Fast Hierarchical Clustering and Its Validation

Manoranjan Dash[a,*], Huan Liu[b], Peter Scheuermann[a], Kian Lee Tan[c]
[a]Department of Electrical & Computer Engineering, Northwestern University, Evanston, IL 60208
[b]Department of Computer Science & Engineering, Arizona State University, Tempe, AZ 85287
[c]School of Computing, National University of Singapore, Singapore 117543

Abstract

Clustering is the task of grouping similar objects into clusters. A prominent and useful class of algorithm is hierarchical agglomerative clustering (HAC) which iteratively agglomerates the closest pair until all data points belong to one cluster. It outputs a dendrogram showing all N levels of agglomerations where N is the number of objects in the dataset. However, HAC methods have several drawbacks: (1) high time and memory complexities for clustering, and (2) inefficient and inaccurate cluster validation. In this paper we show that these drawbacks can be alleviated by closely studying the dendrogram. Empirical study shows that most HAC algorithms follow a trend where, except for a number of top levels of the dendrogram, all lower levels agglomerate clusters which are very small in size and close in proximity to other clusters. Methods are proposed that exploit this characteristic to reduce the time and memory complexities significantly and to make validation very efficient and accurate. Analysis and experiments show the effectiveness of the proposed method.
Keywords: Clustering, validation, large and high-dimensional datasets, Voronoi Diagram

1 Introduction

Clustering is the process of grouping similar objects into clusters or classes. It is an important data exploration task used in diversified areas such as market segmentation in business, gene categorization in biology, spatial discovery, and document classification on the WWW. A popular method of clustering is hierarchical agglomerative clustering (HAC). It starts with each point in separate clusters and iteratively agglomerates the closest pair of clusters in each iteration until all points belong to a single cluster. The final hierarchical cluster structure is called a *dendrogram* (see Figure 1), a tree like structure that shows which clusters are agglomerated at each level. Each level of a dendrogram can be evaluated by a cluster validation method and the best level and its corresponding clusters are returned.

HAC algorithms are non-parametric – assume little about data, natural and simple in grouping objects, and capable of finding clusters of different shapes by using different similarity measures. See [16, 19] for surveys on HAC. However, they have several drawbacks. First, HAC algorithms have high time and memory complexities. For example, an efficient algorithm for centroid method (that represents each cluster by its centroid) – priority queue algorithm – has time complexity $O(N^2 \log N)$. Although, to apply HAC to large data, some techniques are proposed in BIRCH [32] and CURE [15],

*Corresponding author. E-mail: manoranj@ece.northwestern.edu

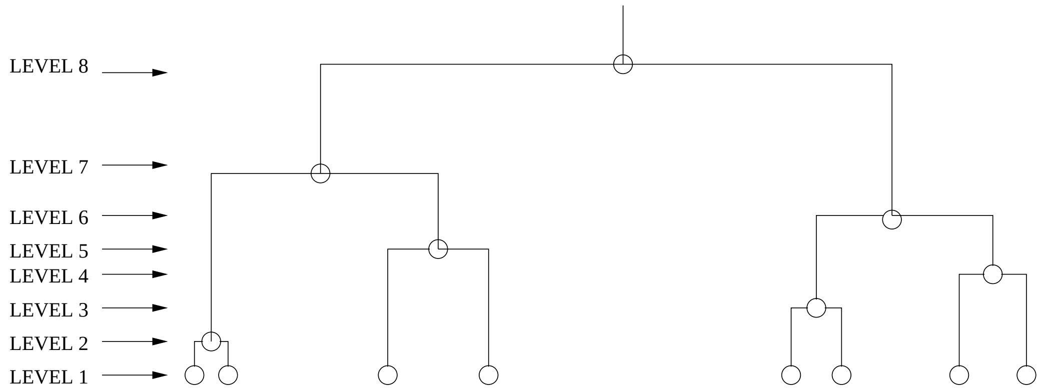


Figure 1: Dendrogram: Output of Hierarchical Clustering

they do not make the traditional HAC algorithms faster. Instead, they use approximations (e.g., summarized (BIRCH) or sampled (CURE) points) so as to reduce the computational cost without losing too much of accuracy. Note that the accuracy here is with respect to the dendrogram produced by the traditional HAC algorithms. In addition, the summaries or samples still need to run the traditional HAC algorithms to generate the dendrogram.

Second, cluster validation using the dendrogram has several limitations. Cluster validation is the task of determining an 'optimal' number of clusters in a dataset. Typically, validation methods evaluate all N levels of the dendrogram to output the best level (see [24] for a review of validation methods). Time complexity for evaluating all N levels is $O(N^2)$ or higher which is prohibitive for large datasets. Furthermore, many validation methods show some *distracting patterns* for lower levels of the dendrogram leading to inaccurate estimation of 'optimal' number of clusters. In Figure 8 we have given an example of distracting pattern.

In this paper we alleviate the above drawbacks by proposing fast yet accurate HAC algorithms, and efficient cluster validation method. Extensive empirical study shows that, except for a number of top levels of the dendrogram, all lower levels agglomerate clusters which are very small in size and close in proximity to other clusters. We name this characteristic as *90-10 rule* to suggest that most levels from the bottom merge pairs of very small clusters separated by very small distance.

Based on this 90-10 rule we propose fast HAC algorithms that significantly reduce the time and memory requirement of the existing HAC methods. A data structure – hyper-plane clustering tree (hpc-tree) – is proposed to cluster high-dimensional data efficiently. Among other benefits, parallel HAC based on the proposed method is significantly better than the existing parallel algorithms, and the proposed method can easily be applied to other clustering algorithms.

In the paper the 90-10 rule is also applied to improve the efficiency and accuracy of existing cluster validation methods significantly. Out of the N levels of the dendrogram, only a small number of top levels are potential candidates for validation because the optimal number of clusters K is typically very small compared to N. The validation process becomes more efficient and accurate if the large number of lower levels, which are mostly useless from validation point of view and consume most of the computational cost, are rejected. The 90-10 rule is used to efficiently reject these unnecessary levels and focus only on the potential levels, thus reducing the computational cost significantly and improving the accuracy by avoiding the distracting patterns.

The rest of the paper is organized as follows. The next section describes some important observations on HAC. Section 3 describes the proposed methods for fast HAC. Section 4 describes the proposed methods for efficient cluster validation. Section 5 concludes the paper with brief discussion on future work.

2 Important Observations

Let us assume that our dataset consists of N data points or objects each with M dimensions or attributes. We shall denote X_{i_j} as the j^{th} dimension value of i^{th} data point where $i = 1...N$ and $j = 1...M$. We assume only real type data. Dissimilarity between two data points containing real data is measured by their Euclidean (L_2) distance. Other distance measures such as *Manhattan* distance can be used (see [8]).

In the following we give an outline of a HAC algorithm.

Algorithm: Hierarchical Agglomerative Clustering
Input: Data (N,M)
Output: Dendrogram
1. for $i = 1$ to $N - 1$
2. find the closest pair of clusters
3. merge the closest pair of clusters
4. return dendrogram

In step 2, depending on the similarity metric used, the closest pair in each iteration can vary. In Section 2.2 we discuss different types of similarity metrics. Different data structures and techniques used for finding the closest pairs are described in Section 3.1.

2.1 The 90-10 Rule

We run centroid type HAC method over a 2-D dataset with 100 clusters and some noise (see Figure 2). In the centroid type, each cluster is represented by its centroid and the cluster pair having closest centroids are merged in each iteration. Based on the HAC results we have plotted closest pair distance, size of the larger cluster, and total time until each iteration in *distance*, *size*, and *time* plots respectively in Figure 2. X-axes of these plots stand for percentage of iterations of HAC.

Notice that, in the distance plot most agglomerations, except for a small portion towards the end, have distance very small compared to the maximum closest pair distance. Similarly, in the size plot, except for the last portion, the size is very small compared to N. The time plot shows that initial agglomerations are more time consuming than later agglomerations because number of clusters remaining to be merged reduces as the algorithm progresses. We experimented with many datasets having varying characteristics. The characteristics chosen are those that affect the clustering of a data. Datasets were selected having noise and consisting of clusters with different shapes (e.g. spherical, arbitrary), with multiple resolutions, and with overlap among the clusters. For varying M, N (typically large – at least a few thousand points), and K, the general trend is as follows: if majority points are inside clusters then the shape of the distance plot (and the size plot) are similar to the one shown in Figure 2. A counter example, to this assumption that majority points are inside clusters, is uniformly distributed high-dimensional data. But typical clustering applications, that assume that *data is not uniformly distributed but consists of some clusters*, follow the trend.

We name these observations *90-10 rule* stated as follows:

in a dendrogram, most levels from the bottom merge pairs of very small clusters separated by a very small portion of the maximum closest pair distance.

Note that the '90' and '10' are only symbolic and do *not* suggest that exactly 90% agglomerations merge clusters separated by distance less than exactly 10% of the maximum closest pair distance. In fact, one can name it 80-20 or 70-30 but the idea need to be conveyed is that until the last portion of agglomerations clusters are very close in proximity and very small in size.

2.2 Adherence to the 90-10 Rule by Other Similarity Metrics

Similarity between two clusters is measured by different metrics which are broadly categorized as *geometric* and *graph*. In this section we examine their adherence to the 90-10 rule. For an HAC algorithm to follow the 90-10 rule they should (1)

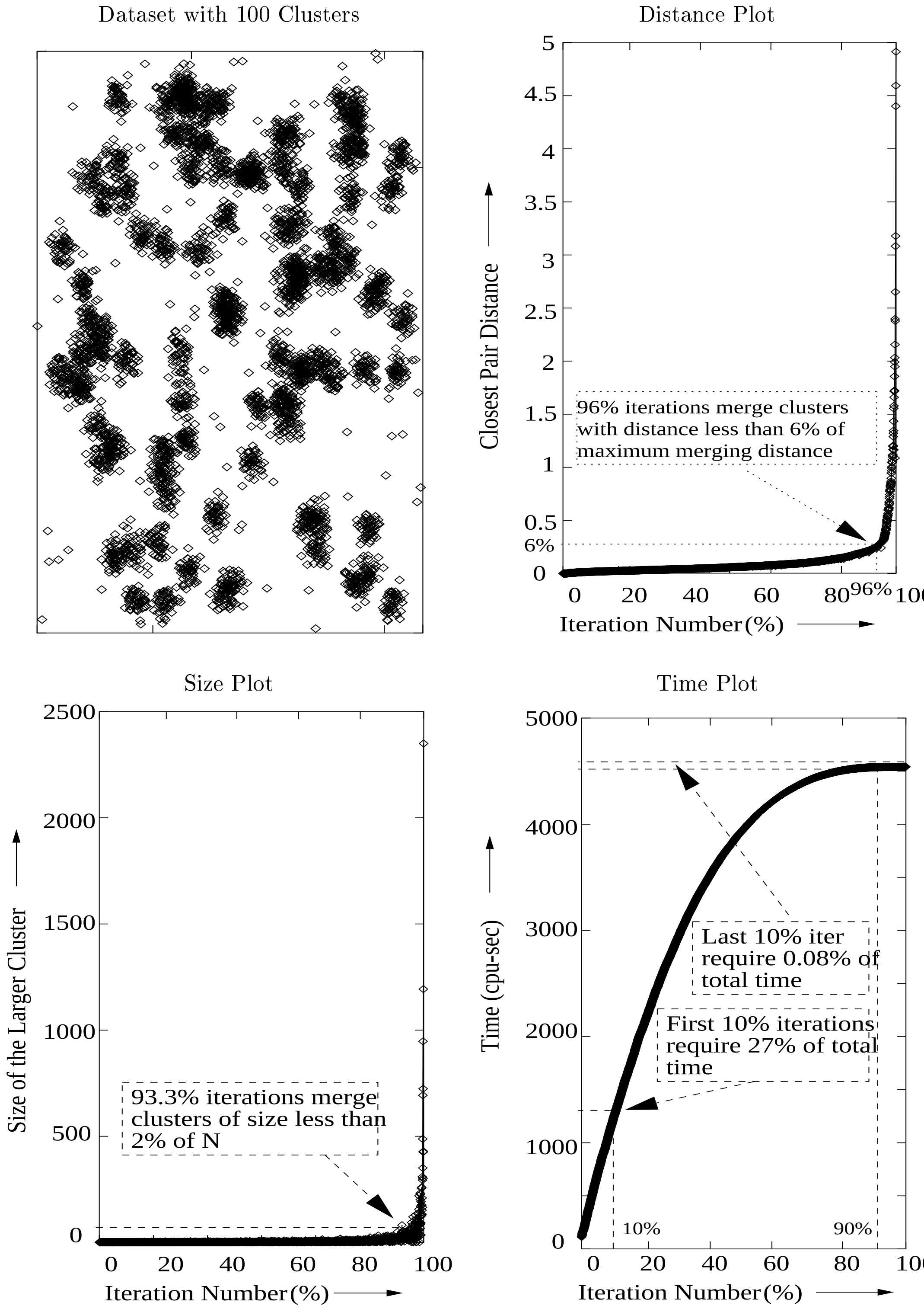


Figure 2: The 90-10 Rule: Important Observations about HAC

merge small clusters in the beginning and the larger clusters towards the end, and (2) merge clusters of close proximity in the beginning and the distant clusters towards the end. In other words, an HAC algorithm adheres to the rule if most agglomerations from the beginning are intra-cluster and only a small number of iterations towards the end are inter-cluster. The following discussion is based on empirical experiments conducted on different types of datasets.

Geometric Metrics These metrics represent each cluster by a single representative point. For example, the centroid method we just discussed belongs to this category where each cluster is represented by its centroid. The other prominent methods in this category are median and minimum variance Ward's method [30]. Although both centroid and median methods represent each cluster by its centroid and measure similarity of two clusters by the distance between their centroids, the difference lies in how they calculate the centroid after merging the closest pair. Centroid type gives equal weight to each point in the two clusters whereas median type gives equal weight to each cluster. In Sneath and Sokal terminology [28] centroid type is UPGMC or unweighted centroid and median type is WPGMC or weighted centroid. Like the centroid and median methods, Ward's minimum variance method represents a cluster by its centroid. But unlike them, in each iteration it merges the pair of clusters with the least increase in the sum of square error. Square-error for a cluster is the sum of squared distances of all points to its centroid.

Experiments and analysis show that, all three geometric metrics follow the 90-10 rule when majority of points in the data belong to some clusters. The reason is as the clusters are represented by single representative points they merge small and closely packed clusters in the beginning which are intra-cluster and the bigger and distant clusters towards the end which are inter-cluster.

Graph Metrics In contrast to geometric type, a graph metric considers each point in a cluster to be its representative. Examples are single link, complete link and average link. In single link, dissimilarity between two clusters is the minimum distance between points in the two clusters. In complete link, dissimilarity between two clusters is the maximum distance between points in the two clusters. Single link method has the tendency to merge large clusters connected by a chain of points. It is known as *chaining effect* [19]. In contrast, complete link method favors compact and small clusters. Average link method measures the dissimilarity between two clusters as the average distance between points in the two clusters. In Sneath and Sokal terminology [28] average link method is grouped into two types: UPGMA (each point has equal weight) and WPGMA (each cluster has equal weight). Experiments show that complete and average link methods follow the 90-10 rule just like the geometric metrics when majority points belong to some clusters. The reason is they prefer close and compact clusters over far and big clusters. Although single link method follows the 90-10 rule for datasets that have well formed clusters, it violates the rule when clusters are connected by a chain of points. For example, if a dataset contains only a chain of equally spaced points then the distance plot for single link method will show only one distance value. Due to the chaining effect, initial agglomerations can be inter-cluster.

In summary, most similarity metrics follow the 90-10 rule in typical clustering applications where data consists of some clusters and majority points belong to these clusters. So, irrespective of any particular application domain, the adherence or lack of it depends on the similarity metric used. For example, if a similarity metric can correctly find clusters in an application, then irrespective of the type of application domain, it can be expected that the 90-10 rule will apply as well. Based on this 90-10 rule, the next section proposes fast HAC algorithms and Section 4 proposes efficient and accurate cluster validation methods.

3 Fast Hierarchical Agglomerative Clustering

In this section we propose fast HAC algorithms based on *p*artially *o*verlapping *p*artitioning (POP). POP exploits the 90-10 rule to significantly reduce the time and memory requirement. Next section describes the complexities of existing HAC algorithms. Section 3.2 presents two algorithms based on POP and a new data structure to cluster high-dimensional data efficiently. Section 3.3 discusses the experimental results.

3.1 Complexities of Existing HAC Algorithms

In this section we discuss time and memory complexities of some prominent HAC algorithms that work for both graph and geometric metrics. We use centroid method (described earlier) for the rest of the paper if not otherwise mentioned.

The first centroid algorithm proposed uses a dissimilarity matrix that stores dissimilarities between clusters. In each iteration, the dissimilarity matrix is searched to find the closest pair of clusters. Let us call this '*step 1*'. For n ($n = N ... 2$) remaining clusters in an iteration, it requires $O(n^2)$ time to search through the dissimilarity matrix, and for $N-1$ iterations this step takes $O(N^3)$ time. After merging the closest pair, dissimilarity matrix is updated by deleting the column entries for the closest pair and by creating a new row for the new cluster by determining distance from other clusters. Let us call this '*step 2*'. For n clusters in an iteration, this step takes $O(n)$ time to update, and so for $N-1$ iterations it takes

$O(N^2)$ time. Hence the overall time complexity of the algorithm is $O(N^3)$. The memory required is $O(N^2)$ because of the dissimilarity matrix. This simple algorithm can be improved by maintaining a nearest neighbor array that stores nearest neighbor for each cluster. This way, *step 1* requires only to find the minimum of the nearest neighbor distances from the nearest neighbor array in $O(n)$ time for n clusters in any iteration. But, *step 2* requires $O(n^2)$ time in each iteration if a naive nearest neighbor algorithm of $O(n)$ time complexity is used. Day and Edelsbrunner [9] suggested two ways to improve the time complexity of this algorithm by improving *step 2*'s complexity. Type (1) – By obtaining an improved bound on the required number of nearest neighbor updates: If the number of clusters to update after an iteration is α then the overall complexity becomes $O(\alpha N^2)$. Using geometric preliminaries they proposed an upper bound for α as $2(3^M - 2)$ where M is the number of dimensions. Type (2) – By obtaining an improved bound on the time required for each update: They suggested using a heap-based priority queue that requires $O(\log n)$ time for each insert and delete operation on a priority queue. This results in an overall time complexity of $O(N^2 \log N)$. [1]

Anderberg [1] classified hierarchical algorithms into 'stored matrix' and 'stored data' types. Stored matrix algorithms maintain a dissimilarity matrix whereas stored data algorithms do not but instead calculate the dissimilarities as required. A major distinction is that stored matrix methods are preferred when memory is sufficient to store the dissimilarity matrix of size $O(N^2)$, otherwise stored data is the way out. Type (2) algorithm of Day and Edelsbrunner that uses priority queues is a stored matrix method while type (1) algorithm is more suitable as stored data if M is not large and if memory is not enough to store dissimilarity matrix. Priority queue algorithm is used in the paper to describe the proposed techniques, but complexity analysis and experiments are done for the above different types of algorithms.

3.2 Proposed Methods

In this section we propose new algorithms based on *p*artially *o*verlapping *p*artitioning (POP).

3.2.1 2-Phase Algorithm

We propose a new 2-phase algorithm for HAC based on POP. In *phase 1* data is partitioned into p number of overlapping cells. The region of overlapping is called δ-region where δ is the separating distance. Figure 3 shows an axis-parallel POP that divides the data-space uniformly into p cells. For centroid metric (and other geometric metrics) each cluster is

[1] For metrics that satisfy *reducibility property* [25], one can perform clustering in $O(N^2)$ time. But the geometric metrics, such as centroid and median metrics, do not satisfy this property, and hence only approximate algorithms are proposed in [25] for these cases.

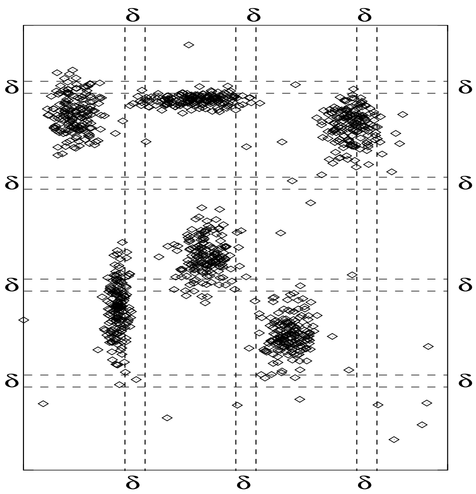


Figure 3: Axis-Parallel Partially Overlapping Partitioning (POP)

represented by a single representative point. If the representative point of a cluster falls in a δ-region then each affected cell holds it, otherwise only one cell holds it. In [8] we give a procedure about how to handle graph metrics where each point in a cluster is a representative of the cluster. The basic idea of POP is, in each iteration, the closest pair is found for each cell, and from those the overall closest pair is found. If the overall closest pair distance is less than δ then the pair is merged and the priority queues (or the nearest neighbor array depending on the algorithm employed) of only the container cell are updated. If the closest pair or the merged cluster is in a δ-region then the priority queues of the affected cells are also updated. *Phase 1* terminates when the closest pair distance exceeds δ. *Phase 2* merges the remaining clusters of *phase 1* using the traditional algorithm, thus completing the dendrogram. The complete algorithm is given in Table 1. By setting δ to the closest pair distance at the turning point of distance plot (see Figure 2), a large number of small clusters are merged in *phase 1* that uses POP while only a small number of larger clusters are merged in *phase 2* that uses the traditional algorithm. This way the overall computational time is reduced significantly.

Accuracy As the POP in *phase 1* ensures that *any pair with distance less than δ must reside together in at least one cell*, and as *phase 2* is the traditional method itself, 2-phase algorithm guarantees correct dendrogram. The correctness of the dendrogram is with respect to the traditional HAC algorithm. Note that ties between the closest pair distances can produce different dendrograms because there is no unique way of resolving them.

Algorithm	Time Complexity	Memory Complexity
Input: Data (N,M), p, δ		
Output: Dendrogram		
/* Phase 1 */		
1. divide the data into p number of overlapping cells	$O(N)$	$O(N + p * \lvert\delta\rvert)$
2. Create priority queues P for each cell	$O(p * (\frac{N}{p} + \lvert\delta\rvert)^2)$	$O(p * (\frac{N}{p} + \lvert\delta\rvert)^2)$
3. repeat		
4. for each cell get its closest pair	$O(p * (\frac{n}{p} + \lvert\delta\rvert))$	
5. determine overall closest pair (C_1, C_2)	$O(p)$	
6. if $dist(C_1, C_2) > \delta$ go to 10		
7. merge C_1 and C_2, update corresponding P	$O((\frac{n}{p} + \lvert\delta\rvert) \log(\frac{n}{p} + \lvert\delta\rvert))$	
8. update P of the affected cells, if any	$O(\beta * (\frac{n}{p} + \lvert\delta\rvert) \log(\frac{n}{p} + \lvert\delta\rvert))$	
9. until $dist(C_1, C_2) > \delta$		
/* Phase 2 */		
10. merge k' clusters by traditional method	$O({k'}^2 \log k')$	$O({k'}^2)$
11. return dendrogram		
Over-All Complexity	$O((N - k') * (\beta + 1)$ $*(\frac{N}{p} + \lvert\delta\rvert) \log(\frac{N}{p} + \lvert\delta\rvert))$ $+O({k'}^2 \log k')$	$O(p * (\frac{N}{p} + \lvert\delta\rvert)^2)$ or $O({k'}^2)$

Table 1: 2-phase Algorithm Using Priority Queues

Complexity Analysis To simplify the analysis, we assume equal cell size and equal δ-region size ($|\delta|$) for each cell. $|\delta|$ denotes the number of clusters in the δ-region of any particular cell. The original priority queue algorithm proposed by Day and Edelsbrunner [9] had an overall time complexity $O(N^2 \log N)$ and memory complexity $O(N^2)$. On the other hand, 2-phase algorithm, as shown in Table 1, has an overall time complexity $O((N-k')*(\beta+1)*(\frac{N}{p}+|\delta|)\log(\frac{N}{p}+|\delta|))+O({k'}^2 \log k')$ assuming $\log(\frac{N}{p} + |\delta|)$ to be greater than p where k' is the number of clusters remaining after *phase 1* and β is the average number of cells affected (other than the container cell) in each iteration. The memory complexity is $O(p * (\frac{N}{p} + |\delta|)^2)$ or $O(k'^2)$ whichever is larger. If δ is set to the closest pair distance at the turning point of the distance plot, then both $|\delta|$ and k' are very small. So, if we do not consider $|\delta|$ and k', then the time complexity becomes $O(N * (\beta + 1) * \frac{N}{p} \log \frac{N}{p})$, i.e. $O((\beta + 1) * \frac{N^2}{p} \log \frac{N}{p})$ (gain factor $= \log_{\frac{N}{p}} N \times \frac{p}{\beta+1}$), and the memory complexity is $O(p * \frac{N^2}{p^2})$, i.e. $O(\frac{N^2}{p})$ (gain factor =

Type of Algorithm	Traditional Algorithm	2-Phase Algorithm After Simplification	2-Phase Algorithm Gain Factor $= \frac{Traditional}{Proposed}$
Time Complexity			
Stored Matrix			
– Similarity Matrix	$O(N^3)$	$O(\frac{N^3}{p})$	p
– Priority Queues	$O(N^2 \log N)$	$O((\beta+1) * \frac{N^2}{p} \log \frac{N}{p})$	$\log_{\frac{N}{p}} N \times \frac{p}{\beta+1}$
Stored Data	$O(\alpha * N^2)$	$O((\beta+1) * \alpha * \frac{N^2}{p})$	$\frac{p}{\beta+1}$
Memory Complexity			
Stored Matrix			
– Similarity Matrix	$O(N^2)$	$O(\frac{N^2}{p})$	p
– Priority Queues	$O(N^2)$	$O(\frac{N^2}{p})$	p
Stored Data	$O(N)$	$O(N)$	1

Table 2: Comparison between 2-phase and Traditional Algorithms

p). The average number of affected cells including the container cell is $\beta + 1$. Its value depends on the data distribution: in the worst case, it is 2^M when each agglomeration affects maximum possible number of cells for M dimensions while in the best case it is just 1 when each agglomeration affects only its container cell. Experiments show $\beta + 1$ is small and, for all test datasets in this paper, it is less than 2.

The original centroid algorithm uses a dissimilarity matrix instead of priority queues and does not maintain nearest neighbors. Its time complexity is $O(N^3)$ and memory complexity is $O(N^2)$. If one uses the 2-phase algorithm then the overall time complexity becomes $O((N - k') * p * (\frac{N}{p} + |\delta|)^2) + O(k'^3)$ and memory complexity is $O(p * (\frac{N}{p} + |\delta|)^2)$ or $O(k'^2)$ whichever is larger. Note that average number of affected cells is not a factor in this case as the overall complexity is due to the quadratic time required for searching through the dissimilarity matrices. After simplification (i.e. not considering $|\delta|$ and k') the time complexity becomes $O(N * p * \frac{N^2}{p^2})$, i.e. $O(\frac{N^3}{p})$ (gain factor $= p$) and the memory complexity is $O(\frac{N^2}{p})$. (gain factor $= p$)

The above two types of algorithm are 'stored matrix' type according to Anderberg's classification [1]. In the traditional

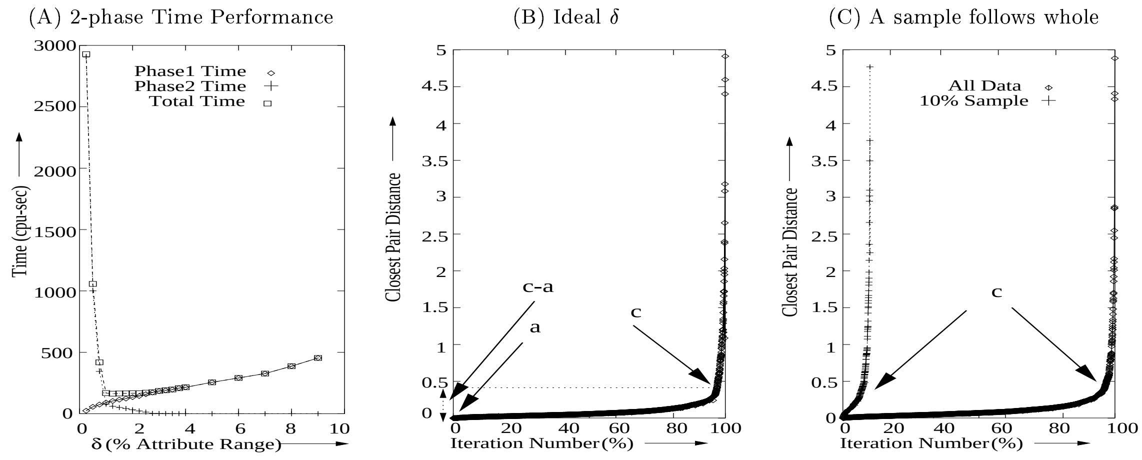


Figure 4: Analysis of 2-Phase Algorithm

'stored data' type a nearest neighbor array is maintained giving a time complexity $O(\alpha * N^2)$ and memory complexity $O(N)$ where α is the number of updates per iteration. On the other hand, time complexity of 2-phase stored data algorithm is $O((\beta+1) * (N-k') * \alpha * (\frac{N}{p} + |\delta|)) + O(\alpha * k'^2)$. After simplification, it becomes $O((\beta+1) * \alpha * \frac{N^2}{p})$ (gain factor $= \frac{p}{\beta+1}$). The memory complexity is $O(p * \frac{N}{p} + |\delta|)$ or $O(k')$ whichever is higher. After simplification it becomes $O(N)$ (gain factor = 1).

To summarize, the 2-phase algorithm is more efficient than traditional algorithms in both time and memory complexities making it attractive to be deployed.

3.2.2 Nested Algorithm

The performance of the 2-phase algorithm depends on the number of partitions p and the overlapping width δ. As shown in Figure 4, there exists an ideal δ $(= c)$ at the turning point of the distance plot at which the total time taken by the 2-phase algorithm is minimum. Figure 4 (A), that plots the time taken by *phase 1* and *phase 2* for the dataset shown in Figure 2, shows that if δ is smaller than c then a large k' number of clusters remain for the traditional algorithm of *phase 2* whereas if δ is larger than c then time is spent unnecessarily in partitioning in *phase 1* even when the number of remaining clusters is very small for *phase 2*. Figure 4 (B) shows that nearest neighbor search will gain maximum for $\delta = a$ because any pair of points with distance less than a must be found together in at least one of the cells [27] whereas clustering gains maximum for $\delta = c$. Hence, we see successful application of POP in clustering is different from that in nearest neighbor search. The ideal δ or the turning point of the distance plot can be estimated in several ways. Sampling is useful in the following way: take a random sample and obtain its distance plot. Figure 4 (C) shows how the distance plot for a 10% sample follows that for the whole data. The turning points of the distance plot for the sample and the whole data almost match because although a sample typically increases the small intra-cluster distances but it retains the large inter-cluster distances. This is also the reason why, in general, the sampling approach has been effective for clustering. In Section 4 we discuss methods to estimate the turning point of the distance plot. However, estimating the turning point has the disadvantage of running HAC algorithm once more to create the distance plot. We can avoid this by dynamically setting the δ and p values. It starts with a very small δ and gradually increases it until there remains very few or just one cluster. As δ increases, p which is set to a high value in the beginning is gradually reduced. We call this method *nested*.

Complexity Analysis: Let us assume that the nested sequence of δ and p is $<p_j,\delta_j>$ for j=1...s where p_j and δ_j are p and δ values in the j^{th} nested iteration respectively, and s is the number of iterations in the nested sequence. Let n_j be the number of clusters remaining after nested iteration $j-1$; initially n_1 is N and finally (after s nested iterations) n_{s+1} is k'. According to the complexity analysis of Table 1, time complexity of nested algorithm is $\sum_{j=1}^{s}((n_j - n_{j+1})O((\beta+1) * (\frac{n_j}{p_j} + |\delta_j|)\log(\frac{n_j}{p_j} + |\delta_j|$ $O(k'^2 \log k')$. By specifying enough nested iterations s, k' is made negligible, and so $O(k'^2 \log k')$ is discarded. If $|\delta_j|$s are not considered then the time complexity becomes $\sum_{j=1}^{s}((n_j - n_{j+1})O((\beta+1) * \frac{n_j}{p_j}\log\frac{n_j}{p_j}))$. Compare this with the time complexity $O((\beta+1) * \frac{N^2}{p}\log\frac{N}{p})$ of 2-phase algorithm. The gain of nested partitioning over 2-phase will depend largely on the sequence of $(n_j - n_{j+1})$s. The memory complexity of nested algorithm is $\max_{j=1\ldots s}[p_j * O((\frac{n_j}{p_j} + |\delta_j|)^2)]$, and after simplification it becomes $\max_{j=1\ldots s}[O(\frac{{n_j}^2}{p_j})]$. In Table 3 time and memory complexities of different HAC algorithms for nested partitioning is given.

Experiments show that the nested algorithm is more efficient than the 2-phase algorithm even when δ is set ideally for the 2-phase algorithm. For example, for the dataset shown in Figure 2, the minimum time for 2-phase algorithm is 125.4 cpu sec while nested algorithm takes only 57.8 cpu sec. The reason is nested method improves the POP efficiency further by dividing the difference (c-a) (see Figure 4 (B)). Hence we use the nested algorithm in the rest of the paper.

3.2.3 High-Dimensional Data and hpc-tree

High-dimensional data poses problems for the simple axis-parallel partitioning technique discussed before. If a single dimension is used to partition the data, then the δ distance in single dimension will be too small in high-dimensional

Type of Algorithm	Nested Algorithm
Time Complexity	
Stored Matrix	
– Similarity Matrix	$\sum_{j=1}^{s}((n_j - n_{j+1})O(\frac{{n_j}^2}{p_j}))$
– Priority Queues	$\sum_{j=1}^{s}((n_j - n_{j+1})O((\beta+1) * \frac{n_j}{p_j}\log\frac{n_j}{p_j}))$
Stored Data	$\sum_{j=1}^{s}((n_j - n_{j+1})O((\beta+1) * \alpha * \frac{n_j}{p_j}))$
Memory Complexity	
Stored Matrix	
– Similarity Matrix	$O(\frac{N^2}{p})$
– Priority Queues	$O(\frac{N^2}{p})$
Stored Data	$O(N)$

Table 3: Complexities of Nested Algorithm after Simplification

space which reduces the efficiency drastically. If all (or many) dimensions are used to partition the space and if each is divided into at least two, then for M dimensions there will be 2^M partitions; e.g. for $M = 20$ there will be $2^{20} \approx 10^6$, for $M = 100$ there will be $2^{100} \approx 10^{30}$ (!) partitions. Most of the partitions will be empty, even when number of data points N is large, leading to extreme waste of memory and cpu time.

Before we discuss how to efficiently partition the high-dimensional data we discuss two important arguments both for and against any such partitioning. These two studies are on nearest neighbor search which is closely linked to the problem of finding clusters in the data. In [31], it is shown that almost all partitioning schemes for efficient nearest neighbor search in high-dimensional data fail if the data is distributed uniformly. A clustering application implicitly assumes that data is not uniformly distributed but has clusters in it, and so the assumption of uniform distribution will be irrelevant for this paper. The other study in [3] argues that some partitioning techniques can still be efficient if there are clusters in high-dimensional data. The authors argue that as dimensionality increases the points within a cluster must be increasingly closer than points in different clusters. This can be justified as follows. The authors show in the paper that distance between any pair of points approach the maximum distance if data points are uniformly distributed. So, the gap between intra-cluster and inter-cluster distances increase relatively with increasing dimensionality. With respect to the topic of this paper it can be safely concluded that efficient partitioning techniques are possible for high-dimensional clustering.

Our goal is to construct a partitioning scheme that handles high-dimensionality well. Some additional desirable properties of a partitioning scheme that incorporates POP are: (1) $|\delta|$ should be small, (2) time spent in distributing points and determining affected partitions should be small, and (3) partitions should not hold too many or too few points. In OptiGrid [17] an optimal hyper-plane partitioning scheme is defined for implementing grid-based clustering. This is not applicable for POP as a main goal of OptiGrid is not to break any cluster. This objective can distribute a large number of points in a single cell violating the criterion 3 and resulting in poorer POP performance. An efficient way to satisfy criterion 2 is a tree structure that needs $O(\log n')$ number of comparisons on average to distribute a point and to determine the affected partitions where n' is the number of nodes in the tree. One way to satisfy criterion 1 is to use Voronoi diagram which partitions the space into cells, directed by a set of *split* points $Q = q_1, q_2, ...$ such that for each cell corresponding to point q_i, the points x in that cell are nearer to q_i than to any other split point in Q. (See [26] for details on Voronoi diagram.) Hence, by constructing a hierarchy of Voronoi diagrams, all the above criteria can be satisfied. We call such a structure as hpc – *h*yper *p*lane *c*lustering – tree where the top or root node gives a brief summary of the whole data and is split to many Voronoi cells which are split as well and so on. A challenging aspect of hpc-tree is the selection of split points. In [6] a hierarchical structure of Voronoi diagram called GNAT is proposed for efficient nearest neighbor search in high-dimensional data. The authors have argued that for efficient neighborhood search in high-dimensional data, a hierarchical geometrical structure of data is more suitable than structures that do not consider the intrinsic geometry of the data. In GNAT, split points are chosen randomly and user is required to specify the number of split points in each level of the hierarchy. Wrong selection of split points results in unbalanced tree resulting in disproportionate distribution among the cells and poor POP performance. An ideal split point should be a summary of the points that belong to the corresponding Voronoi cell. In the literature, there are different methods of creating a summary of data (e.g. CF-tree (BIRCH [32]), and Bradley et al's method [5]). CF-tree is more suitable for hpc-tree because of its hierarchical structure. An artifact of CF-tree is that it does not guarantee that a parent node is the closest parent of all of its children which is necessary for hpc-tree. See Appendix A for a brief description of CF-tree construction process.

POP and hpc-tree The overall construction process of hpc-tree is as follows:

1: Create CF-tree and get the split points (i.e., summary points) of different levels;

2: In each level remove split points with distance less than δ from some other split point;

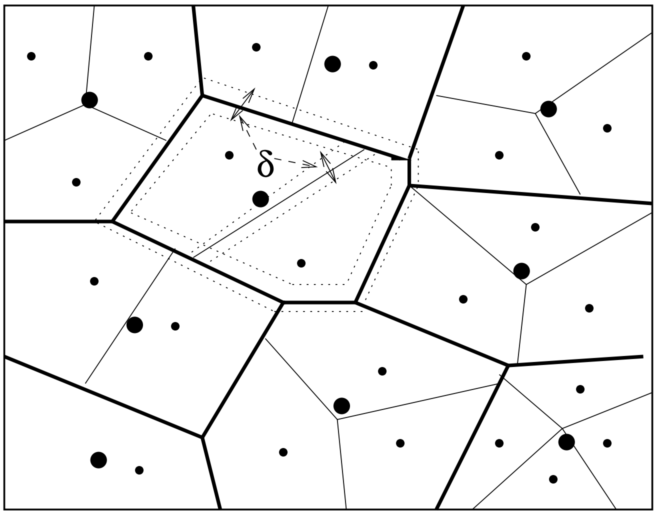


Figure 5: A Simple hpc-tree

3: Starting with the root assign each split point to its closest parent.

The leaf nodes (a node consists of a split point among other information) of hpc-tree are the cells of POP. Starting with the root, for each data point, the path that contains the nearest child is descended until the leaf level is reached. If a point does not belong to δ-boundary of any node of the path, then it is assigned to only one leaf node as a non-boundary point. Otherwise, each affected node is descended and the point is assigned to all the affected leaf nodes as a boundary point, and so on. A simple example of hpc-tree is illustrated in Figure 5. In the 1^{st} level only the root exists which is not shown in the figure. In level 2, there are 9 nodes with heavier split points and thicker boundary lines. The 3^{rd} level has finer split points and thinner boundary lines. The dashed lines show the boundary of width δ for one branch of the hpc-tree. Unlike the axis-parallel partitioning where it is easy to determine the neighboring cells of a data point, in hpc-tree each boundary point stores pointers to the affected partitions. Nested partitioning can use the same hpc-tree for different δ if the minimum separation between the split points is not less than the new δ; otherwise, the tree is reconstructed with the new δ.

The algorithm for nested POP with hpc-tree is similar to the nested algorithm discussed before. The algorithm requires an additional $ThresPts$ parameter to specify a threshold for average number of points in each cell. If after any nested iteration the average number of points becomes less than $ThresPts$ then the tree height is reduced by 1. This will reduce p, the number of leaf nodes in the tree, and hence will increase the average. The default value for $ThresPts$ is two.

Complexity Analysis We first analyze the complexity of building an hpc-tree followed by complexity analysis of nested POP with hpc-tree.

Building hpc-tree: The complexity of building an hpc-tree has two parts: building CF-tree and then building hpc-tree from it. Some notations and assumptions used in the analysis are: size of CF-tree and hpc-tree are assumed to be equal – and it is n' – although hpc-tree is typically smaller than CF-tree, the number of partitions p is equal to the number of leaf nodes and $p \approx \frac{B-1}{B}n'$ where B is the branching factor of CF-tree. In [32] the complexity of CF-tree building is described which we do not repeat here. It is $O(N * B * (1 + \log_B n') + \log_2 \frac{N}{N_0} * (\frac{M}{ES}) * B * (1 + log_B n'))$ where M is the memory allocated for CF-tree, ES is the CF entry size, N_0 is the number of data points loaded into memory initially. Building hpc-tree requires to (a) remove very close split points, (b) recursively assign each child to its closest parent, and (c) distribute all data points among leaf nodes. Removing very close split points in each level is of complexity $O((\log_B n')^2)$. Recursively assigning each child to its parent is of complexity $O((\log_B n')^2)$. Distribution of points among leaf nodes or cells has worst-case complexity $O(N * n')$ when each point is in the boundary of all leaf nodes; it has the best-case complexity of $O(N * B * (1 + log_B n'))$ when no point is in boundary. So, the total cpu cost of building hpc-tree is $O(N * B * (1 + \log_B n') + \log_2 \frac{N}{N_0} * (\frac{M}{ES}) * B * (1 + log_B n') + (\log_B n')^2 + N * n')$.

As for I/O, BIRCH requires a little more than one scan to build the CF-tree. Building hpc-tree requires one scan to distribute the points among the cells.

Nested POP with hpc-tree: Complexity analysis of axis-parallel partitioning discussed in Section 3.2.1 is also applicable here.

Accuracy Arguments of accuracy discussed in Section 3.2.1 also apply here. So, nested POP with hpc-tree outputs a correct dendrogram.

3.3 Experimental Study

We conducted an extensive performance study to determine the effectiveness of the proposed algorithm over the traditional algorithm. Both stored matrix and stored data types are implemented. Priority queue algorithm is used for stored matrix type while nearest-neighbor array is used for stored data type. The proposed method uses nested POP with hpc-tree. Performance is measured by the time taken (total time including the time required to create hpc-tree) and, wherever relevant, by the memory occupied.

3.3.1 Parameter Setting

Performance of the proposed method is directly influenced by p and δ.

Setting p Number of partitions p is directly determined by the size of hpc-tree which is influenced by the size of the CF-tree. According to the authors of BIRCH, size of the CF-tree is affected by memory and page sizes: smaller (larger) page size tends to decrease (increase) the running time, and produces fewer (more) leaf nodes. Similarly, as memory size increases the tree size increases together with the running time. Empirical experiments show that cpu time does not vary more than $\pm 20\%$ for a wide range of values. Poor performance is avoided by not setting too low or too high page and memory sizes. Some insight on what is too low and too high are: do not set these two so low that the average number of points per cell is more than 20 or so high that it is less than 5; a proper page size for 2-D data is 256 and it should be increased with the number of dimensions; a proper memory size for a 2-D data with $N = 6k$ is 80Kbytes and it should be increased with data size.

Setting δ Initially δ is set to a very small value; if the user finds it difficult to set then a possible initial value is 0. For the next nested iteration, δ is set to $(1+\delta_{incr})$ times the last closest pair distance which exceeded the previous δ. So δ_{incr} can be varied for performance tuning. In the experiments, δ_{incr} is set to 0.1, i.e., a 10% increase over the last closest pair distance.

3.3.2 Scalability

Scalability tests are done by varying N and M. Synthetic and benchmark datasets are used for the tests. Synthetic datasets are DS2 (taken from BIRCH [32]), t4.8k, t5.8k, t8.8k, t7.10k (taken from CHAMELEON [20] and CURE [15]). These datasets are shown in Appendix B. Benchmark dataset is SEQUOIA [29] used in DBSCAN [12]. These datasets have different shapes of clusters (e.g. arbitrary and Gaussian) and different noise levels (SEQUOIA has very high noise level). In plots of Figures 6,7 X-axis stands for N (number of points) or M (number of attributes), and Y-axis stands for *time* or *memory* gain factor which is computed as $\frac{traditional}{proposed}$.

<u>**Stored Matrix**</u>

Varying N: N is varied from 1k to 6k (for datasets of size more than 6k traditional algorithm required more memory than that is available in our server) while M is 2. Figure 6 (A) shows the *time* and *memory* gain factors. We observe

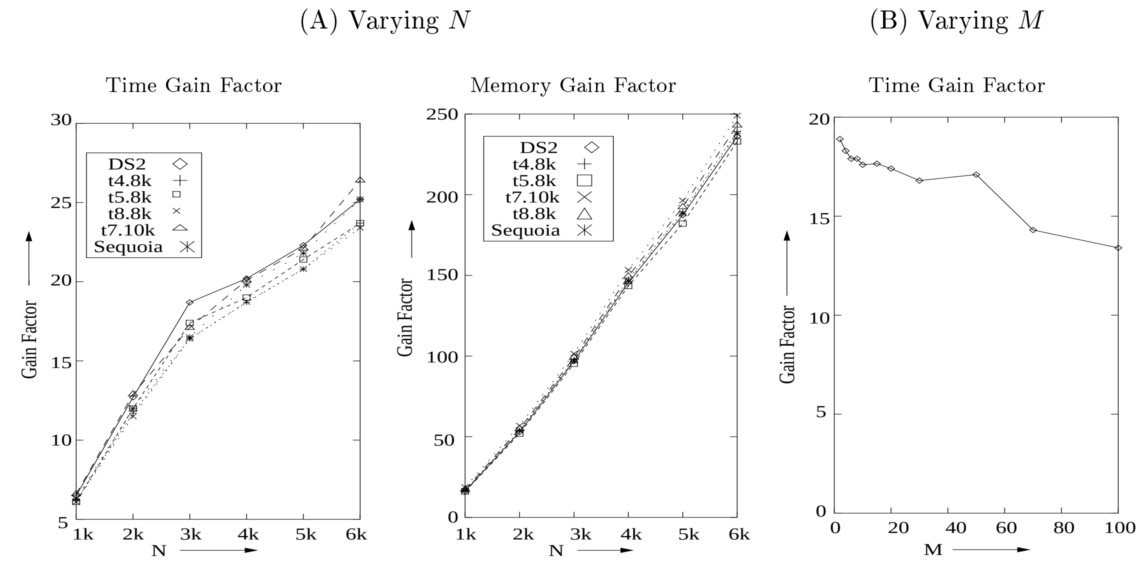


Figure 6: Stored Matrix: Time and Memory Gain Factor ($= \frac{Traditional}{Proposed}$)

that the time gain factor increases with N which is expected from the complexity analysis of Section 3.2.1. As N increases, the time complexity of the traditional algorithm $O(N^2 \log N)$ increases faster than that of the proposed algorithm $\sum_{j=1}^{s} ((n_j - n_{j+1})O(\frac{n_j}{p_j} \log \frac{n_j}{p_j}))$ (after simplification).

We also note that the memory gain factor increases with N as well. The complexity study in Section 3.2.1 supports this. As N increases, the memory complexity of traditional algorithm $O(N^2)$ increases faster than that of the proposed algorithm $\max_{j=1...s} O(\frac{{n_j}^2}{p_j})$.

Varying M: The number of dimensions M is varied from 2 to 100 for DS2 dataset while N is set to 3k. Figure 6 (B) shows the time gain factor (memory comparison is not reported as increase in M does not affect the memory required for priority queues which only stores the dissimilarities). Increasing M does not affect the time performance of the priority queue algorithm much as the dominant time consuming step of updating priority queues is mostly independent of M. Hence, the gain factor does not increase and instead remains unchanged or decreases a little.

Stored Data

We report time performance of stored data type for both memory and disk based algorithms in Figures 7 (A) and (B) respectively. Memory comparison is not shown as the memory requirements of the two algorithms are almost equal.

Memory Based

Varying N: N is varied from 3k...100k while M is 2. As expected, the results show that the time consumed by

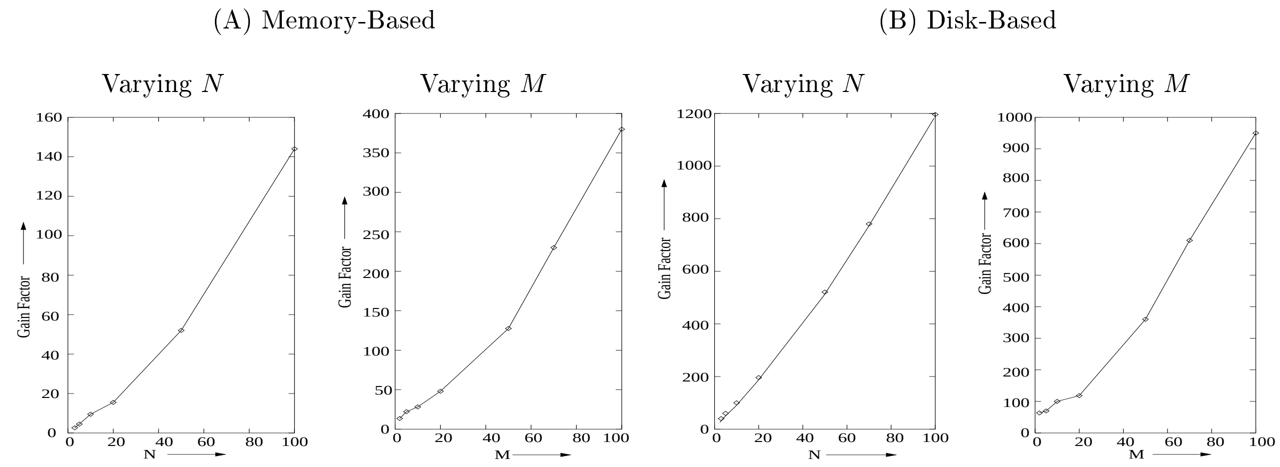


Figure 7: Stored Data: Time Gain Factor

the traditional algorithm with complexity of $O(\alpha N^2)$ increases much faster than that of the proposed algorithm with complexity $\sum_{j=1}^{s} ((n_j - n_{j+1})O(\alpha * \frac{n_j}{p_j}))$.

Varying M: M is varied from 2...100 while N is 16k. In contrast to the results for stored matrix where the time gain factor did not increase with M, in stored data memory-based type the gain factor increases rapidly with increasing M. The reason is M has a direct effect on the performance of stored data type where dissimilarities are calculated on demand. This is unlike stored matrix type where a dissimilarity matrix is maintained.

Disk Based

Varying N: Like the memory-based type, we note that the gain factor increases with N. However, it is much higher. The reason is as the data is accessed from disk, I/O time is included which is much higher for the traditional than for proposed algorithms.

Varying M: Similarly, for varying M, the gain factor for disk based algorithm is more than twice that for memory-based type.

3.3.3 Datasets from UCI ML Repository

We evaluated our method on a number of Irvine datasets [4] that has continuous attribute values, no missing value and relatively large N. We present the results of three datasets here. They are artificial character, image segmentation and pen digits. See Appendix B for a description of these datasets. We applied stored data memory-based method as the dissimilarity matrix size for traditional algorithm exceeded the memory size. The following table shows that our method has significant cpu time gain factors over traditional methods.

Datasets	N	M	Gain Factor ($= \frac{traditional}{proposed}$)
Character	61,313	7	47.4
Pen Digits	10,992	16	9.4
Image Segmentation	2310	19	4.5

3.3.4 Discussion

The previous sections showed that POP with hpc-tree is significantly better than earlier HAC algorithms both in time and memory performances. The proposed method leads to efficient parallel HAC as well. Existing parallel HAC algorithms distribute rows of dissimilarity matrix or priority queues (in 'stored matrix' type), and the data points (in 'stored data' type) among the processors. This is inefficient compared to parallelization using POP where each cell is completely assigned to a processor. In case of 'stored matrix' type, the existing methods distribute $O(N^2)$ dissimilarities whereas parallelization using POP requires to distribute only $p \times O((\frac{N}{p} + \|\delta\|)^2)$. Moreover, the updating effort in case of POP is limited to only a few processors that hold the affected cells, whereas the existing methods require to update each processor. In [8] we give details of this comparison. The proposed method is also applicable, with a little modification, to other clustering algorithms such as density-based clustering DBSCAN [12] and a modified single-link algorithm CURE [15]. Due to page limits we refer the interested readers to [8] for detailed study on the above. Finally, in general POP is suitable for clustering algorithms that are based on neighborhood search. POP's suitability comes from the fact that, for a given δ, cells can be searched independently. This reduces the number of required search by a factor close of the number of partitions without compromising the accuracy.

In the next section the 90-10 rule is applied for efficient cluster validation. The turning point of the distance plot which serves as an ideal δ for efficient clustering is shown to be useful in overcoming the drawbacks of existing validation methods.

4 Efficient Cluster Validation

In most real-world applications of clustering, one faces difficulty in choosing the *number of clusters* as virtually all clustering methods provide little, if any, information as to the number of clusters present in the data. Excluding such clustering algorithms that suggest a K value, for example, OPTICS [2] and edge-centric clustering [11], most methods require cluster validation to determine an 'optimal' (if there exists any) K number of clusters. For example, partitional clustering algorithms such as K-means, EM – Expectation Maximization, and K-medoids require the user to input a K value. Similarly, hierarchical clustering algorithms that outputs a dendrogram showing clustering for all $k = 1...N$ do not address the problem of selecting optimal K. Typically, after a clustering algorithm partitions the data into clusters, a validation method evaluates the clustering using a criterion function. Output of HAC, the dendrogram, is often preferred over partitional clustering such as K-means because the dendrogram already contains clusterings for all $k = 1...N$ while one needs to run K-means N times to get the N number of possible clusterings. See [10, 13, 24] for surveys on cluster validation methods. If not otherwise mentioned, a *level* of the dendrogram and an *iteration* of HAC are used to mean the same thing in this section.

This section describes two drawbacks of existing validation methods – (1) high time complexity and (2) inaccuracy due to distracting patterns, and propose solutions using the 90-10 rule to overcome them. The rest of the section is organized as follows. The next section describes the two drawbacks. Section 4.2 describes the proposed algorithms. Section 4.3 gives the experimental results.

4.1 Drawbacks of Existing Validation Methods

Gordon [13] divided validation methods into *global* and *local* types. Global methods evaluate all clusters in each level of the dendrogram, i.e., for $k = 1...N$. These methods seek the optimal (maximum or minimum) value among all levels. Local methods are based on tests of whether or not a pair of clusters should be agglomerated, or a single cluster should be subdivided. Unlike global methods, they are thus restricted to the analysis of hierarchically-nested sets of partitions, and use only a part of the dataset. Most validation methods in the literature are global type. For example, among 30 validation methods compared in Milligan and Cooper [24] study, 22 are global, and 7 out of the remaining 8 local methods test whether a pair of clusters should be agglomerated (we denote these local methods as local-A) and only one method tests whether a cluster should be subdivided.

Time Complexity Because of the within cluster sum of squares (defined in the next paragraph) calculation, time complexity of most global clustering criteria is at least $O(N)$ for evaluating a single level of the dendrogram. So, to evaluate all N levels, the time complexity of global methods is at least $O(N^2)$. There are higher complexities though, e.g.

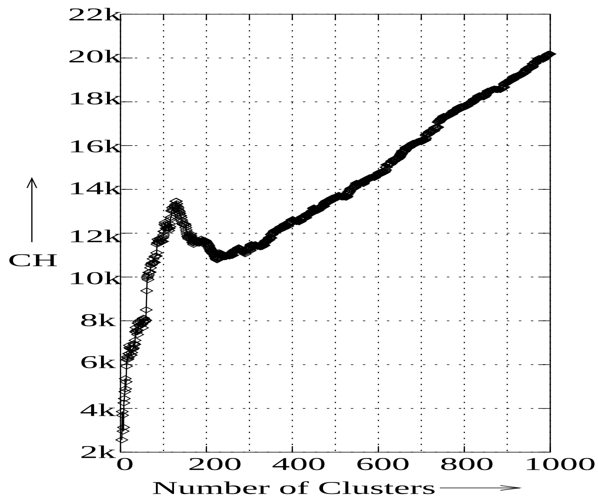


Figure 8: Distracting Pattern of Calinski and Harabasz Index (CH)

Silhouette index [21]. Local-A methods, in comparison, require less time as they need to evaluate only a part of dataset for each level. When N is large, global, and to some extent local-A, methods become computationally expensive.

Inaccuracy due to Distracting Patterns Another drawback of many validation methods is that they show some distracting patterns for lower levels of dendrogram, i.e., for large k, leading to inaccurate estimation of K. The term 'distracting' means 'to turn away from the original trend'. This is typically the case with global methods. As an example, we show an experimental study of the rank one validation method of Milligan and Cooper study. It is CH index by Calinski and Harabasz [7] which is computed as:

$$CH(k) = \frac{B(k)/(k-1)}{W(k)/(N-k)} \tag{1}$$

where $W(k) = \sum_{j=1}^{k} \sum_{X_i \in \chi_j} (X_i - m_j)^2$ is the within cluster sum of squares, $B(k) = \sum_{j=1}^{k} n_j \times (m_j - m)^2$ is the between cluster sum of squares for k clusters, $m_j = \frac{1}{n_i} \sum_{X_i \in \chi_j} X_i$ is the mean or centroid of cluster χ_j with n_j points, and $m = \frac{1}{N} \sum_{i=1}^{N} X_i$ is the mean of the whole data.

Figure 8 shows CH values for the 2-D dataset of Figure 2 which has 5000 points in 100 clusters. Clusters are randomly distributed in the 2-D space. Centroid type HAC is used to create the dendrogram (In Section 2 a definition of centroid method is given). In the figure we show CH values only for $k = 1...1000$ as CH keeps increasing further for $k > 1000$. Notice that, at around $k = 120$ (which is close to 100, the actual number of clusters), CH value is higher than the surrounding k values giving a close estimate of the actual number of clusters; but when k reaches 450 the CH value exceeds that at $k = 120$. Hence, if all levels of the dendrogram are evaluated and the level corresponding to maximum CH value is returned then the close estimate of $k = 120$ is missed. In this experiment clusters with only one data point are removed while calculating CH value as the distraction becomes even greater if single point clusters are included. This and other types of distraction (e.g. spikes) occur for almost all global methods we tested, particularly for large and noisy datasets, with large number of clusters and/or with clusters that are not very well formed. Some examples of the global methods that showed distraction are given below.

- Silhouette index is proposed by Kaufman and Rousseeuw in 1990 [21]. Silhouette statistic is defined by $s(i) = \frac{b(i)-a(i)}{max(a(i),b(i))}$ where $a(i)$ is the average distance of a data point to other points in the same cluster, and $b(i)$ the average distance to points in the nearest cluster besides its own. The authors propose to choose the optimal number of clusters k as the values maximizing the average $s(i)$ over the dataset.
- Krzanowski and Lai [22] proposed the quantity $DIFF(k) = (k-1)^2 W(k-1) - k^{\frac{2}{M}} W(k)$ and chose k to maximize $KL(k) = \| \frac{DIFF(k)}{DIFF(k+1)} \|$.
- C-Index [18] is computed as $CI = \frac{W(k)-min(W(k))}{max(W(k))-min(W(k))}$. The minimum values across the hierarchy levels is used to indicate the optimal number of clusters.

In the above discussion one can notice that the clustering criteria are proposed very early in 1970s and 80s. Indeed the survey on cluster validation by Gordon in 1999 [13] supports this fact.

Culprit It is difficult to pin-point any single factor as the culprit for the distracting pattern. In general it is an interplay of within and between cluster sum of squares.

Problem Statement The above drawbacks raise the question:

what is the range of levels in a dendrogram that are useful to validate.

Existing Approaches Existing approaches to overcome these drawbacks are largely ad-hoc. Milligan and Cooper [24], in their study of 30 clustering criteria, solved the problem by evaluating only the top 25 levels of the dendrogram where the synthetically generated datasets are known to have fewer than 25 clusters. When dataset size is small it may not be very difficult to estimate an initial level such as 25 in their case. But, as shown by the distracting pattern example, when dataset is large and complex, one needs more systematic and accurate methods. Related work is reported in [14, 23] where the authors proposed methods to find genuine clusters from a dendrogram. Although this reduces the size of the dendrogram, but it does not overcome the two drawbacks.

4.2 Proposed Methods

The above two drawbacks can be solved by estimating an initial iteration (denoted as I_V) for validation. Possible approaches include guessing a large fraction of N as I_V because number of clusters is typically very small as compared to N. But our experiments with the CH and other methods show that it is not easy to guess when number of clusters is large. We propose a systematic method to estimate I_V based on the 90-10 rule of HAC. The goal is to systematically and cheaply estimate I_V and to perform more costly validation only for the remaining $N - I_V$ iterations, thus improving the accuracy by avoiding the distracting patterns and reducing the over-all complexity from quadratic to almost linear.

We discussed the 90-10 rule about HAC in Section 2 which states that in a dendrogram, most levels from the bottom merge pairs of very small clusters separated by a very small portion of the maximum closest pair distance. Notice that the mirror-L shape of the distance plot in Figure 2 shows that until the sharp turning point in the right side, the closest pair distance is very small. We argue that a user can safely skip the initial iterations until this turning point because earlier clusters are too small in size and too close to other clusters to be proper clusters just by themselves. Small but well separated clusters are not skipped by this procedure because of their large distance from other clusters. Experimental results support this claim. The iteration corresponding to this sharp turning point is the estimated I_V. Below we describe two methods to detect I_V automatically.

Histogram Method In this method a histogram consisting of b bins is created over the closest pair distance. Frequency of each bin is determined by assigning the distances to their corresponding bins and counting them. Typically the histogram has a few densely populated bins in the beginning where closest pair distance is small and a large number of sparse or empty bins afterwards. Using a simple procedure one can skip the bins from the beginning with frequencies higher than a threshold frequency f until a bin with the threshold frequency or less is found. The first iteration corresponding to that bin is output as I_V. The algorithm is given in Table 4.

Parameter Setting In the histogram method in Table 4, b is the number of bins and f is the frequency threshold. As the maximum closest pair distance is typically much larger than the distance at the turning point, for a good estimation the bin width should be small, i.e., b should be large. If b is very large then a smaller I_V is returned in most cases because

Algorithm: Histogram Method to estimate I_V
Input: Dataset, f, b
Output: Initial Iteration for Validation (I_V)
1. Perform HAC and obtain distance plot
2. Create Histogram H of b bins of equal width
3. for $i = 1...b$
4. if $frequency[i] \leq f$
5. return the first iteration corresponding to bin i

Table 4: Histogram Method

smaller widths of bins result in early termination. In the experiments we set b to 100, i.e., bin width is 1% of the maximum closest pair distance. Parameter f is the threshold frequency which is used to stop the algorithm. It should typically be small because a large f returns a smaller I_V due to early termination. A default value is 0 which we use in all experiments.

Slope Method It is based on the observation that slope of the distance plot changes very little until the turning point after which it increases significantly (see Figure 2). The algorithm is given in Table 5. $CalculateSlope()$ measures slope by linear regression: at any iteration a total of $2 * l + 1$ iterations (l before and l after the iteration) are taken for regression. By comparing the current slope with the average of l previous slopes one can estimate the I_V: if the current slope is higher by a threshold percentage ($t\%$) then the corresponding iteration is output as the estimated I_V. X and Y values of the distance plot are normalized for this method.

Parameter Setting In the slope method in Table 5 parameter l is used for linear regression. A large l typically leads to a smaller I_V as separation between current slope and the average of l previous slopes increases. In all experiments we set $l = 10$. Parameter t is a stopping criterion of the algorithm. A large t leads to large I_V because the required jump of $t\%$ occurs towards later iterations. In all experiments we set $t = 10\%$.

Except these parameters, for datasets with identical points, one should skip the initial iterations with zero closest pair distance.

Algorithm: Slope Method to estimate I_V
Input: Dataset, l, t
Output: Initial Iteration for Validation (I_V)
1. Perform HAC and obtain distance plot
2. Normalize distance plot
3. for $iter = l + 1...N - l - 1$
4. $\quad slope = calculateSlope(iter)$
5. $\quad$ if $slope > avg_{previous\_slopes}$ by more than $t\%$
6. $\quad\quad$ return $iter$

Table 5: Slope Method

Complexity Analysis Although both algorithms can be intertwined with the HAC of step 1, we treat them separately for the ease of understanding.

Histogram Method The steps 2–5 of the histogram method in Table 4 (excluding the step 1 that performs HAC) estimates I_V. Populating the histogram in step 2 is done in $O(N)$ time using a simple hash function to locate the bins. Determining the bin that exceeds the threshold f is of complexity $O(b)$. As b is typically much smaller than N so the over-all complexity is $O(N)$. Memory complexity of steps 2–5 is of complexity $O(b)$ because only the bins need to be in the memory.

Slope Method The steps 2–6 of the slope method in Table 5 (excluding the step 1 that performs HAC) estimates I_V. It has time complexity $O(l * N)$. This complexity can be easily reduced to $O(N)$ by adding the new iteration and deleting the oldest one in the regression computation. As only $O(l)$ distances need to be in memory at any time, the memory complexity is $O(l)$.

Hence, both methods have linear time complexity in N.

Comparison with Existing Approaches As existing approaches are largely *ad hoc*, we compare the proposed methods with the original time complexity which is $O(N^2)$ for CH criterion. In the proposed methods, after estimating I_V, validation is done only for $N - I_V$ iterations. For example, let us say a data set consists of 100 points ($N = 100$), and after performing HAC the sharp turning point of the distance plot (I_V) is estimated to be 90. Then validation is done only for $N - I_V = 10$ iterations (i.e., 91–100) leading to a reduction of 90% computation. The new over-all time complexity for both methods

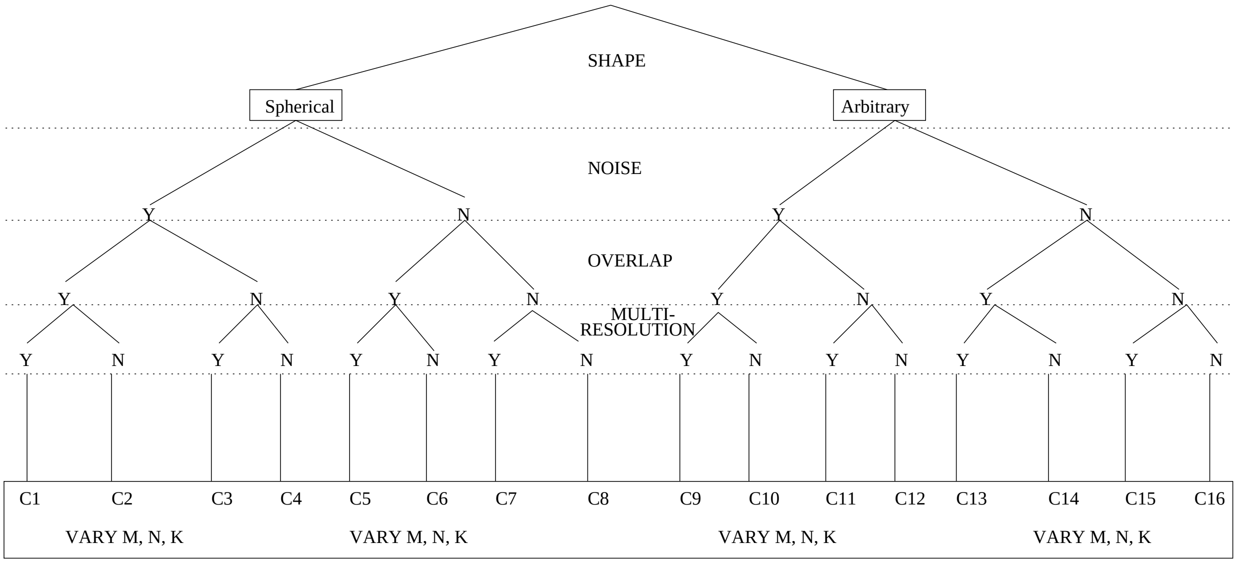


Figure 9: Test Bed for Cluster Validation

is $O(N) + O((N - I_V) \times N)$. If $N - I_V \ll N$ (which is typically the case for large data sets), *the over-all time complexity reduces from quadratic to almost linear.*

Note that we did not include HAC complexity in the analysis as validation does not include clustering. The previous section gives detailed analysis of HAC complexities.

4.3 Experimental Study

The aim is to determine whether the proposed methods are able to avoid the distracting patterns and reduce the computation by estimating K' $(= N - I_V)$ a little larger than the actual K number of clusters. Centroid method is adopted for HAC. We use CH as the clustering criterion for all experiments. A method is *accurate* if the distracting pattern of CH is avoided and $K' > K$. *Efficiency* of a method is defined as the percentage of levels rejected, i.e., $100 \times \frac{I_V}{N}\%$.

4.3.1 Datasets

In addition to testing the proposed methods over the datasets used in Section 3, we generated datasets of different characteristics using a test-bed for a more comprehensive experimental study.

Test-bed Performance of a validation method varies with data characteristics. Some important data characteristics are: *shape of clusters (spherical or arbitrary), noise, overlapping among clusters, and resolution (single or multiple).* Other features such as size (N), dimensionality (M), and number of clusters (K) are the common variants of the above characteristics. In Figure 9 we show the test-bed in a hierarchical manner. For the ease of understanding, the leaf levels are named as C1 to C16. Spherical clusters are produced by Gaussian distribution. For noisy data we add 5–10% uniformly generated data points as noise. Overlapping clusters are produced by adjusting mean and variance values for the Gaussian distributions. Multi-resolution clusters are produced by varying the number of points of clusters without changing the variance. M is set to 2 and 20; N is set to 20k and 50k; and K is set to 10 and 100.

4.3.2 Results

Tables 6 and 7 show the results for spherical clusters (C1 to C8). [2] A row in the table shows number of dimensions (M), number of data points (N) and actual number of clusters in the dataset (K). The last 6 columns show results ($K'(= N - I_V)$, 'accuracy' (Yes or No), and efficiency ($= 100 \times \frac{I_V}{N}$)) of slope and histogram methods. When 'accuracy' is 'No' the efficiency is '-' meaning efficiency is irrelevant in such cases. Table 8 gives results for bench-mark datasets and data containing arbitrary clusters in 2-D.

Summary of Results Notable results are:

- slope method is 'accurate' for all datasets while histogram method returns $K' < K$ (i.e., not 'accurate') in 13 out of the 70 datasets;
- histogram method is wrong mostly for datasets containing overlapping clusters because close proximity between clusters populate some bins which otherwise would have frequency less than or equal to threshold;
- efficiency of both methods is more than 95% for all datasets;
- histogram method has higher efficiency than slope method in 43 out of 70 datasets;
- results can vary by fine tuning the parameters of both algorithms but the important trend remains, i.e., slope method, although a little less efficient, is most of the time 'accurate' while the more efficient histogram method can lead to 'inaccurate' results for some datasets. Hence, we recommend slope over histogram method.

[2] To represent categories C9 to C16, datasets of previous section are used.

M	N	K	Slope			Histogram			Slope			Histogram		
			K'	Acc	Eff(%)	K'	Acc	Eff(%)	K'	Acc	Eff(%)	K'	Acc	Eff(%)
			C1						C2					
2	20k	10	150	Y	99.2	36	Y	99.8	150	Y	99.2	36	Y	99.8
		100	210	Y	98.9	14	N	-	230	Y	98.8	39	N	-
	50k	10	310	Y	99.3	21	Y	99.9	210	Y	99.5	19	Y	99.9
		100	250	Y	99.5	18	N	-	230	Y	99.5	22	N	-
20	20k	10	250	Y	98.7	218	Y	98.9	250	Y	98.7	214	Y	98.9
		100	310	Y	98.4	243	Y	98.8	310	Y	98.4	241	Y	98.8
	50k	10	370	Y	99.2	417	Y	99.1	370	Y	99.2	431	Y	99.1
		100	410	Y	99.1	461	Y	99.0	430	Y	99.1	462	Y	99.0
			C3						C4					
2	20k	10	150	Y	99.2	22	Y	99.9	150	Y	99.2	34	Y	99.8
		100	310	Y	98.4	101	Y	99.5	310	Y	98.4	103	Y	99.5
	50k	10	250	Y	99.5	23	Y	99.9	250	Y	99.5	15	Y	99.9
		100	350	Y	99.2	101	Y	99.7	350	Y	99.7	101	Y	99.7
20	20k	10	250	Y	98.7	204	Y	99.0	250	Y	98.7	205	Y	98.9
		100	210	Y	98.9	296	Y	98.5	210	Y	98.9	298	Y	98.5
	50k	10	370	Y	99.2	428	Y	99.1	370	Y	99.2	436	Y	99.1
		100	470	Y	99.0	511	Y	98.9	470	Y	99.0	518	Y	98.9

Table 6: Results of Spherical Clusters with Noise (C1–C4)

- Among the results of slope method, note that for non-noisy (i.e., C5 to C8) and high-dimensional (i.e., $M = 20$) test-bed datasets, the estimated K' is close to K; e.g. for C5 type with $M = 20$, $N = 10k$, and $K = 100$, the estimated K' is 110. Results for high-dimensional benchmark datasets (e.g. Pendigit, image segmentation, and character) also show similar close estimates. The reason is with increasing dimensionality the difference between intra-cluster and inter-cluster distances increases, while difference among the intra-cluster distances decreases and difference among the inter-cluster distances decreases as well. This leads to good estimation of the actual number of clusters by the proposed method. Interested readers may refer to [8] for a detailed reasoning on the above.

M	N	K	Slope			Histogram			Slope			Histogram		
			K'	Acc	Eff(%)	K'	Acc	Eff(%)	K'	Acc	Eff(%)	K'	Acc	Eff(%)
			C5						C6					
2	20k	10	210	Y	98.9	11	Y	99.9	210	Y	98.9	12	Y	99.9
		100	150	Y	99.2	36	N	-	210	Y	98.9	20	N	-
	50k	10	210	Y	99.5	11	Y	99.9	210	Y	99.5	17	Y	99.9
		100	150	Y	99.6	38	N	-	230	Y	99.5	22	N	-
20	20k	10	30	Y	99.8	10	Y	99.9	30	Y	99.8	8	N	-
		100	110	Y	99.4	51	N	-	110	Y	99.4	75	N	-
	50k	10	30	Y	99.9	11	Y	99.9	30	Y	99.9	16	Y	99.9
		100	110	Y	99.7	54	N	-	110	Y	99.7	52	N	-
			C7						C8					
2	20k	10	210	Y	98.9	15	Y	99.9	210	Y	98.9	17	Y	99.9
		100	310	Y	98.4	103	Y	99.5	310	Y	98.4	108	Y	99.4
	50k	10	250	Y	99.5	14	Y	99.9	210	Y	99.5	11	Y	99.9
		100	310	Y	99.3	104	Y	99.7	310	Y	99.3	102	Y	99.7
20	20k	10	30	Y	99.8	12	Y	99.9	50	Y	99.7	19	Y	99.9
		100	110	Y	99.4	101	Y	99.5	110	Y	99.4	121	Y	99.4
	50k	10	30	Y	99.9	11	Y	99.90	30	Y	99.9	31	Y	99.9
		100	110	Y	99.7	103	Y	99.7	110	Y	99.7	157	Y	99.6

Table 7: Results of Spherical Clusters without Noise (C5–C8)

5 Conclusion and Future Work

In this paper we discussed a 90-10 rule between the number of iteration of HAC and the closest pair distance which states that in a dendrogram, most levels from the bottom merge pairs of very small clusters separated by a very small portion of the maximum closest pair distance. Using this rule the time and memory requirement of existing HAC algorithms are reduced by a factor of the number of partitions. By detecting the turning point of the distance plot existing drawbacks (high time complexity and inaccuracy due to distracting patterns) of cluster validation methods were overcome. Analysis and experiments supported these claims.

A future direction of the present work is to apply the 90-10 rule to find natural clusters. HAC methods often generate clusters that are not natural. The reasons, as pointed out in [20], are that the similarity measure used to merge cluster pairs

Dataset	M	N	K	Slope			Histogram		
				K'	Acc	Eff(%)	K'	Acc	Eff(%)
Iris	4	150	3	55	Y	70.0	23	Y	85.0
Segmentation	19	2310	7	33	Y	98.6	12	Y	99.5
Character	7	61,313	8	22	Y	99.9	46	Y	99.9
PenDigit	16	10992	10	20	Y	99.8	30	Y	99.7
t4.8k	2	8k	6	47	Y	99.4	25	Y	99.7
t5.8k	2	8k	6	51	Y	99.4	28	Y	99.6
t8.8k	2	8k	8	46	Y	99.4	23	Y	99.7
t7.10k	2	10k	9	70	Y	99.3	35	Y	99.6

Table 8: Results of UCI Repository Datasets and Arbitrary 2-D Datasets with noise

consider only the *aggregate inter-connectivity* between the pairs of clusters as in average link method or only the *closeness* between pairs of clusters as in single link method. Karypis et al. proposed a new algorithm called CHAMELEON [20] that is based on a combined similarity metric of inter-connectivity and closeness. It is a two phase algorithm where in the first phase, data is split into a number of small clusters and in the second phase it merges these small clusters using the combined similarity measure. The 90-10 rule can play a role in the phase one of CHAMELEON. The amount of computation required to decide which cluster pair to merge is less than that to decide which cluster to split [10]. This is also a reason why agglomerative hierarchical clustering that merges the closest pair in each iteration has been used more frequently than the divisive type where the best cluster is chosen to split. By using the 90-10 rule we plan to stop the merging process at the sharp turning point of the distance plot. The clusters at that point can be given as input to the phase 2 of CHAMELEON to extract natural clusters. Another future direction is to extend the proposed methods to cluster streaming data. Streaming data where data arrives continuously in a possibly infinite stream requires dynamic clustering. By maintaining and dynamically updating the clusters at the sharp turning point of the distance plot, it is possible to output valid clusters at any time of the streaming data processing.

References

[1] M. R. Anderberg. *Cluster Analysis for Applications.* Academic Press, New York, 1973.

[2] M. Ankerst, M. Bruenig, H.-P. Kreigel, and J. Sander. OPTICS:ordering points to identify the clustering structure. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, June 1999.

[3] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 217–235, 1999.

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

[5] P. S. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 9–15, 1998.

[6] S. Brin. Near neighbor search in large metric spaces. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1995.

[7] R. B. Calinski and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics*, 3:1–27, 1974.

[8] M. Dash. *Efficient Clustering for Knowledge Discovery in Large High-Dimensional Databases.* PhD thesis, Department of Computer Science, School of Computing, National University of Singapore, 2002.

[9] W. H. E. Day and H. Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification*, 1(1):7–24, 1984.

[10] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*, chapter Unsupervised Learning and Clustering. John Wiley & Sons, 1973.

[11] S. Epter and M. Krishnamoorthy. A multiple-resolution method for edge-centric data clustering. In *Proceedings of International Conference on Information and Knowledge Management (CIKM)*, 1999.

[12] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996.

[13] A. Gordon. *Classification (2nd edition).* Chapman and Hall/CRC press, London, 1999.

[14] A. D. Gordon. Clustering algorithms and cluster validation. In P. Dischdedt and R. Ostermann, editors, *Computational Statistics: Papers collected in the occasion of the 25th Conference on Statistical Computing*, pages 497–512. Heidelberg: Physica-Verlag, 1994.

[15] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 73–84, 1998.

[16] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.

[17] A. Hinneburg and D. A. Keim. Optimal grid-clustering: towards breaking the curse of dimensionality clustering. In *Proceedings of International Conference on Very Large Databases (VLDB)*, 1999.

[18] L. J. Hubert and J. R. Levin. A general framework for assessing categorical clustering in free recall. *Psychological Bulletin*, 83:233–253, 1976.

[19] A. K. Jain and R. C. Dubes. *Algorithm for Clustering Data*, chapter Clustering Methods and Algorithms. Prentice-Hall Advanced Reference Series, 1988.

[20] G. Karypis, E-H. Han, and V. Kumar. CHAMELEON: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer*, 32:68–75, 1999.

[21] L. Kaufman and P. Rousseuw. *Finding Groups in Data - An Introduction to Cluster Analysis*. Wiley Series in Probability and Mathematical Statistics, 1990.

[22] W. J. Krzanowski and Y. T. Lai. A criterion for determining the number of groups in a dataset using sum of squares clustering. *Biometrics*, 44:23–34, 1985.

[23] I. C. Lerman and N. Ghazzali. What do we retain from a classification tree? an experiment in image coding. In E. Diday and Y. Lechevallier, editors, *Symbolic-Numeric Data Analysis and Learning*, pages 27–42. Nova Science, New York, 1991.

[24] G. Milligan and M. Cooper. An examination of procedures for determining the number of clusters in a data set. *Pshychometrika*, 50:159–179, 1985.

[25] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26:354–359, 1983.

[26] F. P. Preparata and M. I. Shamos, editors. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[27] M. O. Rabin. Probabilistic algorithms. In J. F. Traub, editor, *Algorithms and Complexity*, pages 21–39. Academic Press, New York, 1976.

[28] P. H. A. Sneath and R. R. Sokal. *Numerical Taxonomy.* W. H. Freeman and Company, San Francisco, 1936.

[29] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The SEQUOIA 2000 storage benchmark. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 2–11, 1993.

[30] Jr. J. H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58:236–244, 1963.

[31] R. Weber, H.-J Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 194–205, 1998.

[32] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 103–114, Montreal, Canada, June 1996.

A CF-tree

CF-tree is used in a clustering method called BIRCH [32]. BIRCH is especially suitable for very large databases with limited resources (i.e., available memory and time constraints). A CF (*C*lustering *F*eature) tree is a height-balanced tree with two parameters: branching factor B and threshold T. Each non-leaf node contains at most B entries of the form [CF_i,$child_i$], where $i = 1, ..., B$, "$child_i$" is a pointer to its i^{th} child node, and CF_i is the CF of the sub-cluster represented by this child. So a non-leaf node represents a cluster made up of all the sub-clusters represented by its entries. A leaf node contains at most L entries, each of the form [CF_i], where $i = 1, ..., L$. A leaf node also represents a cluster made up of all the sub-clusters represented by its entries. But all entries in a leaf node must satisfy a *threshold requirement*, with respect to a threshold value T: the diameter (or radius) has to be less than T.

Such a CF tree is built dynamically as new data objects are inserted. It is used to guide a new insertion into the correct sub-cluster for clustering purposes just the same way as a B+-tree is used to guide a new insertion into the correct position for sorting purposes. The CF tree is a very compact representation of the dataset because each entry in a leaf node is not a single data point but a sub-cluster (which absorbs many data points with diameter (or radius) under a specific threshold T).

B Datasets

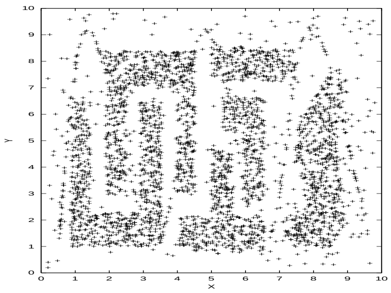
B.1 UCI ML Repository Datasets

Character dataset has 6,000 instances with each for a capitol letter (A, C, D, E, F, G, H, L, P, R) described by a set of segments (lines) which resemble the way an automatic program would segment an image. There are total 61,313 segments each with seven attributes: four attributes are coordinates of the segment, another two are length of the segment and diagonal of the smallest rectangle which includes the picture of the character respectively, the last attribute is superfluous.
Image segmentation dataset consists 2310 objects in 19 dimensions. Data objects are drawn from a database of 7 outdoor images (brick-face, sky, foliage, cement, window, path, grass). Each of the 7 images is represented by 330 objects.
Pen digit dataset consists of 10,992 objects in 16 dimensions. There are 10 classes corresponding to digits 0...9. The 16 dimensions are drawn by re-sampling from handwritten digits.
Iris dataset has 150 points in 4 dimensions in 3 clusters. Dimensions are sepal length, sepal width, petal length, and petal width. Clusters are Iris Setosa, Iris Versicolour, and Iris Virginia. Each of the 3 clusters is represented by 50 objects.

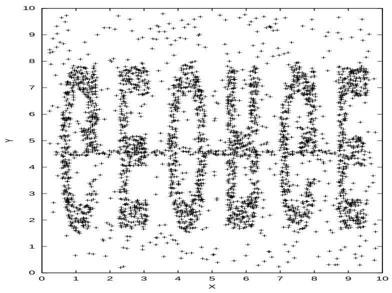
B.2 Other Datasets

These 2-D datasets having clusters of arbitrary shape and multiple resolution are taken from CHAMELEON [20], CURE [15], and BIRCH [32].

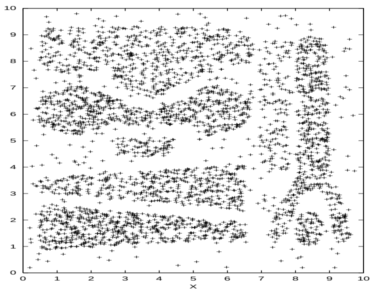
t4.8k



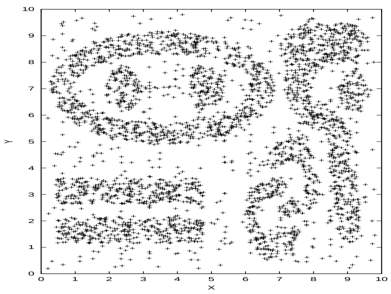
t5.8k



t8.8k



t7.10k



DS2 (2-D)