Turkish Journal of Electrical Engineering & Computer Sciences
http://journals.tubitak.gov.tr/elektrik/
Research Article
Turk J Elec Eng & Comp Sci
(2014) 22: 143 – 154
© TÜBİTAK
doi:10.3906/elk-1202-83

M-FDBSCAN: A multicore density-based uncertain data clustering algorithm

Atakan ERDEM, Taflan İmre GÜNDEM[*]
Department of Computer Engineering, Boğaziçi University, İstanbul, Turkey

Received: 21.02.2012 • **Accepted:** 07.09.2012 • **Published Online:** 20.12.2013 • **Printed:** 20.01.2014

Abstract: In many data mining applications, we use a clustering algorithm on a large amount of uncertain data. In this paper, we adapt an uncertain data clustering algorithm called fast density-based spatial clustering of applications with noise (FDBSCAN) to multicore systems in order to have fast processing. The new algorithm, which we call multicore FDBSCAN (M-FDBSCAN), splits the data domain into c rectangular regions, where c is the number of cores in the system. The FDBSCAN algorithm is then applied to each rectangular region simultaneously. After the clustering operation is completed, semiclusters that occur during splitting are detected and merged to construct the final clusters. M-FDBSCAN is tested for correctness and performance. The experiments show that there is a significant performance increase due to M-FDBSCAN, which is not just due to multicore usage.

Key words: Data mining, uncertain data management, clustering, concurrent execution

1. Introduction

In many applications like data cleaning, data integration, and sensor networks, managing huge amounts of uncertain data is one of the most crucial issues. Recently, many new algorithms dealing efficiently with uncertainty have been developed. For example, in [1], the customer preference uncertainty problem was examined. The preferences of customers that are unfamiliar with a new technology are assumed to be uncertain. The proposed solution was based on a fuzzy ranking approach. In [2], a novel accuracy function was proposed as a solution to the problem of ranking intuitionistic fuzzy numbers accurately. Most of the solutions to the uncertain data problems that we see in the literature are essentially derived from the solutions to the certain data counterparts of these problems, as in [3] and [4]. The most critical parameter in the performance of uncertain data management algorithms is the size of the uncertain data. Thus, the issue of processing huge amounts of uncertain data efficiently is of utmost importance.

In this paper, we adopt fast density-based spatial clustering of applications with noise (FDBSCAN) [3], a density-based uncertain data clustering algorithm, to multicore systems. We name our new algorithm multicore FDBSCAN (M-FDBSCAN). The main idea behind the M-FDBSCAN algorithm is to split the 2-dimensional dataset into $c$subdatasets, where c is the number of cores of the multicore system, and then apply a merge operation for the split subdataset pairs sequentially to get the final clusters, which are the same as those that are discovered by the single-core algorithm.

Our proposed algorithm is one of the first on adapting data mining algorithms for multicore system architecture. In [5], a k-means clustering algorithm for multicore system architecture was presented, but the algorithm was based on certain data.

[*]Correspondence: gundem@boun.edu.tr

In Section 2, the related research is summarized. In Section 3, the M-FDBSCAN algorithm is explained. Finally, in Section 4, the performance evaluation of the proposed algorithm is presented.

2. Related work

In [5], one of the most well-known clustering algorithms, the k-means algorithm, was reimplemented for multicore system architecture. To each core, some part of the dataset was assigned for clustering. After the k-means clustering algorithm was applied to the relevant part of the dataset by each dedicated core, simultaneously, a merge operation was run to get the final clusters. In [5], the dataset was composed of certain data.

Uncertain data versions of the k-means, DBSCAN, and OPTICS clustering algorithms were proposed in [3], [4], and [6], respectively. Based on the fuzzy c-means algorithm, novel fuzzy clustering solutions for different data domains and problems were proposed in [7], [8], and [9]. Note that the algorithms in [3,4] and [6–9] were for single-core system architecture use.

In this paper, we adapt the FDBSCAN algorithm for multicore system architecture. FDBSCAN is a density-based fuzzy clustering algorithm. Since the concerned data of the algorithm are uncertain, a fuzzy distance function is used to evaluate the similarity of the fuzzy objects. The key idea behind the FDBSCAN algorithm is summarized in the following:

Let us assume that fuzzy data objects are represented in a 2-dimensional space. To label a fuzzy data object region in 2-dimensional space as a cluster, the following 2 conditions must be satisfied: 1) All of the pairs of fuzzy data objects in the region must be density-reachable. 2) The total number of fuzzy data objects in the region must be equal to or greater than a certain value, μ. A pair of data objects is density-reachable if there is at least one path between them, such that the fuzzy distance between each of the adjacent nodes is in the ε neighborhood, where ε is a certain given value. Two data objects are in the ε neighborhood if the distance between these data objects is less than or equal to ε. Each fuzzy data object is represented by s sample data objects. Probabilistically, the fuzzy distance between 2 fuzzy data objects, fo_1 and fo_2, is equal to d if more than half of all of the possible distances among the sample data objects associated with fo_1 and fo_2 are equal to d.

3. Proposed algorithm

One of the most time-consuming, but at the same time important, parts of the M-FDBSCAN algorithm is the sample matrix construction part. Most of the algorithm is based on manipulating the sample matrices. Thus, the sample matrix computation should be efficient. During the execution of the FDBSCAN algorithm, one sample matrix is constructed for each fuzzy data object in the dataset. A sample matrix of a fuzzy data object, fdo_x, represented by $\mathrm{SM}(\mathrm{fdo}_x)$, is an s $\times$ s matrix, such that s is the number of sample data objects. fdo_x[i] is the ith sample data object of the fuzzy data object fdo_x, where i = 1,..,s. Let $\mathrm{SM}(\mathrm{fdo}_x)$[i,j] be the matrix element of $\mathrm{SM}(\mathrm{fdo}_x)$. The value of $\mathrm{SM}(\mathrm{fdo}_x)$[i,j] then represents the number of fuzzy data objects that are different from fdo_x and of which the jth sample data objects are in the ε neighborhood of the ith sample data object of fdo_x.

A sample matrix example with 4 sample data objects is shown in Figure 1.

For a dataset of f fuzzy data objects and s sample data objects per each fuzzy data object, there are $s^2 \times (f + (f-1) + (f-2) \ldots + 1) = s^2 \times (f \times (f+1)/2) = s^2 \times (f^2 + f)/2$ distance calculations for the sample matrix elements. Please note that the result is not simply $s^2\times\ f^2$. This is because the distances between the fuzzy data object pairs (fdo_x, fdo_y) and (fdo_y, fdo_x) are obviously the same. Thus, calculating the distances between 2 fuzzy data objects just once is sufficient.

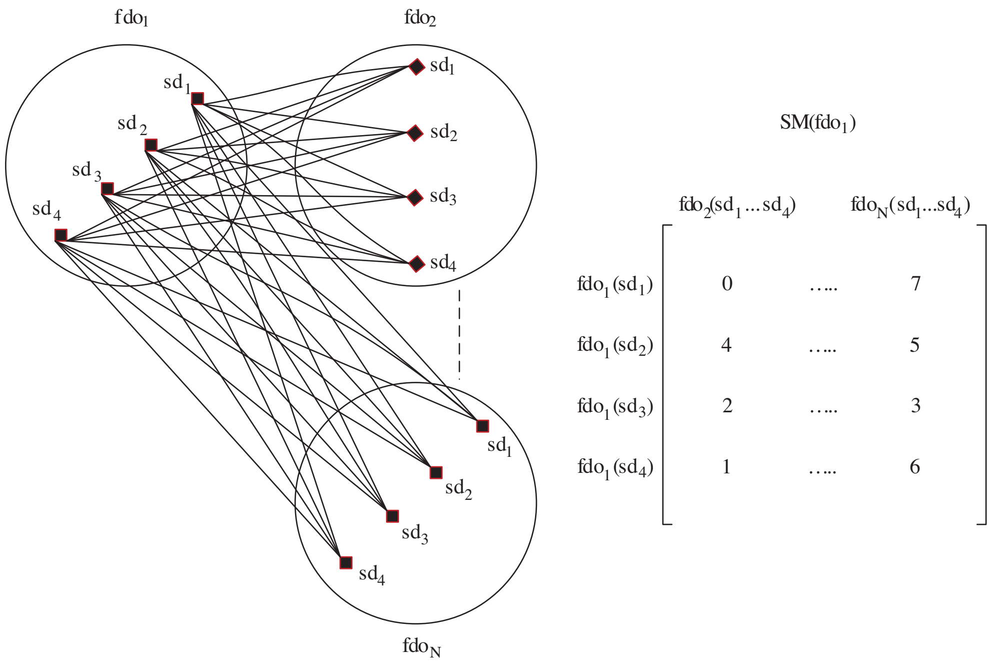


Figure 1. Sample matrix with 4 sample data objects for each fuzzy data object.

In our proposed algorithm, we aim to parallelize these computationally time-consuming distance calculation operations by splitting the dataset into c subdatasets of equal or nearly equal size. Here, c is the number of cores in the associated multicore system. Thus, assuming that the sizes of all of the subdatasets are equal, $s^2 \times (f^2/c^2 + f/c)/2$ distance calculations are done by each core. The total distance calculation of all of the cores is $c \times s^2 \times (f^2/c^2 + f/c)/2$.

For c $>$ 1,

$$s^2 \times (f^2 + f)/2 > c \times s^2 \times (f^2/c^2 + f/c)/2 \rightarrow f^2 > f^2/c. \tag{1}$$

In Eq. (1), it is interestingly shown that dataset splitting reduces the number of distance calculations. Thus, even for sequential execution, the execution time performance of our proposed algorithm is better than that of FDBSCAN.

After processing all of the subdatasets, sequential merge operations are needed for each subdataset pair to establish the final clusters. Thus, deciding the number of subdatasets is important. An increase in the number of subdatasets implies an increase in the number of sequential merge operations. A detailed description of the proposed algorithm is given in Section 3.1.

3.1. M-FDBSCAN

The main idea behind the algorithm is to split the 2-dimensional fuzzy data object dataset into c subdatasets horizontally or vertically, where c is the number of cores in the multicore system. By applying the FDBSCAN algorithm to each subdataset concurrently, the final cluster regions are determined partially. After merging the split subdataset pairs by our merge function, the final cluster regions are obtained. Parallelization is done using OpenMP in C. The M-FDBSCAN algorithm is basically composed of 3 parts:

- Splitting the dataset into subdatasets.
- Applying FDBSCAN to each split subdataset concurrently.
- Merging subdataset pairs to get the final cluster regions.

3.1.1. Splitting the dataset into subdatasets

We gained significant performance improvement by splitting the original dataset into subdatasets. This is because smaller subdatasets mean smaller sample matrices and smaller sample matrices mean fewer distance calculations. However, since the bigger the number of subdatasets is, the bigger the number of operations is, there is a threshold point at which we have the optimum number of partitioning (splitting). Splitting must be done up to this number. The execution times of the splitting and merging processes for a different number of partitions are given in the experimental results in Section 4.

According to the M-FDBSCAN algorithm, splitting is done with respect to some horizontal or vertical lines. Subdatasets are formed according to the regions between these lines. In the experiments, we tested 2 different splitting approaches:

a) Special splitting algorithm: For splitting, we developed a special algorithm that is derived from a binary search algorithm. The goal of this algorithm is to determine $n-1$ lines that split the dataset into n (nearly) equal-sized subdatasets. The execution time of the algorithm changes according to the distribution of the data in 2-dimensional space. That is, the data distribution determines the number of iterations needed for finding the appropriate splitting lines.

b) n equal-distance lines for $n-1$ subdatasets: The other approach is simply determining n equal-distance horizontal or vertical lines for $n-1$ subdatasets.

In the experiments, we observed the execution times of both splitting approaches. Even if the execution time of the second splitting approach is almost 0, when the data are not well distributed in the 2-dimensional space, the big differences in the size of the subdatasets cause long total execution times. On the other hand, for the first splitting approach, the sizes of the generated subdatasets are nearly the same. Thus, the overall execution time performance is better, even if a negligible amount of time is needed initially for the splitting process.

3.1.2. Applying FDBSCAN to each split subdataset concurrently

M-FDBSCAN assigns each subdataset to a core for clustering. After running the FDBSCAN algorithm by the cores concurrently, temporary cluster regions are established. We must note that if the number of cores in the multicore system is less than the optimum number of subdatasets, then the subdatasets must be apportioned among the cores (i.e. each core processes several subdatasets serially). As we showed earlier, splitting improves the overall execution time performance, even for a single-core system.

3.1.3. Merging subdataset pairs to get the final cluster regions

In the last part of the M-FDBSCAN algorithm, the processed subdataset pairs are merged in order to get the final cluster regions. The proposed merge operation is mainly set on the concept of the ε neighborhood of the splitting line. A fuzzy data object that is in the ε neighborhood of any point on the splitting line is called the insider *fdo* and, conversely, a fuzzy data object that is not in the ε neighborhood of any point on the splitting line is called the outsider *fdo*.

During the merge operation, first, the cluster regions of the insider *fdo*s are handled. Cluster region decisions for the outsider *fdo*s are made according to the insider *fdo*s that are density-reachable to the outsider *fdo*s. If a fuzzy data object fdo_i is an outsider fdo, then there are 2 possible situations:

1. fdo$_i$ is a noisy fuzzy data object. This means that it could not be assigned to any temporary cluster. In this case, if fdo$_i$ is density-reachable to any insider fdo that is in a cluster region C$_x$, then fdo$_i$ and all of the other density-reachable outsider *fdo*s are also put into the cluster region C$_x$.

2. fdo$_i$ is already in a temporary cluster region C$_y$. In this case, if fdo$_i$ is density-reachable to any insider *fdo* that is assigned to a cluster region C$_x$, then the label of the cluster region C$_y$ is changed to C$_x$. This means that the cluster regions C$_y$ and C$_x$ are merged and the new cluster region label is set as C$_x$.

The merge operation is done between 2 adjacent subdatasets. Temporary cluster structures in other subdatasets are not affected during the merge operation between 2 specific subdatasets. Thus, we can state that one application of the merge operation is independent of another of its applications. Consequently, several merge operation applications may be processed in parallel. A merge operation can be processed on any one of the c cores that the FDBSCAN algorithm is run on for obtaining the neighboring clusters.

The complexity of parallel merging is O(log$_2$c). This is because, at each iteration, each core is assigned to merge 2 subdatasets.

3.2. Algorithm for M-FDBSCAN

Input:

D: Dataset of 2-dimensional fuzzy data objects.

ε : Minimum distance between any pair of data objects in a cluster.

μ : Minimum number of fuzzy data objects in a cluster.

c: Number of cores.

D[i]: i*th* subdataset.

Output:

CS: Set of clusters of fuzzy data objects.

NS: Set of noisy fuzzy data objects.

```
M-FDBSCAN(D , ε, μ, c)
BEGIN
1 Split_Dataset(c) ;
/* Beginning of the parallel section */
2       For i : 1 to c Do
3               FDBSCAN(D[i] , ε, μ ) ;
4       End For ;
/* End of the parallel section */

// k : index of core starting from 1
5 mergedDatasets ← D[1]
6 for  j = 1 to log2c do
        /* Beginning of the parallel section */
7       for  k = 1 to c do
8               if (  k mod 2^j  ) = 0, then
9                       Merge_Datasets( D[k], D[k-2^(j-1)],ε,μ )
10              end if
11      end for
        /* End of the parallel section */
12 end for
13 mergedDatasets ← D[c]
END;
```

```
Split_Dataset(c)
BEGIN
/* Split_Dataset is an implementation of a binary search like splitting algorithm */
1       Subdataset_Size : = Size(D)/c ;  /* Approximate size of each dataset */
2       m[1]: = min_y ; /* min_y is the minimum y value in the original dataset D */
3       for i in 2..c loop
4               min_temp : = m[i-1] ;
                /* max_y is the maximum y value in the original dataset D */
5               max_temp : = max_y ;
6               Temp_Size : = 0 ;
7               While (Temp_Size < (Subdataset_Size – delta) or Temp_Size > (Subdataset_Size + delta))
                and   (min_temp ≤ max_temp) loop
                /* delta is an approximation parameter for decreasing the number of search steps */
8                       mid : = min_temp + (max_temp – min_temp) / 2;
9                       $y_1$ : = m[i – 1]  ;
10                      $y_2$ : = mid ;
11                      Temp_Size : = Number of fuzzy data objects in the region that is between $y_1$ and
                        y2 lines
12                      if Temp_Size < Subdataset_Size, then
13                              min_temp : = mid + 1 ;
14                      else
15                              max_temp : = mid - 1;
16                      end if ;
17              end loop ;
18              D[i-1] ← The region between $y_1$ and $y_2$ lines
19              m[i]: = mid ;
20      end loop;
21      $y_1$ : = mid ;
22      $y_2$ : = max_y ;
23      D[c] ← The region between $y_1$ and $y_2$ lines
END;
```

In Figure 2, the original dataset D is split into 2 subdatasets, D1 and D2, by a splitting line.

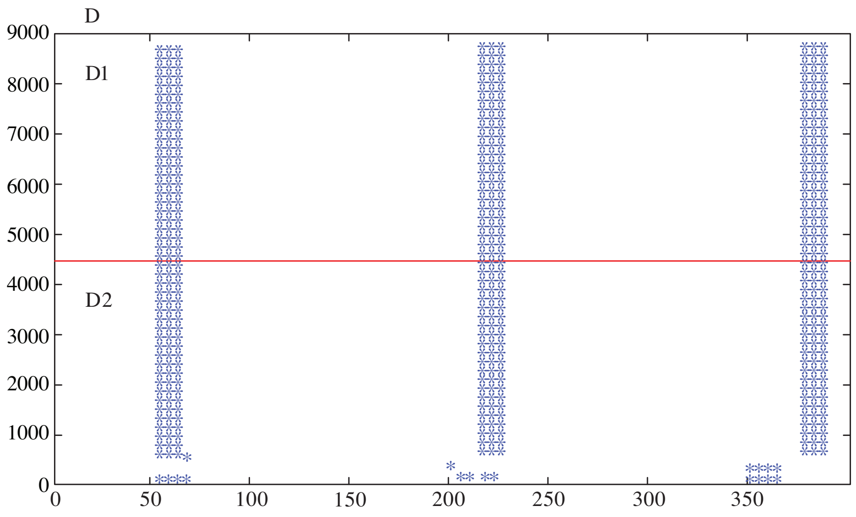


Figure 2. Splitting line and the D1 and D2 subdatasets.

```
Merge_DataSets(D1, D2, ε, μ )
/* Merges 2 datasets by means of cluster regions and noisy fuzzy data objects. C(f_1) is the label of the cluster
region of fuzzy data object f1 in D1, C(f_2) is the label of the cluster region of fuzzy data object f_2 in D2. If
C(f_x) is a null value, then this means that fx is a noisy data object. */
BEGIN
1    Select fuzzy data objects into R1 from D1 where $y \geq max(y) - \varepsilon$
2    Select fuzzy data objects into R2 from D2 where $y \leq min(y) + \varepsilon$
3    For each nonvisited fuzzy data object f1 in R1 Do
4        For each nonvisited fuzzy data object f2 in R2 Do
5            If fuzzy_distance(f1,f2) $\leq \varepsilon$ then
6                If C(f1) is not null and C(f2) is not null then
7                    Change the cluster label of all fuzzy data objects of which the
                     cluster label is C(f1) in D1 as C(f2)
8                Else if C(f1) is null and C(f2) is not null then
9                    Set the cluster label of f1 as C(f2)
10                   Set also the cluster label of all noisy fuzzy data objects that are
                     density-reachable from f1 as C(f2).
11               Else if C(f1) is not null and C(f2) is null then
12                   Set the cluster label of f2 as C(f1)
13                   Set also the cluster label of all noisy fuzzy data
                 objects that are density-reachable from f2 as C(f1)
14               Else if C(f1) is null and C(f2) is null then
15                   If ($\Sigma$(density reachable data objects from f1) + 1 + $\Sigma$(density
                     reachable data objects from f2) + 1) $\geq \mu$ then
16                       Create a new cluster label $C_{new}$ and set the cluster
                         label of f1, f2, all fuzzy data objects that are reachable
                         from f1, and all fuzzy data objects that are reachable
                         from f2 as $C_{new}$
17               End If;
18               End If;
19           End If;
20       End For;
21   End For;
END;
```

In Figure 3, all of the possible merging situations are shown. In Figure 3a, the fuzzy data objects between the splitting line and the lines that are ε-far from the splitting line are assigned to a certain temporary cluster. In Figures 3b and 3c, the fuzzy data objects on one side of the splitting line are assigned to a certain temporary cluster, while on the other side, all of them are noisy. In Figure 3d, all of the fuzzy data objects between the splitting line and the lines that are ε-far from the splitting line are noisy, which means that they cannot be assigned to any cluster.

4. Experimental study

In our experiments, we use synthetic datasets of sizes 1000, 5000, 10,000, 25,000, and 50,000. These datasets include 2-dimensional fuzzy data objects. Each fuzzy data object is represented by 7 sample data objects. The sample data objects for each fuzzy data object are produced by choosing a core data object in 2-dimensional space and selecting 7 data objects in 1 neighborhood of this core data object. Each data object is generated in the range of 0 to 10,000. The tests are done on an Intel Xeon X5650 2.67 GHz 24 Core CPU and 72 GB

RAM server. The operating system of the computer is a 64-bit Linux Centos 5.5. The performances of the FDBSCAN and M-FDBSCAN algorithms are tested with predefined ε and μ values of 120 and 7, respectively.

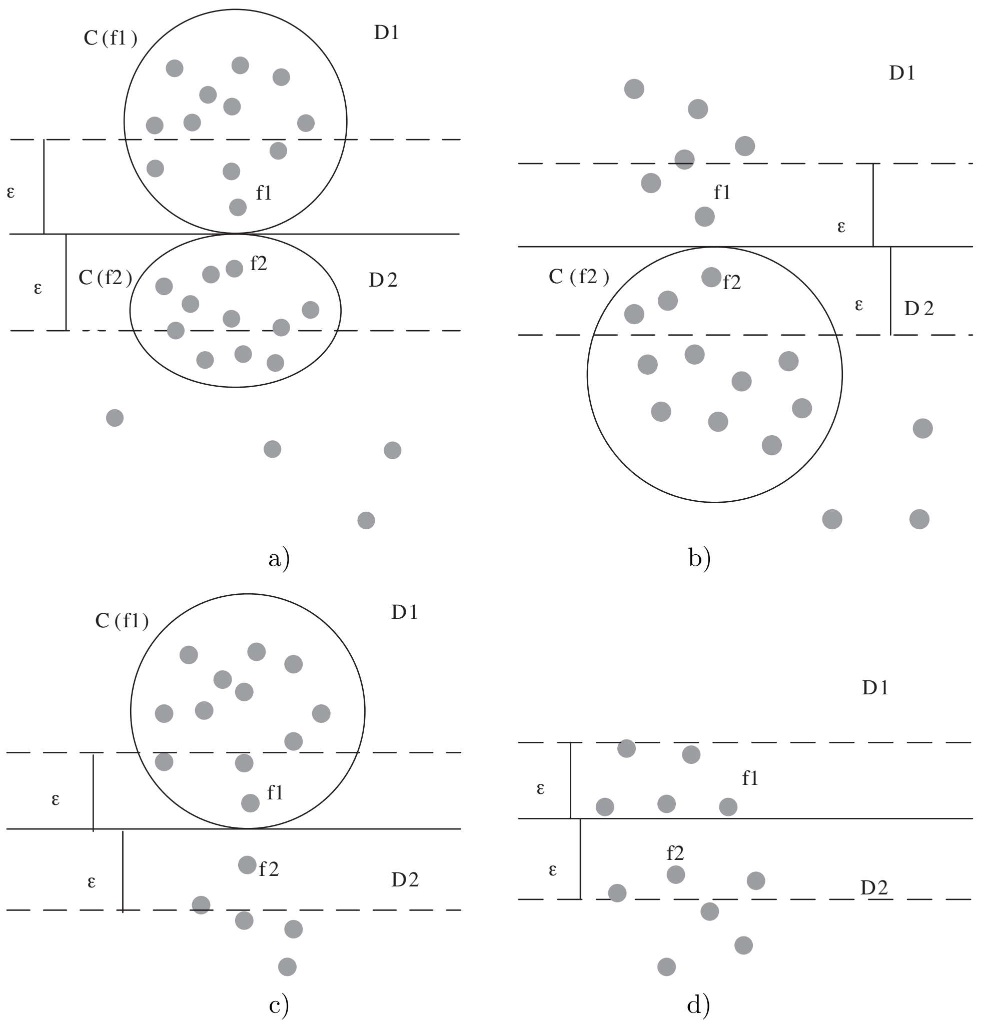


Figure 3. a) f1 and f2 belong to the C(f1) and C(f2) clusters. b) f1 is noisy and f2 belongs to the C(f2) cluster. c) f1 belongs to the C(f1) cluster and f2 is noisy. d) f1 and f2 are noisy.

The results of the tests show that by splitting the original dataset and assigning each subdataset to a certain core, the M-FDBSCAN algorithm completes the overall clustering processes, approximately, and the splitting and core times are faster than that of the FDBSCAN algorithm. Even for a single core, by only splitting the dataset, the execution time performance increases linearly. During our tests, we split the original dataset into 2, 4, 8, 16, and 24 subdatasets.

In the Table, the detailed observation results are listed.

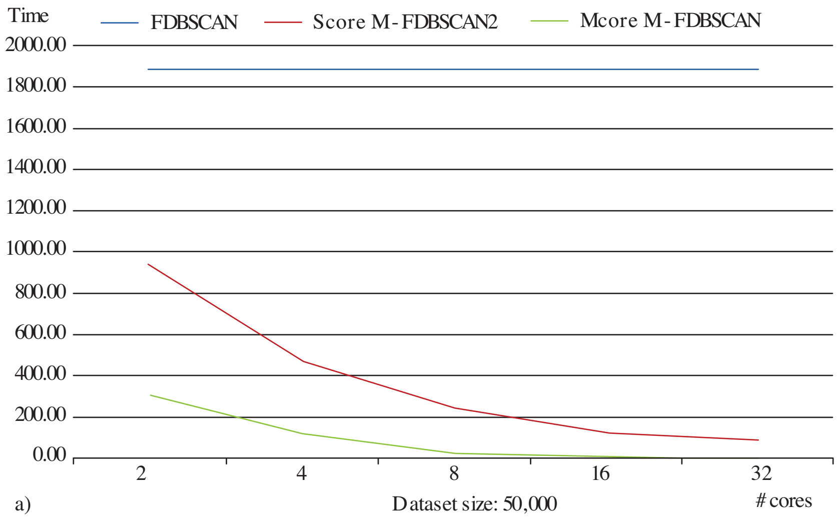
As seen in the Table, the total execution time decreases, while the splitting number increases. Determining the optimum number of splitting and running M-FDBSCAN only once according to this optimum value is the subject of our future research.

Table. Observation results.

Cores	Splits	Dataset size	M-FDBSCAN			FDBSCAN (s)
			Merge (s)	With a split algorithm (s)	Without a split algorithm (s)	
1	2	1000	0.000125	0.70327	0.703717	1.380909
1	2	5000	0.002682	10.856971	10.859653	20.127721
1	2	10,000	0.01233	38.825364	38.831015	78.16387
1	2	25,000	0.071175	235.816623	235.828666	474.169815
1	2	50,000	0.258704	938.837119	938.87024	1884.31
1	4	1000	0.000699	0.370862	0.371315	1.380909
1	4	5000	0.008624	6.079342	6.081973	20.127721
1	4	10,000	0.036881	19.241537	19.245336	78.16387
1	4	25,000	0.21746	119.039382	119.055095	474.169815
1	4	50,000	0.774352	470.46193	470.495162	1884.31
1	8	1000	0.001681	0.184651	0.185121	1.380909
1	8	5000	0.020122	3.478109	3.480742	20.127721
1	8	10,000	0.07075	11.067125	11.072784	78.16387
1	8	25,000	0.459046	59.873915	59.889656	474.169815
1	8	50,000	1.869456	239.86957	239.902948	1884.31
1	16	1000	0.004037	0.098348	0.09879	1.380909
1	16	5000	0.063088	2.202966	2.205592	20.127721
1	16	10,000	0.167074	6.173402	6.17902	78.16387
1	16	25,000	1.007704	31.07222	31.082821	474.169815
1	16	50,000	4.032029	124.082786	124.115831	1884.31
1	24	1000	0.005828	0.069812	0.070264	1.380909
1	24	5000	0.1243	1.633174	1.635821	20.127721
1	24	10,000	0.254816	4.285194	4.290068	78.16387
1	24	25,000	1.583718	22.502006	22.517911	474.169815
1	24	50,000	6.228687	87.01267	87.046208	1884.31
2	2	1000	0.000133	0.376023	0.376483	1.380909
2	2	5000	0.0029	6.146009	6.148674	20.127721
2	2	10,000	0.013296	20.165065	20.170722	78.16387
2	2	25,000	0.066694	127.959727	127.970343	474.169815
2	2	50,000	0.098917	305.322412	305.342816	1884.31
4	4	1000	0.000304	0.095415	0.095877	1.380909
4	4	5000	0.004766	2.132391	2.135046	20.127721
4	4	10,000	0.014265	5.89921	5.904839	78.16387
4	4	25,000	0.084978	30.932535	30.946148	474.169815
4	4	50,000	0.263683	121.593786	121.62266	1884.31
8	8	1000	0.00039	0.024685	0.025138	1.380909
8	8	5000	0.005349	0.582897	0.585523	20.127721
8	8	10,000	0.016793	2.127637	2.133287	78.16387
8	8	25,000	0.077743	8.781387	8.797253	474.169815
8	8	50,000	0.299754	32.13779	32.162884	1884.31
16	16	1000	0.0007	0.014105	0.01457	1.380909
16	16	5000	0.01003	0.260199	0.262835	20.127721
16	16	10,000	0.03966	1.090425	1.096262	78.16387
16	16	25,000	0.149994	4.128454	4.144334	474.169815
16	16	50,000	0.577006	14.324988	14.358222	1884.31
24	24	1000	0.028725	0.065827	0.066277	1.380909

Table. Continued.

Cores	Splits	Dataset size	M-FDBSCAN			FDBSCAN (s)
			Merge (s)	With a split algorithm (s)	Without a split algorithm (s)	
24	24	5000	0.025327	0.257525	0.260255	20.127721
24	24	10,000	0.074669	0.665056	0.670703	78.16387
24	24	25,000	0.17598	2.801175	2.816897	474.169815
24	24	50,000	0.715664	8.650306	8.675448	1884.31
24	24	1000	0.028725	0.065827	0.066277	1.380909
24	24	5000	0.025327	0.257525	0.260255	20.127721
24	24	10,000	0.074669	0.665056	0.670703	78.16387
24	24	25,000	0.17598	2.801175	2.816897	474.169815
24	24	50,000	0.715664	8.650306	8.675448	1884.31



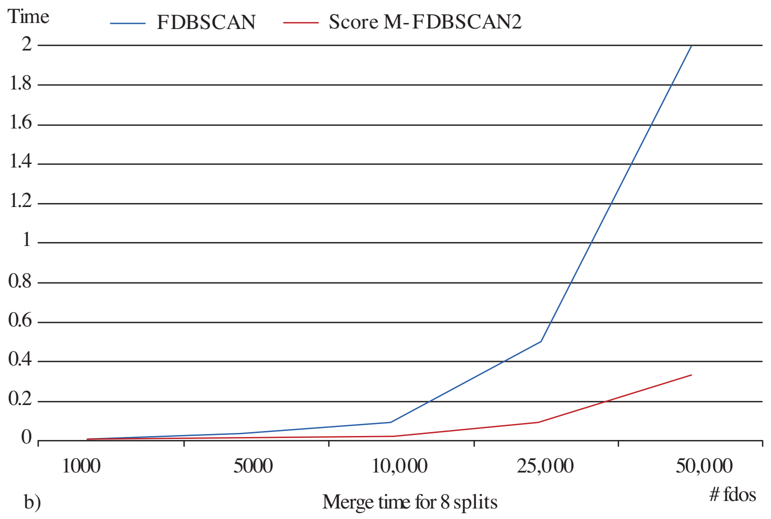


Figure 4. a) For 50,000 *fdo* records, the execution times of FDBSCAN, M-FDBSCAN for single-core, and M-FDBSCAN for multicore systems. b) For 8 splits, the merging times needed for M-FDBSCAN for single-core and M-FDBSCAN for multicore systems.

We tested 2 splitting methods:

- Splitting with an algorithm.
- Splitting without an algorithm.

We recorded a negligible amount of performance improvement when the splitting was done with an algorithm.

In Figure 4a, the single-core and multicore executions of M-FDBSCAN are compared with those of FDBSCAN, which uses only a single core by the nature of the algorithm. In Figure 4b, the merging times of the M-FDBSCAN algorithm run on single-core and multicore systems are given. In Figure 4a, the single-core line represents the performance of sequential merging, while the multicore line represents the performance of parallel merging.

Comments on the observation results:

1. During the M-FDBSCAN performance tests, we observed that performance improvements could be achieved linearly by increasing the number of cores and splits.
2. In the case of single-core usage, by just splitting the dataset, we can say that M-FDBSCAN is splitting a number of times faster than FDBSCAN. Our splitting approach converges the single-core performance to a multicore performance.
3. While the number of splits increases, the merge operations become critical for the execution time. With the help of our parallel merge algorithm, the execution time of the merge operations is considerably reduced.
4. With over 10,000 fuzzy data objects, the total clustering time increases dramatically. We observe that the performance gap between M-FDBSCAN and FDBSCAN increases quickly when the dataset size goes over 10,000 fuzzy data objects. Thus, for huge amounts of data, the use of M-FDBSCAN becomes more meaningful.

5. Future work

We have observed that, while the size of the dataset increases, splitting becomes critical. Thus, more intelligent splitting algorithms are needed that learn the characteristics of the data distribution and suggest the best splitting coordinates. Another critical point is determining the optimum splitting number. Learning the data distribution characteristics of a dataset before the clustering process is also very important to determine the optimum splitting number. Thus, for huge datasets, these 2 issues must be examined further in a future research.

6. Conclusion

Because of fuzziness, the sizes of the datasets that store uncertain data are very huge. Thus, such datasets should be processed by multicore systems. In this work, we aimed to improve the performance of FDBSCAN, which is an uncertain data clustering algorithm, by customizing it for multicore system architecture. We named this new algorithm M-FDBSCAN. The main idea behind the M-FDBSCAN algorithm is to split the dataset into a certain number of subdatasets and assign these subdatasets to the cores. At each core, the FDBSCAN algorithm is run concurrently. After that, the subclusters provided by each core are merged by satisfying the predefined constraints, such as the minimum number of fuzzy objects in a cluster and the maximum distance between 2 fuzzy objects in a cluster. As a result, even though M-FDBSCAN is a multicore clustering algorithm, it also proposes a dramatic performance improvement for single-core systems.

References

[1] K. Lin, L. Shih, Y. Cheng, S. Lee, "Fuzzy product line design model while considering preference uncertainty: a case study of notebook computer industry in Taiwan", Expert Systems with Applications, Vol. 38, pp. 1789–1797, 2011.

[2] V. Lakshmana, G. Nayagam, S. Muralikrishnan, G. Sivaraman, "Multi-criteria decision-making method based on interval-valued intuitionistic fuzzy sets", Expert Systems with Applications, Vol. 38, pp. 1464–1467, 2011.

[3] H.P. Kriegel, M. Pfeifle, "Density-based clustering of uncertain data", Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, 2005.

[4] H.P. Kriegel, M. Pfeifle, "Hierarchical density based clustering of uncertain data", Proceedings of the 5th IEEE International Conference on Data Mining, 2005.

[5] S.N. Rao, E.V. Prasad, N.B. Venkateswarlu, "A critical performance study of memory mapping on multi core processors: an experiment with k-means algorithm with large data mining data sets", International Journal of Computer Applications, Vol. 1, 2010.

[6] W. Ngai, B. Kao, C. Chui, R. Cheng, M. Chau, K.Y. Yip, "Efficient clustering of uncertain data", Proceedings of the 6th IEEE International Conference on Data Mining, 2006.

[7] S.P. Chatzis, "A fuzzy c-means-type algorithm for clustering of data with mixed numeric and categorical attributes employing a probabilistic dissimilarity functional", Expert Systems with Applications, Vol. 38, pp. 8684–8689, 2011.

[8] H. Izakian, A. Abraham, "Fuzzy C-means and fuzzy swarm for fuzzy clustering problem", Expert Systems with Applications, Vol. 38, pp. 1835–1838, 2011.

[9] D. Li, H. Gu, L. Zhang, "A fuzzy c-means clustering algorithm based on nearest-neighbor intervals for incomplete data", Expert Systems with Applications, Vol. 37, pp. 6942–6947, 2010.