Grid-Clustering: A Fast Hierarchical Clustering Method for very Large Data Sets

Erich Schikuta

CRPC-TR93358
November 1993

Center for Research on Parallel Computing
Rice University
P.O. Box 1892
Houston, TX 77251-1892

Grid-Clustering: A Fast Hierarchical Clustering Method for very Large Data Sets

Erich Schikuta[1]

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

Abstract

This paper presents a new approach to hierarchical clustering of very large data sets, named Grid-Clustering. The method organizes unlike the conventional methods the space surrounding the patterns and not the patterns. It uses a multidimensional grid data structure. The resulting block partitioning of the value space is clustered via a topological neighbor search. The Grid-Clustering method is able to deliver structural pattern distribution information for very large data sets. It superceeds all conventional hierarchical algorithms in runtime behavior and memory space requirements. The algorithm was analyzed within a testbed and suitable values for the tunable parameters of the algorithm are proposed. A comparison of the executions times to other commonly used clustering algorithms and a heuristic runtime analysis is given.

1. Introduction

Clustering methods are extremely important for explorative data analysis, especially in areas which deal with real life data, like social, medical, behavioral or economic science. Many different algorithms have been proposed, which can be generally divided into hierarchical, like single-linkage, complete-linkage, etc. and partitional one, like K-MEANS, ISODATA, etc.(1). All these methods suffer from specific draw backs handling large numbers of patterns. The hierarchical methods give structural information, as dendrograms, but are only suitable for a small number of patterns. With growing numbers the computational expense magnifies, resulting of the calculation of a dissimilarity matrix, where each pattern is compared to all other. The partitional methods are to some extent not so power hungry, but lack methodical freedom by the necessity of a "good guess" of structural information, like the numbers and the positions of the initial cluster centers. If the choice of the initial clustering is not appropriate the partitional methods become very calculation extensive in computing new cluster centers too.

A number of different algorithms have been proposed to overcome one or another of these mentioned problems(2)(3)(4)(5)(6)(7)(8). Most of the algorithms compare the single patterns to each other or to a predefined cluster center. Out of a calculated distance metric they organize the patterns by combining them into clusters.

The hierarchical Grid-Clustering algorithm proposed in this paper uses a grid structure which, in contrary, organizes the value space surrounding the patterns. The value space is partitioned by rectangular blocks. Using the distribution information of the blocks the patterns are clustered.

1 Authors permanent address: Erich Schikuta, Institute of Applied Computer Science, Dept. of Data Engineering, University of Vienna, Rathausstr. 19/4, A-1010, Vienna, Austria

This algorithm reaches in practice an extreme gain in performance in comparison to conventional algorithms.

The paper is organized as follows. In section 2 the underlying idea is presented and the Grid Structure is described. The algorithm is defined in section 3. In section 4 we show our experiences with practical examples. Appropriate values for tunable parameters of the algorithm, a comparison of the run time to conventional algorithms and a heuristic run-time analysis is presented in section 5.

2. Grid-Clustering

2.1. Idea

All conventional cluster algorithms calculate a distance based on a dissimilarity metric (like the Euclidean distance, etc.) between patterns or cluster centers. The patterns are clustered accordingly to the resulting dissimilarity index.

The presented Grid-Clustering algorithm is different in that case that it doesn't organize the patterns but the value space, which surrounds the patterns*. To organize the value space, a variation of the multidimensional data structure of the Grid File is used, which we call Grid Structure. The patterns are treated as points in a d-dimensional value space and are randomly inserted into the Grid Structure. The points are stored according to their pattern values preserving the topological distribution. The Grid Structure partitions the value space and administrates the points by a set of surrounding rectangular shaped blocks.

Block: Let $X = (x_1, x_2, \dots x_n)$ be the set of n patterns. x_i is the i-th pattern consisting of a tuple of describing features $(a_{i1}, a_{i2}, \dots a_{id})$, where d is the number of dimensions. A block is a d-dimensional rectangular shaped cube containing up to a maximum of bs patterns (bs = block size). The following properties are satisfied, ϕ is the empty set

$$\text{for all } x_i,\ x_i \in B_j$$
$$B_j \cap B_k = \phi, \text{ if } j \neq k$$
$$B_j \neq \phi$$
$$\cup B_j = X$$

With other words, the patterns are disjointly partitioned among the blocks.

The proposed algorithm clusters the blocks B_i (and so the patterns X) into a nested sequence of nonempty and disjoint clusterings, where $(C_{u1}, C_{u2}, \dots C_{uw_u})$ is the u-th clustering. The initial situation (0-th clustering) is that each block is a cluster, i.e. $C_{0j} = B_j$, $j = 1, \dots b$ and $W_0 = b$.

The blocks can be seen as a preclustering phase or an initialization of cluster centers. The cardinality of these centers is dependent on the block size and is defined by $1 < p_B < bs$, where p_B is the number of patterns contained in block B.

The proposed Grid-Clustering algorithm uses this block information via the index structure of the Grid File and clusters the patterns according to their surrounding blocks.

* This differentiation can be found with data structures too, which can be divided into "data record organizing" (like trees, tries, etc.) and "value space organizing" (like hash tables, Grid Files, etc.) structures.

For example, the following figure shows the value space partition after the insertion of 1000 2-dimensional patterns with 3 major clusters. It is easily recognizable how the rectangular blocks adapt to the distribution of the patterns.

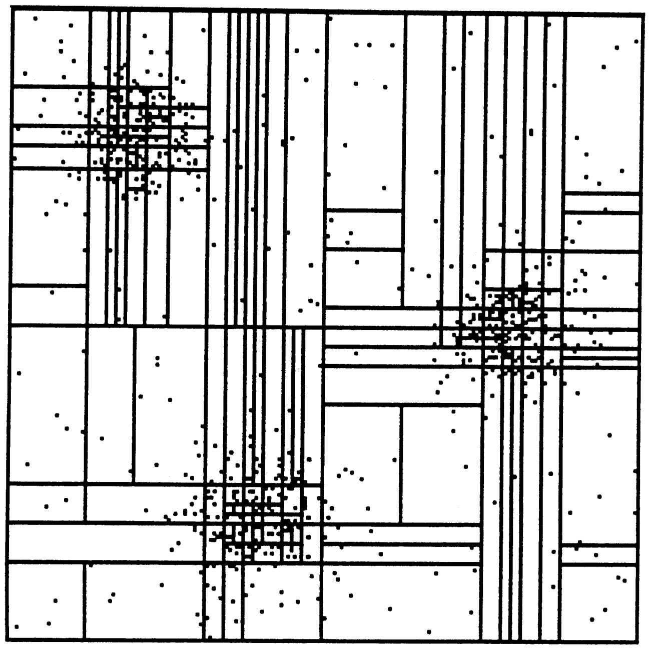


Figure 1: block structure for 1000 patters, 3 clusters

The algorithm calculates the density of each block via the numbers of patterns and the spatial volume of the block.

Spatial volume V_B of a block B is the cartesian product of the extents e of block B in each dimension, i.e.

$$V_B = \prod_i e_{Bi}, \quad i = 1, \dots d$$

Density D_B of block B is the ratio of the actual number of patterns p_B contained in block B and the spatial volume V_B of B, i.e.

$$DB = \frac{p_B}{V_B}$$

The blocks are sorted accordingly to their density. The result is a sequence $<B_{1'}, B_{2'}, \dots B_{b'}>$. i' denotates a permutation of the index i reflecting the sorted order.

It is obvious that a number of blocks will have the same density values and represent *ties*. For example the following blocks are ties:

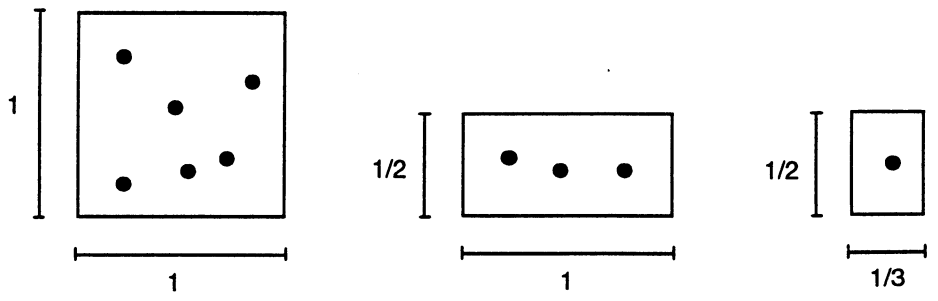


Figure 2: tie blocks

The blocks with the highest density (obviously with strongest pattern correlation) build the clustering centers. Iteratively the remaining blocks are clustered in sequence to their density, building new cluster centers or merging with existing clusters.

Only blocks, which adjoin a cluster, can be merged. Adjacent blocks are called *neighbor* blocks. The neighbors can be distinguished by the dimensionality of the connection. We call the neighbors adjacent via a d-1 dimensional hyperplane, the *nearest neighbors*. For simplification of the clustering algorithm we take only the nearest neighbors in account (see also the algorithm analysis).

A neighbor search is done starting at the cluster center, inspecting adjacent blocks, finding a neighbor and recursively proceed with this block. This search is similar to the traversal of a graph finding the spanning tree(9). The blocks represent the nodes. An edge between two node exists, if the respective blocks adjoin.

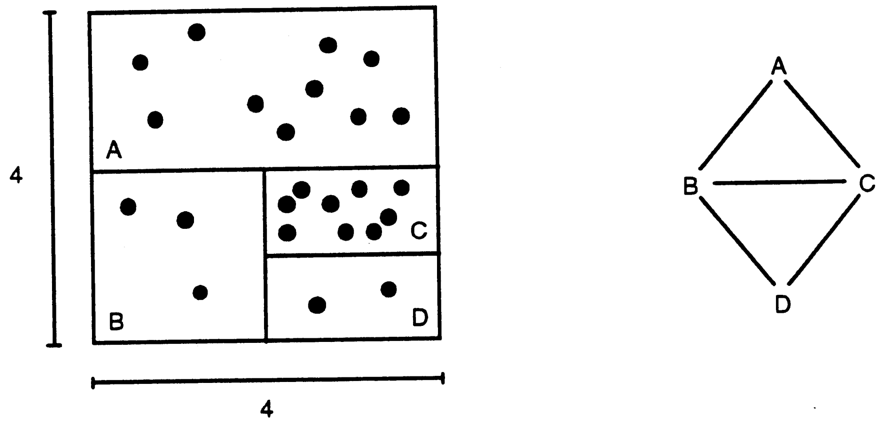


Figure 3: correspondence between block structure and graph

The traversal can be done by a depth-first-search (DFS) algorithm(10) inspecting only these blocks, which density values are smaller or equal to the value of the iteratively processed block.

Example:

According to Fig. 3 four blocks A, B, C, D exist with pattern numbers 10, 3, 9, 2 and spatial volume 8, 4, 2, 2 respectively. The density values are $V_A = 5/4$, $V_B = 3/4$, $V_C = 9/2$ and $V_D = 1$. The initial state depicts a unique cluster for each block (i.e. $C_{10} = A$, $C_{20} = B$, ...). In the first phase of the DFS algorithm C is the block with the highest density and establishes the first cluster center C_{11}. The next clustered block is A. Now the neighbor search starts with center C_{11}. A is merged with cluster C_{11}, because it is adjacent via a 1-dimensional hyperplane, i.e. an edge. The remaining blocks build separate clusters each. The next handled block is D. Starting from C (which represents the center of C_{11}) again D is merged with C and transitive with A. In the last phase B is merged with the cluster C_{31} consisting of the blocks A, C and D. The resulting nested sequence of the clustering process is given by the following table:

Membership of blocks to cluster respective to clustering level

clus. level / blocks	A	B	C	D
0-th	C_{01}	C_{02}	C_{03}	C_{04}
1-st	C_{11}	C_{12}	C_{11}	C_{13}

2-nd	C_{21}	C_{22}	C_{21}	C_{21}
3-rd	C_{31}	C_{31}	C_{31}	C_{31}

Respectively the patterns are clustered via their corresponding blocks. The dendrogram is easily derivable from this table.

For the special case of one pattern per block the algorithms clusters directly the patterns. Because of the properties of the Grid File structure this leads to an increased computational amount, but with a marginal improvement of the result. Sometimes the algorithm produces in these cases less meaningful artificial results. The choice of an appropriate number of patterns per block is of central importance to the algorithm. Suitable values are given in section 5. This makes is obviously that the algorithm delivers useful results for a large number of patterns only (>100).

2.2. The Grid Structure

The pattern space is partitioned into blocks using an adapted Grid File, the Grid Structure. The Grid File is a multidimensional data structure, which adapts gracefully to the distribution of patterns X in the value space V_x. The Grid Structure is a main memory data structure. It lacks the external disk storage support and the rich data manipulation facilities of the original Grid File.

The Grid Structure consists of d scales (for each dimension), the grid directory (a d dimensional array) and the b data blocks.

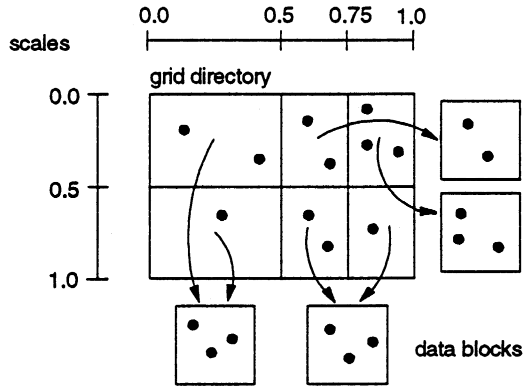


Figure 4: Schematic view of a 2-dimensional Grid Structure

The scale is a 1-dimensional array. Each value of this array represents a d-1 dimensional hyper plane. It partitions the value space of order d into two.

The grid directory is a d dimensional dynamic array and represents the grid partition produced by the d scales.

The data blocks contain the stored patterns. Each element of the grid directory corresponds to a data block. It is possible that two or more directory elements reference the same data block. The value space defined by the union of the directory elements referencing the data block i is called block region V_{Bi}. A block region has always the shape of a d-dimensional rectangular box.**

** In the the literature the term "data bucket" is used. We prefer "data block" to distinguish the main memory data blocks of the Grid Structure from the external storage data buckets of the the original Grid File.

Appropriate algorithms manipulating the scales, the directory and the blocks pertain the Grid File properties. During the insertion of a pattern the component values are compared against the scales and the directory index of the grid cell, which references the corresponding block to store the new pattern, is calculated. If this block overflows two different actions are performed depending on the type of the connected block region. If the block region consists of more than one directory cell, one of the intersecting scale boundary is chosen (commonly in a round robin way), the block is split accordingly into two and the patterns of the original block are distributed among these new blocks correspondingly. If the original block region consists only of one directory cell, a new scale boundary is inserted, which splits the block region into two. The new scale boundary can be chosen by a bisection of the block region (splitting into two equally sized regions) or a median split (about the same number of patterns in the two new buckets). The grid references along the new boundary are adjusted.

For an exact description refer to Nievergelt(11) or Hinrichs(12).

3. The Algorithm

3.1. Properties

According to Dubes(1) the Grid-Clustering algorithm can be classified as

exclusive (non overlapping clusters)
intrinsic (using pattern information only)
hierarchical (nested clustering)
agglomerative (starting with small groups)
polythetic (using all features)

Tie (patterns with the same dissimilarity index) are not only allowed, but are premise to the algorithm because of the block partition.

3.2. The Grid-Clustering Algorithm

According to the description in section 2 the proposed Grid-Clustering algorithm consists of 5 main parts. The numbers in brackets reference to the respective line of the algorithm, which is defined in the following section.

- Creation of the Grid Structure (1)
- Calculation of the block density (2)
- Sorting of the blocks (3)
- Identifying cluster centers (10)
- Traversal of neighbor blocks (14)

3.3. Algorithm GRIDCLUS

In the following we give a comprehensive definition of the algorithm GRIDCLUS. It consists of a main module, which iteratively processes all blocks, and a recursive procedure NEIGHBOR-SEARCH, which assigns the blocks to existing clusters.

The number of the actual run is stored in u and the number of clusters found so far in W[u]. After completion of a run W[u] stores the number of clusters in run u. C[u, v] is a set valued variable containing the clustered blocks of run u and cluster v.

To conform to the block definition each block builds a unique cluster automatically. This trivial situation is not handled by the algorithm, but it can be seen as run 0 with W[u] = b and C containing b clusters with one block.

The statements are numbered for referencing purpose.

```
Algorithm GRIDCLUS
(0)   Initialization
(1)   Create the Grid Structure
(2)   Calculate the block densities $D_{Bi}$
(3)   Generate a sorted block sequence S = <$B_{1'}$.. $B_{2'}$, ... $B_{b'}$>
(4)   Mark all blocks 'not active' and 'not clustered'
(5)   while a 'not active' block exist do
(7)       u := u + 1
(8)       Find active blocks $B_{i'}$ .. $B_{j'}$
(9)       for each 'not clustered' block $B_{k'}$ := $B_{1'}$ .. $B_{j'}$ do
(10)          create a new cluster set C[u]
(11)          W[u] := W[u] + 1
(12)          C[u, W[u]] <- $B_{k'}$
(13)          mark block Bk' clustered
(14)          NEIGHBOR-SEARCH($B_{k'}$, C[u, W[u]])
(15)      endfor
(16)      for each 'not active' block $B_l$ do
(17)          W[u] := W[u] + 1
(18)          C[u, W[u]] <- $B_l$
(19)      endfor
(20)      Mark all blocks 'not clustered'
(21)  endwhile
end GRIDCLUS

Procedure NEIGHBOR-SEARCH(B, C)
(22)  for each 'active' and 'not clustered' neighbor Bn of B do
(23)      C <- $B_n$
(24)      mark block $B_n$ clustered
(25)      NEIGHBOR-SEARCH($B_n$, c)
(26)  endfor
end NEIGHBOR-SEARCH
```

During the initialization phase (0) the blocksize and the splitting strategy (median-bisection) have to be defined. The variables u and W[u] are set to 0. The patterns are inserted into the Grid Structure randomly (1) and the block densities are calculated (2). The sorted block sequence S is created (3) according to decreasing density values. $B_{1'}$ is the bucket with the maximum density and $B_{b'}$ the bucket with the minimum one. i' denotes a permutation of the index i reflecting the sorted order. The block marks are initialized (4) for the while-statement (5) to (21), which is looping until all blocks are clustered. The active blocks (8) are the first 'not active' marked blocks in S with equal density values (subsequence <B_i, ... B_j>). The for-statement (9) to (15) clusters all active blocks, which are not clustered until now. $B_{k'}$ (12) is the initial block of the cluster C[u, W[u]]. In (14) the neighbor search is started to find adjoining active blocks. The for-loop (16) to (19) clusters each remaining block into a single cluster. It is given for algorithm completeness only and can be bypassed in practice to yield a better performance.

The procedure NEIGHBOR-SEARCH traverses the adjoining neighbors of B ((22) to (26)) to add active, not clustered blocks to cluster C (23). The '<-' operator includes block B into cluster set C. In (25) the neighbors of B_n (i.e. transitive neighbors of B) are checked recursively for addition to C.

4. Experiment

The GRIDCLUS algorithm has been implemented on an IBM/PC and was embedded into a highly interactive GridClus Data Analysis System, the GCDAS. The GCDAS gives the possibility to administrate the pattern set easily, to view the data distribution projected to choosable dimensions, to control the parameters of the clustering process and to perform it in a stepwise way. This system is described elsewhere(12).

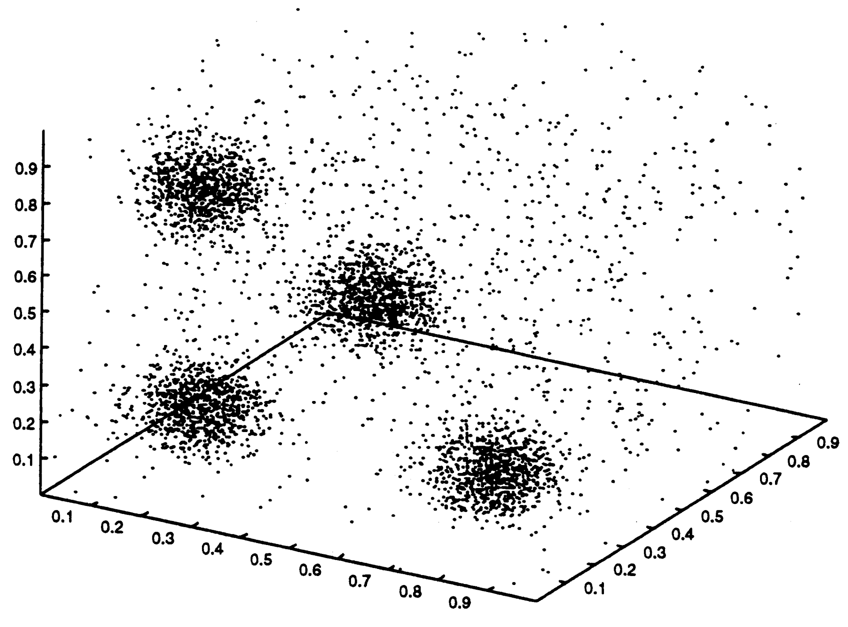
The explorative data analysis of real life data has often to deal with very large (number of patterns > 1000) and/or high dimensional (number of attributes > 3) data sets. Conventional hierarchical algorithms refuse a solution because of execution time and memory space exhaustion. Partitional methods need a good "guess" of the structural data information in advance, like number of clusters and initial cluster centers. In practice this information is supplied by the analyst by viewing the data, using his "most important tool", the human eye. In today available systems (like SPSS or SAS) only 2-dimensional projected views are supported . Analyzing data sets with many describing attributes or a "hiding" cluster distribution (clusters can not be recognized in projected views, because of overlapping positions) this is a difficult and often error prone task.

The GRIDCLUS algorithm is a solution to this type of problem.

In the following example we want to show such a problem. We created an artificial data set of 5000 3-dimensional pattern, where 80 % of them were clustered into 4 groups. The remaining 20 % were evenly distributed over the 3-dimensional value space representing noise. The cluster centers were arranged in that way that the clusters are "hiding" themselves in projected 2-dimensional views of the data set. The cluster centers are

Center \ Dimension	x	y	z
1	0.2	0.2	0.2
2	0.8	0.2	0.2
3	0.2	0.8	0.2
4	0.2	0.2	0.8

The following figure gives a 3-dimensional spatial picture of the pattern distribution and the respective 2 dimensional projections.



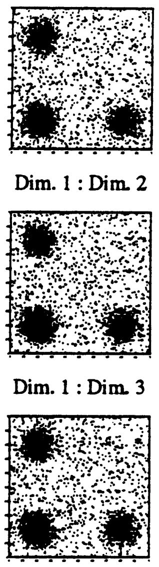


Figure 5: Artificial data set of 5000 3-dimensional patterns, 4 clusters

Only 3 clusters are recognizable by the projections because of overlapping pattern concentrations. The analytical view gives an erroneous guess about the real data distribution. This can lead to wrong initial values for a partitional clustering method. In this case the GRIDCLUS algorithm shows its advantages. Applied to this data set it produces the following dendrogram as a result. The figure is an original screen shot of the GCDAS, therefore the dendrogram lines (5000) build compact areas because of the insufficient screen resolution.

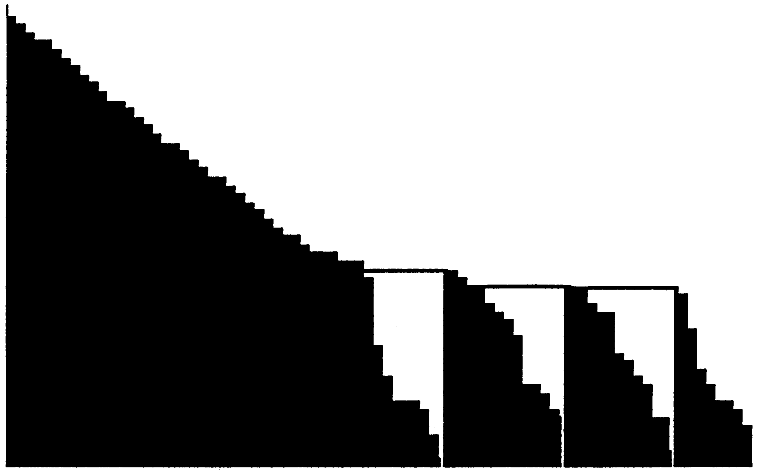


Figure 6: Dendrogram created by the GRIDCLUS algorithm.

Apparently the 4 clusters can be seen. Amazingly the algorithm needed less than 3 minutes on an IBM/PC for the calculation of the result. Easily it can be seen, how well the structural information of the data set is recognized.

5. Analysis

We performed an extensive test suite analyzing the tunable parameters of the algorithm and finding the most appropriate values yielding the best results. We applied the GRIDCLUS algorithm on data sets from 100 to 10000 elements using different algorithm parameters and data set distributions. Each run was performed several times with different randomly generated data sets of the same pattern distribution. The times shown are calculated means. We give a heuristic analysis of the tunable parameters and the execution characteristics of the GRIDCLUS algorithm*** .

The evaluation of the test results was done in a "subjective" fashion, due to the lack of appropriate objective methods. Cluster validity analysis(14) as statistical hypothesis, indices, etc. were not applicable, because of the computational problem to produce comparison structures. Therefore we used the approach to compare the results with the prepositions of the data set generation. The qualification "good" implies that the result reflects the original data set distribution well.

5.1. Tunable Parameters

During the definition of the GRIDCLUS algorithm 3 different tunable parameters have to be considered

- Splitting strategy (Bisection versus Median splitting)
- Bucket size
- Adjacency hyperplane dimension

The splitting strategy (bisection versus median) and the bucket size direct the partitioning of the value space during the creation of the Grid Structure and therefore influence the clustering to a certain degree. The dimensionality of the "touching" hyperplane decides on the adjacency of a neighboring block and controls directly the DFS-traversal.

5.1.1. Splitting Strategy

The Bisection (binary radix method) splits a block into two equally shaped blocks and divides the patterns according to their attribute values. The Median method splits a block into two blocks, containing the same number of patterns.

The test suite showed that Bisection outperformed the Median splitting. In a few test runs the median splitting produced incorrect results in finding to many or to less centers. Further it proved unstable to slightly different data sets. Occasionally it reflected the structure of the distribution very well in one data set and badly in the same set with additional 5% patterns. We deduce that from the fact that the Median splitting always splits groups of highly correlated data into two equally filled blocks. On the long run it produces a rather torn partitioning and doesn't keep groups together. With the Bisection this situation is avoided by the large number of patterns. This fact is one of the reasons to apply the algorithm to a large data set only.

*** This is based on the fact that the Grid File has proved itself extremely complex to a formal analysis. Only results for a few known data distributions exist (Regnier(13)), which are obviously not applicable to the mentioned situation.

5.1.2. Bucket Size

The GRIDCLUS algorithm handles all pattern in a common block as ties. With a large number of pattern, these situation proved not to be a drawback. A large bucket size results in few blocks covering large areas of the value space. In contrary small bucket sizes (near 1) produce an artificially fine partitioned value space. This leads on the one hand to an increased computational amount and on the other hand doesn't reflect the actual pattern distribution. The best results were produced with bucketsizes of 3% to 5% of the data set size. Bucket sizes from 1% to 3% and from 5% to 10% produced equally good results but led to longer calculation times. Bucket sizes beyond this interval showed the above mentioned problems.

5.1.3. Dimensionality of the adjacency hyperplane

An important operation for the DFS-part of the GRIDCLUS algorithm is finding the neighbors of a block. Different types of neighbors exist depending on the dimensionality of the "touching" common hyperplane of adjacent blocks. See the following figure for the 2 dimensional block region R_{B_1}:

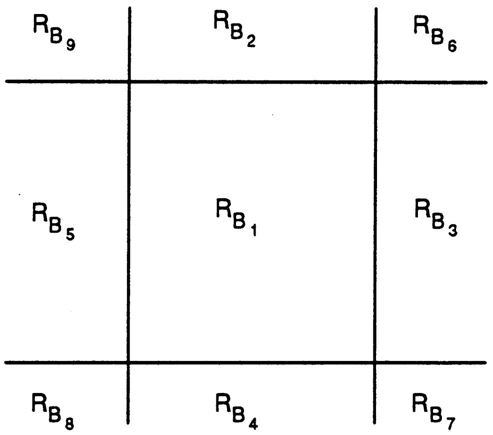


Figure 7: Possible neighbor regions

The 4 regions (R_{B_2}, R_{B_3}, R_{B_4}, R_{B_5}) have an edge (a 1-dimensional hyperplane) and the other 4 regions (R_{B_6}, R_{B_7}, R_{B_8}, R_{B_9}) a point (a 0-dimensional hyperplane) as a border to the bucket region R_{B_1}. In general, a d-dimensional region can have 3^d-1 different neighbors with d - 1 different types depending on the border dimensionality (see the appendix for a proof).

Dimensionality level L defines the neighbor blocks of a block B by the dimensionality d of the adjoining common hyperplane, with

$$D - 1 \leq d \leq D - L,$$

where $1 \leq L \leq D - 1$ and D is the dimensionality of block B.

We tried different dimensionality levels during the test suit but we recognized no apparent influences on the result. Clearly the execution time increases with higher dimensionality levels, because a larger number of possible neighbors has to be checked. Out of these results we propose a dimensionality level of 1 checking for the nearest neighbor only.

5.1.4. Testresults

The performed test suite resulted in the following recommendations for the tunable parameters of the GRIDCLUS algorithm:

- splitting strategy: Bisection,
- bucketsizes: 3% - 5% of the data set size
- nearest neighbor check (dimensionality level = 1)

5.2. Algorithm characteristics

In this section a comparison of the execution time of the GRIDCLUS algorithm to several well known clustering algorithms and a heuristic algorithm analysis is given.

5.2.1 Execution time comparison

The data sets of the test suite were analyzed using conventional methods of a commercial statistical package, the SPSS system. Due to the situation that the PC version of the SPSS package could accomplish the task for small data sets only, we processed the larger sets with the SPSS installation on a mainframe system. The pure processor time for the completion of the applied algorithm was measured and the results are given in seconds.

The used algorithms were:

hierarchical method, single linkage, SPSS-PC ("HM, PC")
partitional method, quick cluster, SPSS-PC ("PM, PC")
hierarchical method, single linkage, SPSS-mainframe ("HM, MF")
hierarchical method, single linkage, SPSS-mainframe, extrapolated ("HM, MF, ex.")
GRIDCLUS, Bisection, blocksize 5%, nearest neighbor check, PC ("GC, PC")

The PC system was a 386, 16MHz without mathematical coprocessor and the mainframe was an ES9000 running VM. The single linkage method is one of the hierarchical clustering methods provided by the SPSS system. Different other hierarchical methods were checked. The execution times for the single linkage algorithm is given only, because the other hierarchical algorithms showed a similar runtime behavior. The quick cluster algorithm is the partial clustering method provided by the SPSS system. The correct number of clusters was supplied as start values. Due to virtual memory exhaustion on the mainframe the execution times of pattern numbers greater than 2000 were extrapolated only. The scale of the y-axis is logarithmic.

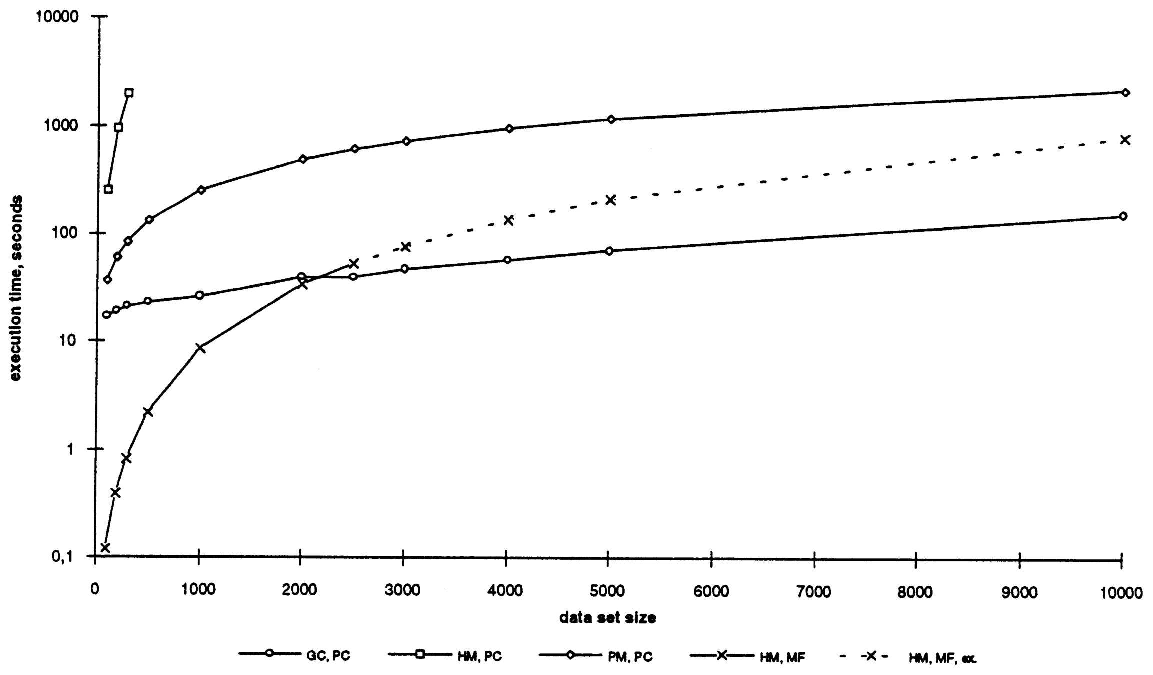


Figure 8: Comparison of the execution times

It can be seen that the GRIDCLUS algorithm outperformed all other analysis methods by far. It reached better run time behavior on a PC than the hierarchical methods of the SPSS system on the mainframe. In particular it delivered results where the other algorithms failed because of runtime or storage exhaustion.

5.2.2. Heuristic algorithm analysis

To explain the runtime behavior of the GRIDCLUS algorithm we have to examine in more detail the different tasks of the algorithm. The following figure shows the correlation of the runtime and memory characteristics of the GRIDCLUS algorithm. The scale of the y-axis is logarithmic. The scale ticks represent different units (time and size) and are therefore not directly comparable. The chart shows the influence of the different algorithm characteristics to each other and helps to understand the run time behavior. The execution time of the algorithm is separated into the time for the Grid Structure creation ("GS creation") and the neighbor search clustering ("Clustering"). The respective numbers of Grid Structure buckets ("buckets"), directory entries ("dir. entries") and clustering iterations ("iterations", while-statement (5) to (21) of the GRIDCLUS algorithm, gives the levels of the dendrogram) for the GRIDCLUS algorithm of the test run of figure 8 are shown.

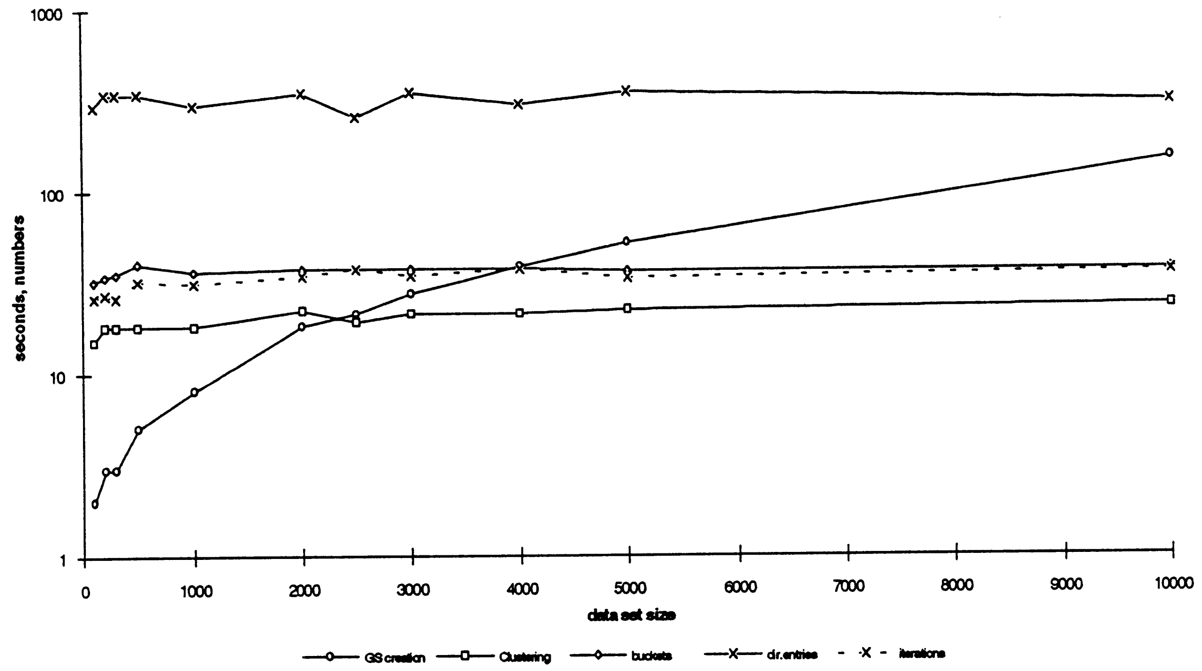


Figure 9: Correlation of runtime behavior and memory requirements

Analyzing the curves it can be seen that the only algorithm characteristic influenced by the increasing data set size is the Grid Structure creation. But this is obvious keeping in mind that the ratio of the blocksize to the data set size is constant. The number of blocks is only dependent on the blocksize ratio and not on the data set size. This leads always to the approximately same number of blocks and directory entries for a given blocksize ratio. Therefore the recursive clustering algorithm has always to deal with the same number of blocks. This explains the rather constant curves representing the block, directory entry and iteration numbers.

We can conclude that the Grid Structure creation determines the run time behavior of the GRIDCLUS algorithm.

6. Conclusion

A new hierarchical clustering method, Grid-Clustering, has been proposed. The method organizes unlike the conventional methods, the space surrounding the patterns and not the patterns . To accomplish this task it uses a multidimensional grid data structure. The resulting block partitioning of the value space is clustered via a topological neighbor search. The Grid-Clustering method proved itself as a valuable tool for analyzing the structural information of a very large data sets. It can also be used as a valuable connecting element between hierarchical and partitioning methods. It can be used for a quick scan over the data set to get structural data information and can be followed by a partitioning method. But it also proved extremely well as a stand alone clustering algorithm. One of the most appealing factors is the extremely good run time behavior. It outperforms all conventional hierarchical methods by far.

7. Acknowledgment

The author wants to thank Prof. G. Vinek for the interest in and the support of this work and Heinz Hromada for the implementation of the algorithm and the interactive cluster analysis system. Further I thank the Center for Research on Parallel Computation and the Rice University for the possibility using the university computing facilities to perform the performance tests.

This research was partially supported by the grant J0742-PHY of the Austrian FWF.

8. References

1. R. Dubes, A.K. Jain, Clustering methodologies in exploratory data analysis, *Advances in Computers* 19, Academia Press, 113-228 (1980)
2. M. Bruynooghe, A very efficient strategy for very large data sets clustering, *Proc. 9th Int. Conf. on Pattern Recognition*, 623-627 (1988)
3. D. Chaudhuri, B.B. Chaudhuri and C.A. Murthy, A new split-and-merge clustering technique, *Pattern Recognition Letters* 13, 399-409 (1992)
4. K. Chidananda Gowda and E. Diday, Symbolic clustering using a new dissimiliarity measure, *Pattern Recognition* 24, 567-578 (1991)
5. T. Kurita, An efficient agglomerative clustering algorithm using a heap, *Pattern Recognition* 24, 205-209 (1991)
6. M.A. Ismail and M.S. Kamel, Multidimensional data clustering utilizing hybrid search strategies, *Pattern Recognition* 22, 75-89 (1989)
7. N.B. Venkateswarlu and P.S.V.S.K. Raju, Fast ISODATA clustering algorithms, *Pattern Recognition* 25, 335-342 (1992)
8. Q. Zhang, Q.R. Wang and R. Boyle, A clustering algorithm for data-sets with a large number of classes, *Pattern Recognition* 24, 331-340 (1991)
9. A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, Reading, MA (1974)
10. J. Nievergelt, H. Hinterberger, K.C. Sevcik, The grid file: an adaptable, symmetric multikey file structure, *ACM Transactions on Database Systems* 9, 38-71, (1984)
11. K.H. Hinrichs, *The grid file system: implementation and case studies of applications*, Diss. ETH Nr. 7734, Zurich (1985)
12. H. Hromada, *The GRIDCLUS data analysis system*, diploma thesis Univ. Vienna, Vienna (1989)
13. M. Regnier, Analysis of grid file algorithms, *BIT* 25, 335-357 (1985)
14. A.J. Jain and R.C. Dubes, *Algorithms for clustering data*, Prentice Hall, Englewoods Cliffs, NJ (1988)

8. Appendix

Proof: 3^d neighbours

Neighbor buckets can be addressed by a adding or subtracting 1 to any number of scale indices, which defines the location in the grid directory.

Let the index of the original bucket region be $(i_1, i_2, \ldots i_d)$. Any neighbor bucket is defined by a d-tupel consisting of changing values, which can be -1 (decrement), +1(increment) and 0 (no in/decrement). The number of changing tuples can be calculated easily by the number of combinations with repetition of 3 elements (-1, +1, 0) to d classes, which is 3^d. The trivial (reflexive) change consisting of d 0's has to be substracted, so there are 3^d-1 neighbours. The number of shifts SH (+1 and -1) in a changing tupel ct gives the type of the border, which is d - SH(ct). The number of borders of type d - SH(ct) is calculated by the number of combinations of -1 and +1 on the shifts with repetition, which is 2^{sh}, times the number of placing sh shifts into d positions, which is the permutation of sh and d-sh elements on d classes without repetition. For example sh = 3 and d = 5,

$$\text{combrep}(2,3) = 2^3, \text{perm}(2,3;5) = \frac{5!}{2!*3!} \Rightarrow 8*10 = 80$$