13] are used for different applications and hence a linkage method most suitably used for an application may yield poor clusters for some other kind of applications. Lance and Williams proposed the Lance–Williams dissimilarity update formula, which gives the distance between a cluster and any other object or cluster. According to the formula, the distance between a cluster '$i \cup j$' and any other object or cluster 'k' is given by

$$d(i \cup j, k) = \alpha_i d(i, k) + \alpha_j d(j, k) + \beta d(i, j) + \gamma |d(i, k) - d(j, k)| \quad (1)$$

where α_i, α_j, β and γ are the criteria of agglomeration.

SL clustering algorithm [13, 14]: It is also called as a 'nearest neighbours' method with $\alpha_i = 0.5$, $\alpha_j = 0.5$, $\beta = 0$ and $\gamma = -0.5$. As its name suggests this method defines the distance between two clusters, as the minimum distance found between all the objects from the first cluster to all the objects of the second cluster. In other words, in SL clustering, two clusters are merged if the minimum of their inter-cluster distances is within the user-defined threshold.

For example, if cluster 1 contains objects a, b, c and cluster 2 contains objects d, e, f then the distance between cluster 1 and cluster 2 is the minimum of the following cases:

dist(a, d), dist(a, e), dist(a, f), dist(b, d), dist(b, e), dist(b, f), dist(c, d), dist(c, e), dist(c, f).

Hence, if two points i and j are grouped into a cluster ($i \cup j$) then the dissimilarity between the cluster ($i \cup j$) and an object 'k' is $d(i \cup j, k) = \min\{d(i, k), d(j, k)\}$ [13]. Thus, the distance measure between two clusters is the only thing that differentiates an SL Hierarchical clustering from other Agglomerative Hierarchical clustering algorithms.

When a threshold criterion is applied to each merging step, then the most similar clusters are merged if and only if the distance between them is less than or equal to the user-defined threshold. Hence, the minimum distance between two merged clusters must be less than or equal to the user-defined threshold value.

SL algorithm has a disadvantage as it suffers from chaining effect [15] because of its notion of similarity, leading to the formation of loosely bound, elongated clusters with low intra-cluster similarity. Despite the chaining effect problem, SL algorithm is still a widely used tool for an early-stage discovery of knowledge of various types of datasets and is highly beneficial in detecting clusters of arbitrary shapes [16, 17].

The classical SL algorithm in its first iteration needs to compute the similarity between all pairs of individual points in the dataset. The proximity matrix requires storing $\frac{n^2}{2}$ proximities (assuming the matrix to be symmetric) where 'n' is the number of data points [18].The overall time complexity of this algorithm if done in a naïve way is $O(n^3)$ [14].

R Sibson gives an SL algorithm of $O(n^2)$ time complexity and $O(n)$ space complexity and provides an improvement upon the naïve implementation of SL clustering, which provides a time and space complexity of $O(n^3)$ and $O(n^2)$, respectively [14, 19]. However, when a threshold criterion is provided then the algorithm should converge with a lower number of iterations and this may decrease the overall time requirement.

As Minimum Spanning Tree (MST) can be closely linked with SL algorithm, Gower and Ross (1969) proposed a method of computing SL clusters using MST [14, 20]. The usual algorithms require an $O(n^2)$ effort for computing MST from a dissimilarity matrix. Therefore, the total complexity for getting SL clusters is thereby $O(n^2)$.

All these algorithms are not only expensive for clustering huge data but are also unsuitable for streaming data clustering as they require the entire database in advance.

A fast SL clustering method based on Tolerance Rough Set Model has been proposed by Patra and Nandi [21] with time complexity less than $O(n^2)$ but not providing exactly identical results as those by the classical SL method.

Jin *et al* [16] present a distributed SL Hierarchical clustering algorithm (DiSC) based on Map-Reduce. It divides the original problem into a set of overlapped subproblems, and solves each sub-problem separately and then combines the solution of each sub-problem into an overall solution. It involves parallelization schemes to achieve higher speed-ups.

Although the algorithm proposed by Patra *et al* [17] reduces the time complexity below $O(n^2)$, the convergence time may be high if the distance among the majority of the data is more than half the user-defined threshold. The first phase of the method uses the Leader algorithm with a threshold of half the user-defined value. With decreasing threshold, a large number of leaders are likely to be produced if the proximity of the patterns becomes low with respect to the used threshold (half the user-defined value). If the number of leaders in the first phase of the clustering algorithm increases then the time required in processing these leaders for getting final clusters also increases.

Our proposed method has utilized the idea [17] of applying the Leader algorithm in the SL method to show how, depending on the proximity of the input patterns, SL algorithm can be accelerated. In other words, our algorithm has utilized the neighbourhood of clusters as a tool to speed up the algorithm. The algorithm removes the disadvantage of requiring the entire dataset in advance and is effective for clustering large datasets. It helps in speeding up the algorithm without using parallelization schemes. It does not sacrifice cluster quality in the process and provides identical clusters as provided by the classical SL method.

3. Accelerated SL algorithm

The proposed algorithm is a two-stage method where the first stage involves a pre-clustering algorithm followed by the second stage, which produces the final clusters.

3.1 *Pre-clustering algorithm*

The algorithm has been divided into two clustering methods. At the first stage, the algorithm uses an incremental pre-clustering technique that partially clusters the input data by scanning the dataset only once. Here, we have applied the well-known Leader algorithm as the pre-clustering step. To further reduce the distance computation of the Leader algorithm, its accelerated form using triangular inequality is used. The algorithm assigns the first input data as the leader of the first cluster. The next step is to find the distance of the next incoming data with the first cluster. Here, comparing a data to a cluster implies its comparison to the leader of that cluster. In other words, the leader of every cluster acts as the reference point of that cluster. If the data is found to lie within the threshold distance of that cluster then assign the data as its follower, else assign it as a new leader if no more clusters are left to be compared. The data is compared to all the existing clusters unless and until it is assigned as a follower or as a leader. The process continues unless and until all the data points are scanned.

Here, the Leader algorithm follows a *priority rule of comparison* depending on the time of formation of leaders. In this case, a leader that forms earlier gets higher priority of being compared to a data point. Hence, if there are three leaders and all are formed following the order 1st $\rightarrow$ 2 nd $\rightarrow$ 3rd then the first leader will get the foremost chance of being compared to a data point followed by the second and third leader. Therefore all the existing leaders will get priority of being compared to a data point according to their order (on time scale) of formation. However, if during this process a data gets merged with an existing cluster then it is not required to be compared to the other clusters and the entire process will start with the next data point.

3.1a *Accelerated Leader algorithm.* The main problem of the Leader algorithm is its increasing number of distance computations when a data point does not lie within threshold from any of the clusters or lies within threshold only with the mth cluster for 'm' existing clusters. In these cases the data point has to be compared to all the existing leaders, delaying the convergence of the algorithm. The distance computations seem to increase with the increase in data points and also with its increase in dimensionality.

Hence the Leader algorithm is unsuitable in the case for large data volume and also for high-dimensional data. To get rid of this problem, the idea of the triangular inequality of the accelerated Leader algorithm proposed in [10] is used in the pre-clustering step for our algorithm.

It provides a bounding interval of the distance between a couple of data points 'a' and 'b' such that

$$\forall a, b, c \in D: |d(a, b)| <= |d(a, c)| + |d(b, c)|. \quad (2)$$

Here '*d*' is a suitable metric over the domain D. Equation (2) provides an upper bound for the distance *d*(a, b).

The 'reverse triangular inequality' provides a lower bound to the distance between 'a' and 'b' using the equation

$$d(a, b) >= |d(a, c) - d(b, c)| \quad (3)$$

The following lemma shows how the 'reverse triangle inequality' can be used to obtain a lower bound for a couple of data points:

Lemma 1 *For any three data points a, b, c, if d(b, c) >= 2 d(a, c) then d(a, b) >= d(a, c).*

Proof Putting the assumption for *d*(b, c) on the RHS of Eq. (3), we get *d*(a, b) >= *d*(a, c).

The accelerating Leader algorithm uses this reverse triangular inequality to eliminate the unnecessary distance computations of the Leader algorithm.

Let 'x' be any data point and 'l_1', 'l_2' be two leaders with 'l_1' being the first generated leader of the leader set (L). Hence, replacing a, b, c of Lemma 1 with 'x', 'l_2', 'l_1,' respectively, one can reach a conclusion about whether $d(x, l_2)$ is >= $d(x, l_1)$ or not.

A data point is assigned as a leader only after being compared to the existing leaders and its exact distance is always calculated from the first generated leader. Hence, both $d[l_1, l_2]$ and $d[x, l_1]$ are already available for computing $d_{min}(x, l_2)$.

A data in the pre-clustering algorithm is compared to the first generated leader before being compared to any other leaders as per the 'priority rule' given in section 3.1. A data gets a chance of being computed with the other leaders only if it lies at a distance that is more than the threshold from the first generated leader. Considering l_1 as the first generated leader, if $d_{min}(x, l_2)$ is found to be greater than $d(x, l_1)$ then it is not required to compare 'x' to 'l_2' to find its exact distance as it is obvious that $d(x, l_2)$ is greater than threshold. Hence, this algorithm has eliminated the need of all the distance computations of the data point from those leaders 'l_j' for which $d_{min}(x, l_j) >= d(x, l_1)$. For any leader '$l_j$', if $d_{min}(x, l_j) < d(x, l_1)$ then the exact distance of 'x' from 'l_j' is required to be computed for taking a clustering decision.

This process of comparison is repeated with all the available leaders unless and until 'x' is assigned as a new leader or as a new follower. The accelerated Leader method is depicted in Algorithm 1.

Algorithm 1:

```
Let $l_i \in L$ ($i = 1$) be the first scanned pattern, '$T$' being the user-defined threshold.
  1. Assign the first input pattern 'x' as $l_i$ with $i = 1$. L $\rightarrow \{l_1\}$
  2. For any input pattern $x \in D$, if $d(x, l_1) > T$, assign 'x' as a new leader $l_i$ for $i = i+1$.
  3. Merge $l_i$ with L such that $L = L \cup l_i$.
  4. Store $d(l_1, l_i)$.
  5. If all 'x' are scanned go to '16' else go to '6'.
  6. For any new input pattern $x \in D$, compute $d(x, l_1)$.
  7. If ($d(x, l_1) > T$) then go to step '9' else go to step '8'.
  8. Assign 'x' as a follower to $l_1$. Go to '5'.
  9. For any unchecked leader $l_i \in L$, if $d(l_i, l_1) >= 2d(x, l_1)$ then reverse triangle
     inequality says $d(x, l_i) >= d(x, l_1)$.
 10. Repeat '9' with each $l_i \in L$ and go to '11' if '9' holds true for all $l_i \in L$. Go to '14' if
     '9'fails for any $l_i$.
 11. Assign x as a new leader $l_i$ for $i = i+1$.
 12. Store $d(x, l_1)$.
 13. $L = L \cup l_i$. Go to 5.
 14. Compute $d(x, l_i)$ for the corresponding $l_i$.
 15. If $d(x, l_i) <= T$, assign x as the follower of $l_i$ and go to 5 else repeat from 9.
 16. Output all the leaders with followers.
```

In the SL algorithm, ∀a∈ A and ∀b∈ B where A and B are two distinct clusters, all A and B are merged if $d_{\min}$(a, b) <= *T*. Hence, the final set of clusters has the following two properties.

(i) The minimum intra-cluster distance is within the threshold.
(ii) The minimum inter-cluster distance is always greater than the threshold.

The pre-clustering algorithm takes a clustering decision only if the input pattern comes within threshold with a leader. It may be possible that the distance of the merged object is either greater than the threshold distance from all or any of the other elements (except the leader) of the cluster or it may be within the threshold distance from all or any of them. In either case the minimum intra-cluster distances of every cluster remain within the threshold limit, thus satisfying the first criterion of the SL clusters.

As the Leader algorithm is an incremental one and scans every data point only once, once the entire dataset is scanned the algorithm stops doing further merging of clusters. It may be possible that the minimum inter-cluster distance between any two clusters can be still less than the threshold and thereby needs to get merged; otherwise, the second property of the SL clusters will not be satisfied. This function is performed by the second stage of the algorithm.

All incremental algorithms are sensitive to the ordering of the data points and so is the pre-clustering algorithm. As the second stage performs the final merging of the pre-clusters following the property of SL clusters, it does not allow the final quality of the clusters to get affected by the ordering of the input data. Hence, for an input dataset, the output will remain the same, irrespective of the variation in the ordering of the input data.

3.2 *The second stage of the algorithm*

The second half of the algorithm creates a neighbourhood matrix but does not fill all the cells by computing the pair-wise distance between all the data points. It creates an '$m - 1 \times 1$' matrix for 'm' pre-clusters provided by the first half of the proposed method. Each cell of the matrix is assigned a flag, which represents the neighbourhood between the clusters corresponding to that particular column and row. Every time two clusters are found in the threshold the flag of that cell is set high, else low.

The operation starts with a single column corresponding to any cluster and scans the entire column to find all the neighbours of that cluster. For a cluster C_j (corresponding to the column), 'C_i' (corresponding to a row) will be a neighbour if $d_{\min}(C_j, C_i)$ is within the threshold. The flag of the cell corresponding to that 'C_j' and 'C_i' is set high if they are found to be neighbours.

After scanning the entire column, 'C_j' is merged with any 'C_i' for which the flag is set high. The row corresponding to this 'C_i' is removed and now 'C_j' will be replaced by '$C_j UC_i$'. The contents of the other cells will remain unchanged. If, at every iteration, the newly merged 'C_i' is denoted by 'C_m' then the cluster representing the column will now be updated as 'C_j' = 'C_{j} U C_m', i.e. 'C_j' has now included a new cluster 'C_m' as its member. After merging 'C_m' with 'C_j', the row corresponding to 'C_m' will be removed from the matrix. At the next step, we have utilized the property of the SL algorithm to speed up the method. SL algorithm merges clusters only if the minimum distance between them lies within the user-specified threshold. In other words, SL algorithm refers to two clusters as neighbours if 'at least' a single pair-wise distance between two clusters is found to lie within the threshold limit. To best utilize the word 'at least', it will check the neighbourhood between 'C_m' and those unmerged 'C_i's for which the flag is still low. This is because those 'C_i's for which the flag is already high are already considered a neighbour to 'C_j' and will get merged to it at any later stage. Hence, checking the neighbourhood of those 'C_i's from 'C_m' (which is itself a part of 'C_j') again will lead to unnecessary distance computations. The flags of all the cells that are found to be a neighbour to 'C_m' will go high. After checking the neighbourhood of all the unmerged 'C_i's (with flag low) from 'C_m', 'C_j' will be merged to any of the 'C_i' of that column for which the flag is high. The newly merged 'C_i' will be then assigned as a 'C_m'. The row corresponding to the 'C_m' will be removed.

This scanning process for a particular 'C_m' will stop after checking its neighbourhood from all the 'C_i's with flag low. After the end of the scanning process for a particular 'C_m', the process repeats with a new C_i (with flag high) as C_m. This assignment of C_m to a C_j will continue unless and until no more 'C_m's can be assigned to that C_j. When such a condition is reached, following three operations are performed:

i. The C_j with all its merged 'C_m's will be considered as a fully grown cluster.
ii. If, for a column, only one 'C_i' is left with its flag low then that will also be identified as a fully grown cluster and the corresponding row will be removed.
iii. If the flags of more than one 'C_i' are still low, then the entire process will be repeated from the beginning with any of these 'C_i's as a new 'C_j'.

The algorithm will converge when all the rows are removed.

This entire clustering principle used in the second stage is depicted in Algorithm 2.

Algorithm 2.

```
$C_j$ = $P_i$ for $i$ = 1, /$P_1$ be the first pre-cluster/
C*= {$C_j$} /C* be the final set of clusters/
$C_i$= $P_i$ for all 'i's for which $C_i \neq C_j$
$C_m$= $C_j$
$j$ = 1
$k$ = $j$+1
no_of_$C_i$_with_flag_high = 0;
While(all rows are not removed)
          {
      While(all $C_i$s with flag low are not compared with current $C_m$)
          {
          If d($C_m$, $C_i$) <= $T$ then
              {
               set flag _ $C_i$ high
               no_of_$C_i$_with_flag_high = no_of_$C_i$_with_flag_high +
1;
               }
           else
               {
                set flag _ $C_i$ low
               }
             }
      If (no_of_$C_i$_with_flag_high > 0)
            {
           for (any $C_i$ with flag high)  do
               {
               $C_j$ = $C_j$U$C_i$
               $C_m$ = $C_i$
               no_of_$C_i$_with_flag_high = no_of_$C_i$_with_flag_high -
1;
               remove $C_m$ row from matrix
               }
            }
      else
             {
             for (any $C_i$ with flag low) do
                  {
                  $j$ = $k$
                  $k$ = $k$ + 1
                  Assign $C_j$ = $C_i$
                  $C_m$ = $C_j$
                  C*= C*U$C_j$
                  Remove $C_m$ row from matrix
                  }
             }
        }
Output C *
```

4. Time and space complexity of the proposed method

4.1 *Calculation of time and space complexity of accelerating Leader algorithm*

With 'm' and 'n' being the size of the leader set and the input data, respectively, the time complexity of the accelerated Leader algorithm is calculated as follows:

$$= [(1+2+3+4+\cdots(m-1)]+[(n-m)(m)] \quad (4)$$

$$= \left[\left(\frac{m-1}{2}(2+(m-1-1))\right)\right]+\left[\left(nm-m^2\right)\right] \quad (5)$$

$$= \frac{(m^2-m)}{2}+(nm-m^2) \quad (6)$$

$$= \frac{(m^2-m)+(2nm-2m^2)}{2} \quad (7)$$

$$= \frac{2nm-m^2-m}{2}. \quad (8)$$

The first expression within [ ] in Eq. (4) gives the time required for creating 'm' leaders and the second one provides the time required for assigning 'n' data points as followers of 'mth' leader. Here '$n-m$' points have been assigned to the m^{th} leader to increase the number of distance computations to get the exact time complexity in the worst case. Hence, the time complexity of the accelerating Leader algorithm is O(mn) as shown by Eq. (8). The clusters provided by the pre-clustering algorithm require O(n) space and the distance of the leaders from the first generated leader occupies O($m - 1$) space, which results in a total space complexity of O(n).

4.2 *Time complexity of the second stage*

To calculate the time complexity of the second stage of the algorithm, let us make three assumptions:

a. All the pre-clusters have an average number of members, say 'P'. This assumption is made to make the calculation less complex.
b. At every iteration, whenever a column is scanned by a C$_i$ (with flag low), 'f' neighbours can be obtained.
c. At every iteration, whenever a column is scanned by a C$_i$ (with flag high), no neighbours can be obtained.

Assumption 'c' helps in getting the maximum possible number of distance computations that our algorithm can undergo. This is because if more and more neighbours are found on a single scan, a lesser number of distance computations will be done at the next iterations.

For 'm' pre-clusters there are 'm − 1' rows with the mth cluster being assigned as 'C$_j$'.

Hence, to scan the column following these assumptions, the number of distance computations for the first time scanning will be $P[m - 1]P$. This is because the comparison of 'C$_j$' with 'm − 1' clusters requires the pair-wise distance computations between all the points of 'C$_j$' with that of the 'm − 1' clusters in the worst case. If 'f' neighbours are reached, the next iteration requires distance computations between all the points of 'C$_m$' with that of the 'm − 1 − f' clusters. All the neighbours perform the scanning, resulting in f (m − 1 − f) computations. After completing scanning with all 'C$_i$'s with flag high, a new C$_i$ with flag low will begin scanning the remaining (m − 2 − f) 'C$_i$'s with flag low. This process will continue unless and until all rows are removed. The second stage avoids the leader to leader distance computations between a 'C$_j$' and all 'C$_i$'s as the leader to leader computations are already done at the pre-clustering stage.

The maximum number of distance computations required for comparing the pre-clusters following these conditions is

$$\begin{aligned}&[\{P(m-1)P\} + f\{P(m-1-f)P\} + \{P(m-2-f)P\}\\&+f\{P(m-2-2f)P\}+\{P(m-3-2f)P\}\cdots\cdots\cdots\\&+\{P(m-\eta-(\eta-1)f)P\} + f\{P(m-\eta-\eta f)P\}]\\&-[\{(m-1)\} + f\{(m-1-f)\} + \{(m-2-f)\}\\&+f\{(m-2-2f)\} + \{(m-3-2f)\}\cdots\cdots\\&+\{(m-\eta-(\eta-1)f)\} + f\{(m-\eta-\eta f)\}]\end{aligned} \tag{9}$$

$$\begin{aligned}&= [\{P^2(m-1)\} + \{P^2(m-2-f)\} + \{P^2(m-3-2f)\}\ldots\\&\ldots + \{P^2(m-\eta-(\eta-1)f)\}] + [f\{P^2(m-1-f)\}\\&+f\{P^2(m-2-2f)\} + f\{P^2(m-3-3f)\} + \cdots\cdots\cdots\cdots\\&+f\{P^2(m-\eta-\eta f)\}] - [\{(m-1)\} + \{(m-2-f)\}\\&+\{(m-3-2f)\}\cdots\cdots\cdots + \{(m-\eta-(\eta-1)f)\}]\\&-[f\{(m-1-f)\} + f\{(m-2-2f)\} + f\{(m-3-3f)\}\\&+\cdots\cdots + f\{(m-\eta-\eta f)\}]\end{aligned} \tag{10}$$

In the worst case, the process will run 'η' times for identification of 'f' set of neighbours, after which all rows are removed. In other words, after getting 'f' neighbours ηth time, no more C$_i$ with flag low will be left for being scanned.

$$m-\eta-\eta f = 0.$$

$$\frac{m}{1+f} = \eta$$

Applying A.P. series, and replacing 'η' in Eq. (10) gives

$$= \frac{P^2m^2}{2} - \frac{P^2m}{2}\frac{(1+f^2)}{1+f} - \frac{m^2}{2} + \frac{m}{2}\frac{(1+f^2)}{1+f} \tag{11}$$

$$= \frac{n^2}{2} - \frac{P^2m}{2}\frac{(1+f^2)}{1+f} - \frac{m^2}{2} + \frac{m}{2}\frac{(1+f^2)}{1+f} \tag{12}$$

The last two series in Eq. (10) give the number of distance computations saved by avoiding the leader to leader distance computation of a 'C$_j$' or 'C$_m$' from all the 'C$_i$'s with flag low. As more and more neighbours (f) are obtained on a single scan, lesser number of distances are computed by the algorithm.

Thus although the time complexity is $O(n^2)$ as shown by Eq. (12), the number of distance computations is highly dependent on 'f', which in turn depends on the data type and the user-defined threshold. Higher the proximity of the patterns in a dataset, larger the chance of getting neighbours on a single scan. The chance also increases with an increasing threshold. Thus, by utilizing the neighbourhood of the input patterns, the proposed algorithm has reduced the redundant distance computations required in traditional variants of SL. Moreover, to get a neighbour, it may not be required in practice to compute the pair-wise distances between all the data points of two clusters but can be stopped whenever a neighbour is identified. Hence, for any data and a suitable threshold where SL algorithm can provide good clusters, our algorithm will be definitely proved as a beneficial one.

Its usefulness has been verified experimentally in 'section 6' after being applied to various real-world datasets.

4.3 *Total time and space complexity of the algorithm*

The total time complexity of the algorithm is given by $O(mn + n^2) = O(n^2)$. However, the space complexity remains $O(n)$ to store 'n' data points of all the clusters and the status of the 'm-1' flags of the neighbourhood matrix.

5. Advantages of the algorithm over other well-known SL algorithms

Advantages of our algorithm

- The pre-clustering algorithm merges data to a cluster only if it comes in threshold to the leader of a cluster. In other words, merging a data to a cluster does not require the computation of its distance from all the members of that cluster. It should also be noted that the pre-clustering algorithm is itself an accelerated approach of the Leader algorithm and also saves a lot of unnecessary distance computations between an input pattern and all the leaders.
- The second stage of the algorithm computes only the inter-cluster distances of the provided pre-clusters and hence the proposed algorithm saves all the intra-cluster follower to follower distance computations under all circumstances and thereby converges faster in comparison with the other well-known variants.

Let us consider two cases to find the number of distance computations that can be saved by the algorithm for 'n' number of data points.

i. There are '$n-2$' pre-clusters with '$n-3$' being singletons and the remaining one of size three with two followers.
ii. There are 'm' pre-clusters with '$m-1$' being singletons and the remaining one has '$n-m+1$'members or '$n-m$' followers.

For case (i), the number of intra-cluster (follower to follower) distance computations is 1. This is because the largest pre-cluster has only two followers and all other pre-clusters are singletons. However, for case (ii), $\frac{(n-m)(n-m-1)}{2}$ follower to follower distance computations are saved. Thus, depending on the type of data and also on the threshold, the minimum number of distance computations that can be saved ranges between 1 and $\frac{(n-m)(n-m-1)}{2}$.

Moreover, if the second stage of the algorithm recognizes more than one 'C_i' as neighbour to a 'C_j' then those two 'C_i's will be merged without calculating any more inter-cluster distances among those 'C_i's and hence the algorithm will be able to merge two clusters even without measuring the distances between the members of those clusters.

Thus, although the time complexity of the proposed algorithm is mathematically $O(n^2)$, it is always an improved version of the SL algorithm (table 1) especially for large datasets as confirmed by the experimental results.

6. Experimental results

To show the performance of the proposed algorithm, we have implemented the Leader, accelerated Leader, proposed algorithm and the distance matrix methods using C language (ChIDE compiler) and executed on a PC with Intel(R) Core(TM) i3-6006U CPU @ 2.00-Ghz processor, 8-GB RAM and 64-bit OS. To show its effectiveness on real-world datasets, we conducted experiments with datasets of table 2, of varying sizes and dimensions as obtained from UCI machine learning repository [22] and the detailed results are provided in tables 3, 4 and 5. We have applied Rand Index (RI) [23] to compare the clustering techniques. The convergence time of the algorithms is calculated by taking an average of 1000 computations for each case to increase accuracy.

Table 2. Experimental dataset

Dataset	No. of instances	No. of attributes	Attribute type
Pendigits			
Testing	3498	16	Integer
Training	7494	16	Integer
Letter	20000	16	Integer
Shuttle	58000	9	Integer

Table 1. Comparison of the proposed algorithm with the other well-known variants

Features	Proposed algorithm	Classical method	Sibson's method	MST approach by Gower and Ross
Distance matrix computation required	No	Yes	Yes	Yes for creating MST
Requires all the data points at once	No	Yes	Yes	Yes
Time complexity	$O(n^2)$	$O(n^3)$	$O(n^2)$	$O(n^2)$
Convergence time	Lowest	High	Low	Low
Space complexity	$O(n)$	$O(n^2)$	$O(n)$	$O(n)$
Partial clustering of streaming data	Yes	No	No	No
Suitability of clustering large dataset	High	Lowest	Low	Low
Cluster quality	Same as original			

Table 3. Results for standard datasets

Dataset	Method	Threshold	Time (approx in s)	RI
Pendigits testing	Leader	0.1	931.399	1.0
	Acc. Leader		112.371	
	Leader	5	927.648	1.0
	Acc. Leader		198.599	
	Leader	10	909.646	1.0
	Acc. Leader		276.219	
	Leader	20	513.322	1.0
	Acc. Leader		226.807	
	Leader	30	191.462	1.0
	Acc. Leader		111.171	
Pendigits training	Leader	0.1	4343.854	1.0
	Acc. Leader		520.953	
	Leader	5	4426.636	1.0
	Acc. Leader		848.514	
	Leader	10	4279.032	1.0
	Acc. Leader		1165.177	
	Leader	20	2272.713	1.0
	Acc. Leader		951.735	
	Leader	30	746.385	1.0
	Acc. Leader		390.20	

Table 4. Results for large datasets

Dataset	Method	Threshold	Time (approx. in s)	RI
Letter	Leader	3	4343.699	1.0
	Acc. Leader		2983.384	
	Leader	3.5	2696.959	1.0
	Acc. Leader		1961.99	
	Leader	4	1510.311	1.0
	Acc. Leader		1190.581	
	Leader	4.5	946.510	1.0
	Acc. Leader		791.393	
	Leader	5	573.241	1.0
	Acc. Leader		512.118	
Shuttle	Leader	3.5	11609.90	1.0
	Acc. Leader		6099.195	
	Leader	4	8024.368	1.0
	Acc. Leader		2893.298	
	Leader	4.5	6153.844	1.0
	Acc. Leader		2174.786	
	Leader	5	4422.03	1.0
	Acc. Leader		1643.392	
	Leader	5.5	3250.847	1.0
	Acc. Leader		1233.483	

a) Tables 3 and 4 show the performance of the accelerated Leader algorithm for various real-world datasets and for varying thresholds, which are not shown by the authors [10]. It shows that in these cases also the accelerated Leader outperforms Leader algorithm by converging in a much shorter time without hampering cluster quality (RI = 1).

b) Table 5 shows that only the initial distance matrix computation converges in a higher time in comparison with the proposed accelerated SL algorithm and hence the latter proves itself to be a better approach as compared with those existing variants where distance matrix computation is necessary.

c) The RI of '1' in all cases of table 5 implies that the final cluster quality is not affected by our algorithm and is identical to that produced by the classical SL algorithm.

d) Table 5 shows that the algorithm provides poor results (few data have been clustered) when applied on dataset like Pendigits for certain thresholds as compared with Letter and Shuttle. This proves that the performance of SL algorithm is dataset dependent like those of all other clustering algorithms and hence the choice of clustering algorithm must be application dependent.

e) Figures 1 and 2 show that a large number of data points have already got clustered at the first stage and hence the second stage converges in a much shorter time as compared with the distance matrix methods. This proves that the pre-clustering algorithm is a suitable one for accelerating SL algorithm.

f) Figure 3 shows the percentage decrease in distance computations of the proposed algorithm from the distance matrix method. It can be seen that the improvement is more prominent with an increase in data size and hence accelerated SL algorithm is highly suitable for handling large data volume.

Table 5. Comparison of the convergence time of the proposed Accelerated Single Linkage algorithm and the distance matrix

			No. of distance computations (approx. in millions)		Time (approx. in min)		
Dataset	Threshold	No. of clusters	Distance matrix	Accelerated SL	Distance matrix	Accelerated SL	RI
Pen-digits Testing	10	3426	6.1	1.09	10.8	6.3	1.0
	20	2269		1.10		10.34	1.0
	30	1106		0.61		2.8	1.0
Pendigits Training	10	7357	28	4.39	52.2	25.87	1.0
	20	4813		4.49		39.48	1.0
	30	2295		2.05		9.07	1.0
Letter	3	5183	199.99	16.98	372.4	70.23	1.0
	3.5	3481		12.03		44.84	1.0
	4	2199		8.21		28.05	1.0
	4.5	1505		9.15		30.04	1.0
	5	969		8.83		29.94	1.0
Shuttle	3.5	9149	1681	16.75	3172.56	642.15	1.0
	4	7199		13.16		70.90	1.0
	4.5	5840		11.00		54.92	1.0
	5	4745		9.21		42.65	1.0
	5.5	3890		8.09		34.29	1.0

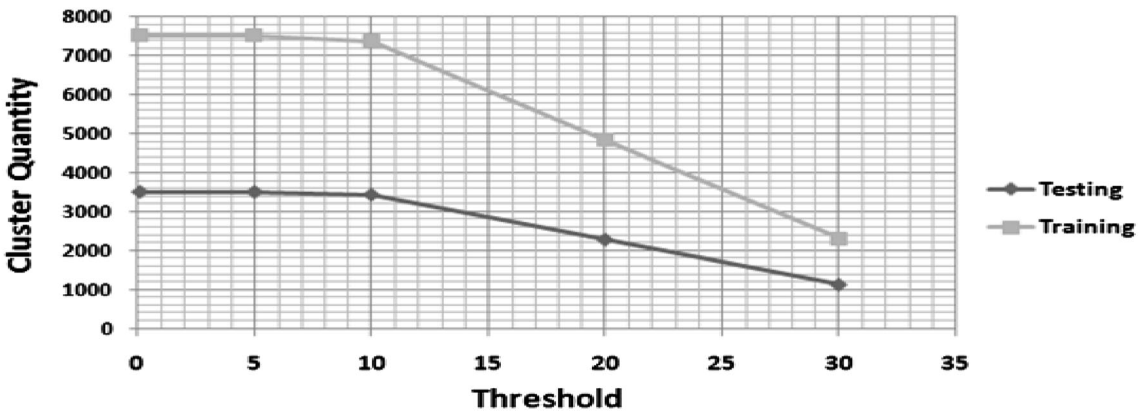

Figure 1. Variation of cluster number with the threshold at the pre-clustering output for Pendigits dataset.

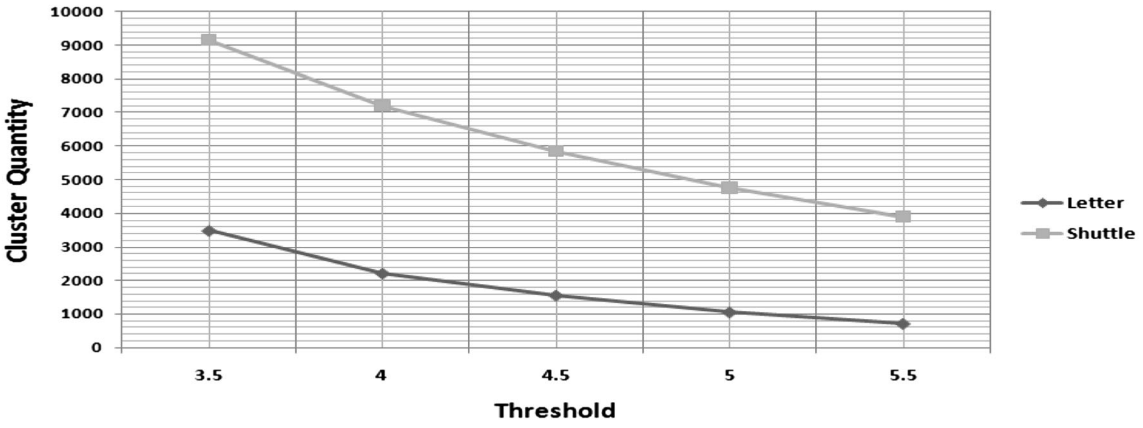

Figure 2. Variation of cluster number with the threshold at the pre-clustering output for Letter and Shuttle datasets.

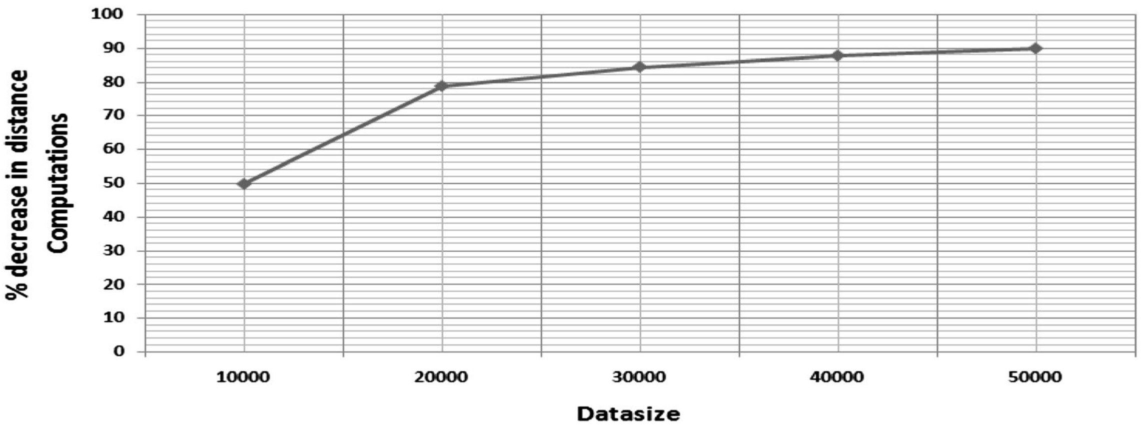

Figure 3. Percentage decrease in distance computations vs. data size for Shuttle dataset (threshold = 3).

7. Conclusion and future scope

The paper presents a two-level, partially incremental SL algorithm that does not need the entire database in advance. The first stage of the algorithm creates some partially formed clusters using an accelerated version of Leader algorithm and the second stage creates the final Dendrogram that does not have individual data points at its base but contains these partially formed clusters. The first stage employs triangle inequality to avoid the redundant distance computations and the second stage, using the property of the SL algorithm, itself merges clusters without directly measuring the distance between them. The proposed algorithm thus saves a lot of unnecessary distance computations, which in turn decreases the convergence time of the algorithm to a large extent, as compared with its well-known variants. The proposed method, although having an incremental stage, does not get affected by the ordering of the data points as the final stage compensates the effect of the ordering of the data. The major advantage of this algorithm is that it does not affect the cluster quality in the process of speeding up the method. A threshold is an important criterion of SL algorithm and an increase in threshold value decreases both the quantity and quality (compactness) of clusters by absorbing distant points to the already available clusters. Hence, it is the sole responsibility of the user to choose the correct threshold to get good clusters. Future work includes modification of the algorithm to incorporate parallelism and hybridization techniques to reduce its convergence time, applying this algorithm to any real-time application and to study its performance. We also aim to apply the triangular inequality to various other clustering algorithms to make them faster than the already proposed methods.

Appendix

Nomenclature used in the paper:

Symbol	Description
$\cup$	Union
α_i, α_j, β, γ	Agglomeration criteria in Lance–William Formula
n	The total number of input data
D	Input dataset
L	Leader set
l_i	Used to denote a Leader
T	User-defined threshold
m	Total no of pre-clusters
C_j	Cluster corresponding to a column in the neighbourhood matrix
C_i	Cluster corresponding to a row in the neighbourhood matrix
C_m	Cluster newly merged to C_j
P	The average number of members in each pre-cluster
f	Number of neighbours obtained in a single scan by a cluster with low flag
η	The maximum number of times a column can be scanned by any cluster with low flag

References

[1] Madhulatha T S 2012 An overview on clustering methods. *IOSR J. Eng.* 2: 719–725

[2] Dunham M H 2003 Clustering. In: *Data mining: introductory and advanced topics*. New Delhi, India: Prentice-Hall/ Pearson Education, pp. 125–128

[3] Hartigan J A 1975 Introduction. In: *Clustering algorithms*. New York, USA: John Wiley & Sons, Inc., pp. 1–7

[4] Jain A K, Murty M N and Flynn P J 1999 Data clustering: a review. *ACM Comput. Surv.* 31: 264–323

[5] Kaur S, Sohal H S and Cheema R S 2013 Implementing UPGMA and NJ method for phylogenetic tree construction using hierarchical clustering. *Int. J. Comput. Sci. Technol.* 4: 303–310

[6] Alexander N, Woetzel N and Meiler J 2011 BCL::Cluster: a method for clustering biological molecules coupled with visualization in the Pymol Molecular Graphics System. In: *Proceedings of the 1st IEEE International Conference on Computational Advances in Bio- and Medical Sciences*, pp. 13–18

[7] Eisen M B, Spellman P T, Brown P O and Botstein D 1998 Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. USA* 95: 14,863–14,868

[8] Elkan C 2003 Using the triangle inequality to accelerate K-Means. In: *Proceedings of the 20th International Conference on Machine Learning (ICML)*, pp. 147–153

[9] Kryszkiewicz M and Lasek P 2010 TI-DBSCAN: clustering with DBSCAN by means of the triangle inequality. In: *Proceedings of the 7th International Conference on Rough Sets and Current Trends in Computing*, pp. 60–69

[10] Patra B K, Hubballi N, Biswas S and Nandi S 2010 Distance based fast hierarchical clustering method for large datasets. In: *Proceedings of the 7th International Conference on Rough Sets and Current Trends in Computing*, pp. 50–59

[11] Ghuman S S 2016 Clustering techniques—a review. *Int. J. Comput. Sci. Mob. Comput.* 5: 524–530

[12] Rafsanjani M K, Varzaneh Z A and Chukanlo N E 2012 A survey of hierarchical clustering algorithms. *J. Math. Comput. Sci.* 5: 229–240

[13] Murtagh F and Contreras P 2012 Algorithms for hierarchical clustering: an overview. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* 2: 86–97

[14] Rohlf F J 1982 Single-link clustering algorithms. In: Krishnaiah P R and Kanal L N (Eds.) *Handbook of Statistics*. Amsterdam: North-Holland, vol. 2, pp. 267–284

[15] El-Hamdouchi A and Willett P 1989 Comparison of hierarchic agglomerative clustering methods for document retrieval. *Comput. J.* 32: 220–227

[16] Jin C, Patwary M M A, Agrawal A, Hendrix W, Liao W K and Choudhary A 2013 DiSC: a distributed single-linkage hierarchical clustering algorithm using MapReduce. In: *Proceedings of the 4th International SC Workshop on Data Intensive Computing in the Clouds*

[17] Patra B K, Nandi S and Viswanath P 2011 A distance based clustering method for arbitrary shaped clusters in large datasets. *Pattern Recognit.* 44: 2862–2870

[18] Firdaus S and Uddin M A 2015 A survey on clustering algorithms and complexity analysis. *Int. J. Comput. Sci. Issues* 12: 62–85

[19] Sibson R 1973 SLINK: an optimally efficient algorithm for the single-link cluster method. *Comput. J.* 16: 30–34

[20] Gower J C and Ross G J S 1969 Minimum spanning trees and single-linkage cluster analysis. *Appl. Stat.* 18: 54–64

[21] Patra B K and Nandi S 2009 Fast single-link clustering method based on tolerance rough set model. In: *Proceedings of RSFDGrC 2009*, pp. 414–422

[22] Blake C L and Merz C J 1998 *UCI repository of machine learning databases*, University of California. Available from: http://www.ics.uci.edu/~mlearn/MLRepository.html

[23] Rand W M 1971 Objective criteria for the evaluation of clustering methods. *J. Am. Stat. Assoc.* 66: 846–850