Indonesian Journal of Electrical Engineering and Informatics (IJEEI)
Vol. 11, No. 3, September 2023, pp. 815~833
ISSN: 2089-3272, DOI: 10.52549/ijeei.v11i3.4786

ACRMiner: An Incremental Approach for Finding Dense and Sparse Rectangular Regions from a 2D Interval Dataset

Dwipen Laskar[1], AnjanaKakoti Mahanta[2]
[1,2]Department of Computer Science, Gauhati University
Jalukbari, Guwahati, Assam, India

Article Info

Article history:

Received May 11, 2023
Revised Sep 8, 2023
Accepted Sep 22, 2023

Keyword:

Dense Regions
Sparse Regions
Closed Rectangles
Interval Data Mining
Support Counts

ABSTRACT

In many applications, transactions are associated with intervalsrelated to time, temperature, humidity, or other similar measures. The term "2D interval data" or "rectangle data" is used when there are two connected intervals with each transaction. Two connected intervals give rise to a rectangle. The rectangles may overlap producing regions with different density values. The density value or support of a region is the number of rectangles that contain it. A region is closed if its density is strictly bigger than any region properly containing it. For rectangle dataset, these regions are rectangular in shape. In this paper an algorithm named *ACRMiner*has been proposed that takes as input a sequence of rectangles and computes all closed overlapping rectangles and their density values. The algorithm is incremental and thusis suitable for dynamic environment. Depending on an input threshold the regions can be classified as dense and sparse. Here a tree-based data structure named as *ACR-Tree*is used. The method has been implemented and tested on synthetic and real-life datasets and results have been reported. Fewapplications of this algorithm have been discussed. The worst-case time complexity the algorithmis $O(n^5)$ where *n* is the number of input rectangles.

Corresponding Author:

DwipenLaskar,
Department of Computer Science,
Gauhati University,
Jalukbari, Guwahati, Assam, India
Email: laskardwipen@gauhati.ac.in

1. INTRODUCTION

Data mining is the process of extracting and discovering unknown, hidden patterns from data[1][32]. Various data mining techniques include clustering, classificarion and association rule mining etc. In data mining, transactions may be event related data in real world and are associated with intervals in both continuous and discrete domains such as intervals of distance, time, blood pressure, etc[2][3]. An interval has start and end values associated with it[4][5][6][7][8]. Interval data mining is a data mining approach that extracts hidden information, patterns, and association rules [27] from interval data sets. In an interval dataset there may be many intervals which overlap. This overlapping interval information help users to group transactions based on a certain similarity measure. The total number of overlapping intervals is called the support of the overlapped region of these intervals. The idea of closed interval [9] is an extension of the concept of closed item-set [10]. The support of a closed interval is strictly more than the support of any interval properly containing it. For an interval dataset the closed intervals are actually the non-empty intersections of intervals in the dataset [11]. This is because if the intersection of two (or more) intervals is extended in either direction then its support decreases. Hence the support of any interval properly containing it will be less than the support of it. Let us consider a data set with two intervals I_1=[4, 7] andI_2=[5, 8] as shown in Figure 1. The overlapping (intersection) of these two intervals is the interval I_3=[5, 7]. The interval

I_3 has support value 2 as it is contained in both the intervalsI_1 and I_2. The interval I_3=[5, 7] is closed because if we extend it in any direction one of the intervals I_1 and I_2 will not contain it.

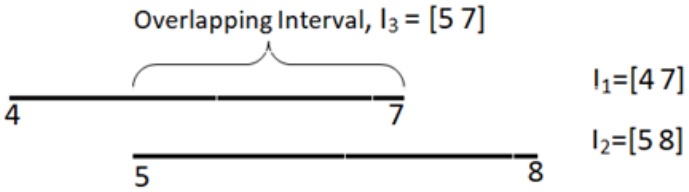

Figure 1. Intervals I_1=[4, 7] and I_2=[5 , 8] with overlapping interval I_3=[5, 7] with support 2

Numerous techniques have already been proposed for one dimensional (1D) interval data set, including mining closed intervals [6][8][9][12][13][14], closed frequent intervals [6][8][9][11][12][15], maximal frequent intervals [6][9][15][16] and Minimal Infrequent Intervals[17]. However, there are many real-world problems in which objects are associated with two intervals and can be represented as rectangles in 2D space as shown in Figure 2.

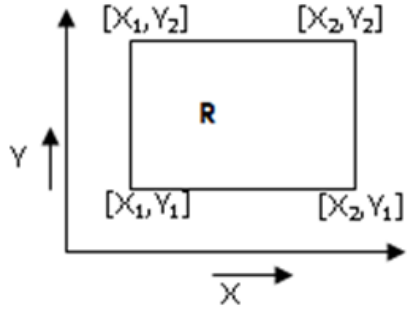

Figure 2. Rectangle R with intervals I_x=[X_1,X_2]and I_y=[Y_1,Y_2]

Here I_x=[X_1,X_2] andI_y=[Y_1,Y_2] are intervals in X-domain and Y-domain respectively, and R is the corresponding rectangle. The rectangle R is generally represented as [X_1, X_2, Y_1, Y_2]. Any point P(x,y) in this rectancle will have $x \in I_x$ and $y \in I_y$. A set of such rectangles is called rectangle data set [6][15]. Finding dense and sparse region from these rectangle datasets is also important for the users. For example, in food storage control system, temperature and humidity are two important environmental factors to preserve the food products. The system can record the temperature interval i.e. minimum and maximum temperature and also the humidity range i.e. the minimum and maximum humidity level for each of the food products that are required to preserve them efficiently. Both the environmental factors temperature and humidity can be visualized as a rectangle in two dimensional spaces as show in Figure 3, for the set of rectangle data given in Table 1.

Table 1. Temperature and Humidity of fruits

Fruit name	Temperature (^{0}F) [max, min]	Humidity (%) [max, min]
Papaya	[10, 17]	[40, 75]
Orange	[14, 19]	[35, 45]
Pineapple	[12, 21]	[35, 65]
Mango	[16, 20]	[60, 85]

Now, if we extract closed rectangles from this set of rectangles that is represented by temperatures and humidity ranges, then it is possible to find out the dominant temperature and humidity intervals that can be applied to preserve the maximum or desired number of food products. Thus, these dominant rectangular areas are dense regions with different density values. In Figure 3, there are 4 fruits Papaya, Orange,

Pineapple and Mango and their respective temperature and humidity factors are as shown in Table 1. The area filled with Red and Green are dense regions with support value 3 because they are the result of overlapping of three rectangles given in Table 1. There may be multiple number of dense regions for a given minimum threshold value of density. The red coloured area represents the rectangle R_1=[16,17,60,65] which is the intersection of 3 rectangles and so its support value is 3. Similarly R_2=[14,17,40,55] is also a rectangle with support 3.

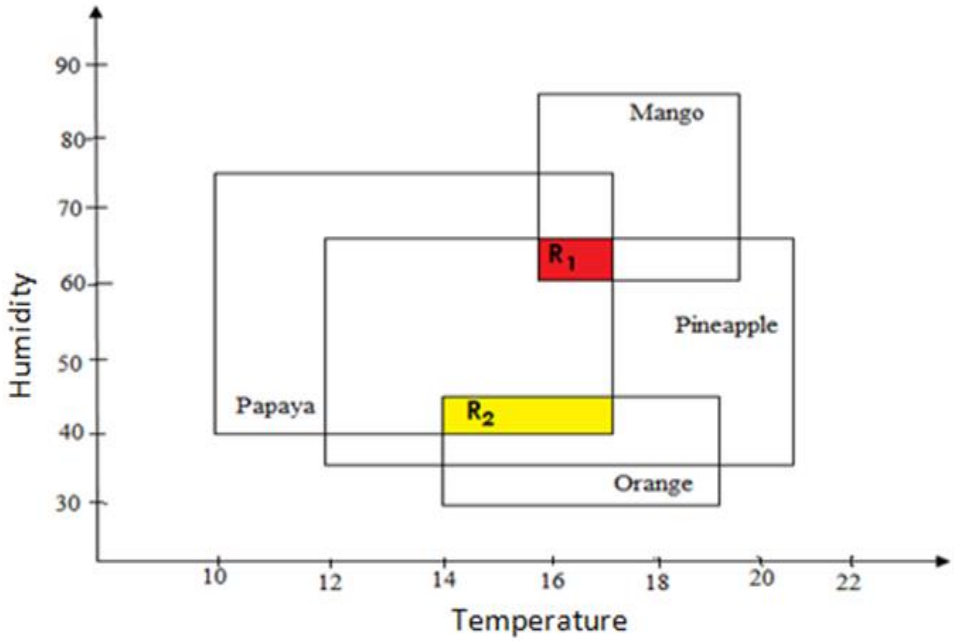

Figure 3. Dense Regions R_1 and $R_2$for temperature and humidity factors of the fruits as given in Table 1

Various rectangle data mining approaches such as mining maximal empty rectangles [18][19[20][22][23], mining holes in large datasets[21][22] and rectangle packing problems [31] have been proposed for multidimensional data. But all these approaches are not directly based on interval data.

Only a few algorithms have been proposed for mining 2D interval data truly based on interval information of underlying data. Mining closed frequent rectangles [6][8] and maximal frequent rectangles [15] are available to extract frequent relevant information from two dimensional interval data. Although, closed frequent rectangles are computed in [6][8][15] they cannot compute all closed rectangles that are present in the dataset. These proposed methods require the threshold value before they can be computed. They compute all closed frequent rectangles or maximal frequent rectangles based on this prior information of threshold. For a new threshold value, the entire process must be recompiled. Also, the methods mentioned above needs data to be preprocessed and are not incremental.

In this paper we propose an incremental algorithm named as *ACRMiner* for finding all closed rectangles and their support counts from rectangle datasets and then classify the region as dense and sparse based on any user defined threshold.The proposed algorithm uses a data structure named as *ACR-Tree* for storing all closed rectangles and their respective support counts present in the dataset. Whenever a new rectangle is inputted, the algorithm updates the *ACR-Tree* to generate the new closed rectangles along with their support counts without visiting the dataset. All the closed rectangles along with their support counts can be generated with a single pass of the dataset. All the frequent closed rectangles can also be computed for any user given minimum support with a single scan of *ACR-Tree*. Finally, based on a minimum density threshold, the regions as classified as dense and sparse regions. In section 3. a number of properties of closed rectangles and dense regions are stated and proved in the form of theorems. These results are used in the algorithm proposed in this paper.

This paper is organized into nine sections. Section 2 discusses literature reviews on related works to the problem at hand. In Section 3, some basic definitions related to mining dense region based on closed rectangle problem are given. In Section 4, a detail about *ACR-Ttee* data structure is discussed. The construction of the *ACR-Tree* and the proposed algorithm is explained in Section 5. Section 6, discusses the complexity analysis of the proposed algorithm. In section 7, the extension of proposed method to higher dimensional space is discussed. The experimental results are given in section 8. Section 9 discusses some

practical applications of the proposed method. The conclusion and scope of the future work is presented in section 10.

2. LITERATURE REVIEW

Following are some of the research works found in interval data mining.

J. F. Allen [7] published the first work in the field of interval data, in which the author defined 13 possible relationships between two intervals and proposed a method for mining knowledge from temporal intervals. The 13 possible relations are: *equal*,*starts*, *startedby*, *before*, *after*, *finishes*, *finishedby*, *overlaps*, *overlapped b*y, *during*, *contains*, *meets*, and *metby*.

P. Papapetrou et al [10] proposed an enumeration tree-based method for mining frequent interval arrangements based on J.F. Allen's 13 possible relationships among the intervals. Their proposed method, known as the A-Close algorithm, finds frequent interval item-sets.

N. Sarmah[8] proposed an algorithm for mining closed frequent intervals from an interval dataset.The proposed algorithm proposed mines the closed frequent intervals directly from interval datasets.

N. J. Sarmah and A.K. Mahanta [6] proposed an algorithm for mining closed frequent rectangles by scanning the rectangles dataset once. The algorithm needs data to be pre-processed and stored in an array. From the pre-processed dataset, the algorithm computes the closed frequent rectangles for a user defined threshold using data structures called CII, CIO, HR and CR.

I. Hazarika and A. K. Mahanta [28] proposed an algorithm to mine maximal frequent rectangles from a rectangle dataset using a data structure called IR-tree. The proposed algorithm is a combination of four algorithms named as Algorithm-A, Algorithm-B, Algorithm-C and Algorithm-D. Algorithm-*A* constructs IR-Tree for *X*-intervals (denoted as IR_x-tree) and Y-intervals (denoted as IR_y-tree) for a rectangle dataset with two domains *X* and *Y*. Algorithm-B extracts all Y-intervals with frequency (denoted as *YList*) associated with an input interval *I* from X-domain by traversing the IR_x tree. Similarly, *XList*can also be constructed by traversing the IR_y tree. Finally, *Algorithm-C* or *Algorithm-D* can be used to mine maximal frequent rectangles (MFRS) from the rectangle dataset.

Laskar et al. [11] proposed an incremental algorithm using a data structure called *SCI-Tree* to mine all closed intervals together with their support counts form an interval dataset. The *SCI-Tree* data structure is the modification of *CI-Tree* [2] data structure that not only stores the support counts of the all closed intervals but also keep a distinction between input closed intervals and generated closed intervals.

Edmonds et al. [18][19] proposed a time efficient algorithm which can find all maximal empty rectangles in large and multidimensional space with a single scan of the data sets. The algorithm needs the data to be preprocessed and sorted.

Liu et al. [20] proposed the first algorithm to identify the maximal empty rectangle (hyper-rectangles) in a k-dimensional continuous space.The proposed approach discovers the set of all possible maximal empty hyper-rectangles (MHR) from a given set of points in k dimesional space and has at least a point bounding each of its surfaces.

Liu B et al. [21] proposed algorithm for finding interesting holes in a large database. The hole is simply a region in the space that contains no data point.

Lemley et al. [22] presented a polynomial time algorithm based on Monte Carlo approach for finding largest empty holes (large hyper-rectangles) in high dimensional data where the dimensionality and input size make it challenging to analyse the data.

A. Duttaand S. Soundaralaksmi[23] developed a time and space efficient algorithm for finding maximal empty hyper rectangles (MEHR) within a bounding hyper rectangle (BHR)in three dimensional spaces.

J. Backer and J. M. Keil [29] proposed the bichromatic rectangle problem for finding the largestaxis-aligned hyperrectangle in d-dimensional space that has only blue points and no red points. All the relevant hyper-rectangles are also ranked by this proposed algorithm.

N E.Costa et al. [30] proposed a fast heat-map visualization algorithm called as OL-HeatMap to vizualize density of overlapping of several 2D axis aligned bounding boxes called as rectangles based on sweep-line paradigm.

From the literature review it has been observed that although many methods have been proposed but they basically focus on finding closed frequent intervals, closed frequent rectanglesand empty rectangles. Based on the type of input data, concept of density measuse etc.used, various featues have been identified viz. *type of input data*, *density measuring approach*, *computation of support count*, *Incremental method (yes/no)* and *outcomeof the method anda*comparative analysis of similarmethods is summarized in Table 2 to highlight the gap between these methods and our proposed work.

Table 2. Comparative Analysis of various proposed methods

Literature	Type of input data	Density Measuring approach	Computation of Support Count	Incremental (Y/N)	Outcome of the method
Sarmah and Mahanta [8]	Interval	Overlapping of intervals	It computes support counts only for the frequent intervals	No	Closed frequent intervals
Sarmah and Mahanta [6]	Interval	Overlapping of Rectangles	It computes support counts of all closed frequent rectangles	No	Closed frequent rectangles
Hazarika and Mahanta [28]	Interval	Overlapping of Rectangles	It computes support counts of maximal frequent rectangles by finding all closed frequent intervals	No	Maximal frequent rectangles
Edmonds et al. [18][19]	Point	Not required	empty regions are extracted	No	Maximal empty rectangles
Liu et al. [20][21]	Point	Not required	empty maximal hyper rectangles are extracted	Yes	All large emptyl Hyper-Rectangles (MHR)
Lemley et al. [22]	Point	Not required	big empty holes are extracted	No	Largest empty rectangles
A. Dutta and S. Soundaralaksmi [23]	Point	Not required	maximum empty rectangles are extracted	No	Maximum empty hyper rectangles (MEHR)
N E.Costa et al. [30]	Interval	Overlapping of Rectangles	Computes support counts of all overlapping rectangles	No	HeatMap Visualization of Overlapping Rectangles
Our Proposed Method	Interval	Overlapping of Rectangles	Computes support counts of all closed rectangles	Yes	Dense and Sprase Rectangular Regions

3. PROBLEM DEFINITION

The problem is to find dense and sparse regions w.r.t. a threshold value in a given rectangle database*RDB*. The concept of closed rectangle is used in the process. Various terms, definitions and theorems will be used in the problem under consideration.

3.1. Terms and Definitions used

Following terms and definitions will be used in the proposed problem of finding sense and sparse regions.

Rectangle: A rectangle *R* is defined by two intervals $I_x=[X_1,X_2]$ and $I_y=[Y_1,Y_2]$ where I_x and I_y are intervals in totally ordered domains *X* and *Y* respectively (as shown in Figure 2). If *p=(x, y)* is a point in the rectangle R then $p \in I_x \times I_y$ where, $(X_1 \le x \le X_2)$ and $(Y_1 \le y \le Y_2)$. Such a rectangle *R* can be uniquely identified by the 4-tuple $[X_1,X_2,Y_1,Y_2]$ and any such 4-tuple will uniquely identify a rectangle, i.e. $R=[X_1,X_2,Y_1,Y_2]$.

Rectangle Dataset: Let $RDB=\{r_1,r_2,r_3,\ldots,r_n\}$ be a rectangle dataset consisting of set of records, where each record$r_i$stores a 5-tuple $[X_1^i,X_2^i,Y_1^i,Y_2^i,f]$ which denotes a rectangle $R^i=[X_1^i,X_2^i,Y_1^i,Y_2^i]$ with frequency *f* in 2D space.

Containment of Rectangles: A rectangle $R=[X_1,X_2,Y_1,Y_2]$ is said to be contained in another rectangle $R'=[X_1',X_2',Y_1',Y_2']$, denoted by $R \subseteq R'$ *iff* $[X_1,X_2] \subseteq [X_1',X_2']$ and $[Y_1,Y_2] \subseteq [Y_1',Y_2']$. An interval [a, b] contains an interval [c, d], denoted as $[c,d] \subseteq [a,b]$ iff $a \le c \le d \le b$. So, containment of rectangles R and R' i.e. $R \subseteq R'$ can be defined as $(X_1' \le X_1 \le X_2 \le X_2')$ and $(Y_1' \le Y_1 \le Y_2 \le Y_2')$.

Proper Containment of Rectangles: A rectangle*R*is said to be properly contained in another rectangle R' denoted by $R \subset R'$ iff $(([X_1,X_2] \subset [X_1',X_2']$ and $[Y_1,Y_2] \subseteq [Y_1',Y_2']))$ or $(([X_1,X_2] \subseteq [X_1',X_2')]$ and $[Y_1,Y_2] \subset [Y_1',Y_2']))$. An interval [a, b] properly contains an interval [c, d], denoted as $[c,d] \subset [a,b]$ iff $(a < c \le d \le b)$ or $(a \le c \le d < b)$ or $(a < c \le d < b)$. So, if $R \subset R'$ then $((X_1' < X_1 \le X_2 \le X_2')$ and $((Y_1' \le Y_1 \le Y_2 \le Y_2'))$ or $((X_1' \le X_1 \le X_2 \le X_2')$ and $((Y_1' \le Y_1 \le Y_2 \le Y_2'))$ or $((X_1' \le X_1 \le X_2 \le X_2')$ and $((Y_1' < Y_1 \le Y_2 \le Y_2'))$ or $((X_1' \le X_1 \le X_2 \le X_2')$ and $((Y_1' \le Y_1 \le Y_2 < Y_2'))$

Support of a Rectangle: Given a rectangle dataset RDB, supports of a rectangle $R=[X_1, X_2, Y_1, Y_2]$ is the sum of the frequency values of the rectangles in RDB that contains R.

Closed Rectangles: A rectangle R is said to be closed if the support of any rectangle R' properly containing R is less than the support of R i.e. if $R \subset R'$ and if $s1$ and $s2$ are supports of R and R' respectively then $s2 < s1$.

Intersection of Intervals: An interval $I_X=[X_1, X_2]$ is said to intersect interval $I'_X=[X'_1, X'_2]$ if the intersection $(I_X \cap I'_X)$ is non-empty i.e. $(X_1 \le X'_1 \le X_2 \le X'_2)$ or $(X_1 \le X'_1 \le X'_2 < X_2)$ or $(X'_1 \le X_1 \le X'_2 \le X_2)$ or $(X'_1 \le X_1 \le X_2 < X'_2)$. The intersection of two intervals $I_X=[X_1, X_2]$ and $I'_X=[X'_1, X'_2]$ is an interval $I^r_X=[X^r_1, X^r_2]$ denoted as $I^r_X=(I_X \cap I'_X)$ where intersection is defined in the usual way defined for sets.

Intersection of Rectangles: A rectangle $R=[X_1, X_2, Y_1, Y_2]$ is said to be intersect rectangle $R'=[X'_1, X'_2, Y'_1, Y'_2]$ if the intersection $R^r=(R \cap R')$ is non-empty. This will hold iff $([X_1, X_2] \cap [X'_1, X'_2])$ and $([Y_1, Y_2] \cap [Y'_1, Y'_2])$ are both non-empty. The result of $(R \cap R')$ is a rectangle $R^r=[X^r_1, X^r_2, Y^r_1, Y^r_2]$ where $[X^r_1, X^r_2]=([X_1, X_2] \cap [X'_1, X'_2])$ and $[Y^r_1, Y^r_2]=([Y_1, Y_2] \cap [Y'_1, Y'_2])$.

Equality of Rectangles: Two rectangles $R=[X_1, X_2, Y_1, Y_2]$ and $R'=[X'_1, X'_2, Y'_1, Y'_2]$ are said to be equal denoted as $R=R'$ iff $X_1=X'_1$, $X_2=X'_2$, $Y_1=Y'_1$ and $Y_2=Y'_2$.

Density value of a region: The density value or support of a region is the number of input rectangles that contain it.

Dense and Sparse Regions: A region R is called dense if its support is greater than any user defined minimum support threshold ρ. A region which is not dense is termed as sparse.

Maximal Dense Region: A dense region is said to be maximal if it is not properly contained in any dense region.

3.2. Theorems used in the Proposed Method

The following two theorems of closed rectangles that were proved in [6] will be used in designing our proposed algorithm and for the sake of completeness these have been stated below.

Theorem A. If $[X_1, X_2, Y_1, Y_2]$ is a closed rectangle then

(i) There is an input rectangle with X_1 as its left end value containing $[X_1, X_2, Y_1, Y_2]$.
(ii) There is an input rectangle with X_2 as its right end value containing $[X_1, X_2, Y_1, Y_2]$.
(iii) There is an input rectangle with Y_1 as its lower end value containing $[X_1, X_2, Y_1, Y_2]$.
(iv) There is an input rectangle with Y_2 as its upper end value containing $[X_1, X_2, Y_1, Y_2]$.

Proof: Proof of Theorem-A is given in [6].

Theorem B. A rectangle $[X_1, X_2, Y_1, Y_2]$ is closed if all of the following are satisfied:

(i) X_1 is the left end value of an input rectangle say R, part(X_1) ≥ X_2 and the Y-interval of input rectangle R contains [Y_1, Y_2].
(ii) X_2 is the right end value of an input rectangle say R', part(X_2) ≤ X_1 and the Y-interval of input rectangle R' contains [Y_1, Y_2].
(iii) Y_1 is the lower end value of an input rectangle, part(Y_1) ≥ Y_2 and the X-interval of the input rectangle contains [X_1, X_2].
(iv) Y_2 is the upper end value of an input rectangle, part (Y_2) ≤ Y_1 and the X-interval of the input rectangle contains [X_1, X_2].

Proof: Proof of theorem-B is given in [6].

In addition to above theorems we have proposed following theorems that will be used in our work.

Theorem 1. If $R=[X_1, X_2, Y_1, Y_2]$ is a rectangle properly containing $R'=[X'_1, X'_2, Y'_1, Y'_2]$ i.e. $R' \subset R$ then $\sup(R') \ge \sup(R)$.

Proof*:* It is obvious since all input rectangles in the rectangle dataset containing $R=[X_1, X_2, Y_1, Y_2]$ will also contain $R'=[X_1', X_2', Y_1', Y_2']$ and so $\sup(R') \geq \sup(R)$.

Theorem 2. If $R=[X_1, X_2, Y_1, Y_2]$ is a rectangle in the input dataset then R is closed.

Proof*:* Let, $R'=[X_1', X_2', Y_1', Y_2']$ be a rectangle properly containing R. i.e. $R \subset R'$. Then, $\sup(R') \leq \sup(R)$. Now, *R* is a rectangle in the input dataset containing rectangle R but not containing the rectangle R'. So, $\sup(R') < \sup(R)$. Therefore, R is a closed rectangle.

Theorem 3. The intersection of two intersecting input rectangles R_1 and R_2 is a closed rectangle.

Proof*:* Let, R_1 and R_2 be two input rectangles. Suppose the rectangles are as shown in Figure 4.

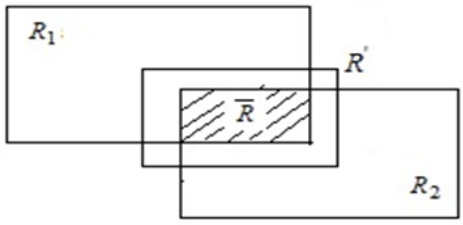

Figure 4. Intersection of closed rectangles R_1 and R_2

Let, $\bar{R}=[X_1, X_2, Y_1, Y_2]$ be the intersection of R_1 and R_2. Let, $R'=[X_1', X_2', Y_1', Y_2']$ be an input rectangle that properly contains rectangle $\bar{R}$. Then $\sup(R') \leq \sup(\bar{R})$. Since, R' properly contains R, at least one (or more) of the followinginequalities will have to be true

(i) $X_1' < X_1$

(ii) $X_2 < X_2'$

(iii) $Y_1' < Y_1$

(iv) $Y_2 < Y_2'$

If (i) or (iv) is true then R_2 does not contain R'.

If (ii) or (iii) is true then R_1 does not contain R'.

In either case, support of R' will be less than that of $\bar{R}$. The proof will follow in the same way for all non-empty intersections of rectangles R_1 and R_2. Hence, $\bar{R}$ is a closed rectangle.

Theorem 4. A rectangle $R=[X_1, X_2, Y_1, Y_2]$ is a closed rectangle iff it is the intersection of four input rectangles all of which may not be distinct.

Proof*: Theorem-A* states that if $R=[X_1, X_2, Y_1, Y_2]$ is a closed rectangle than all its four sides are parts of input rectangles containing R. Hence R is the intersection of these rectangles. These four rectangles however may not be distinct. *Theorem-B* states that if a rectangle R is such that all its four sides are parts of input rectangles containing R then R is a closed rectangle. In this case R will be the intersection of these four rectangles.

Theorem 5. Intersection of two closed rectangles is a closed rectangle.

Proof*:* Let, R_1 and R_2 be two closed rectangles and Let, $\bar{R}$ is their intersection. Suppose $\bar{R}$ is not closed. Then, there is a rectangle properly containing $\bar{R}$ (say R') whose support is same as $\bar{R}$. i.e. $\sup(\bar{R}) = \sup(R')$. But as R_1 is a closed rectangle, the left side of the rectangle R_1 will be a part of an input rectangle which contains R_1 [*Theorem-A and Theorem-B*]. Then this input rectangle will contain $\bar{R}$ but not R' and hence support of $\bar{R}$ cannot be same as R'. The same argument can be applied to any R' containing $\bar{R}$. Therefore, there cannot be a rectangle properly containing $\bar{R}$ having the same support as that of $\bar{R}$ and hence $\bar{R}$ is closed.

Although our work is concentrated on rectangular regions but the concept can be extended to any well-defined region of arbitrary shape. *Theorem 6* and *Theorem 7* are proved in this direction.

Theorem 6. Support of a region r is the support of the smallest closed rectangle containing r. If no such rectangle exists then the support of r is zero.

Proof*:* Let*r*is region as shown in Figure 5. If*r*is not contained in any closed rectangle then*r*is not contained in any input rectangle either (*Theorem-1 or Theorem 2*) and hence support of *r*is zero.
Now, suppose R is the smallest closed rectangle containing *r*. Since $r \subseteq R$, $\sup(r) \geq \sup(R)$.

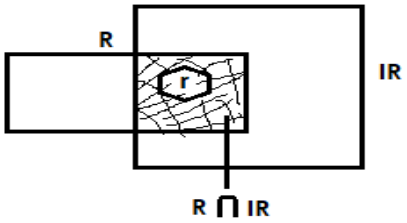

Figure 5. Support of a region*r*is the support of the smallest closed rectangle containing*r*.

Suppose, $\sup(r) > \sup(R)$. Then there is at least one input rectangle containing *r*but not containing*R*. Let, *IR*be one such rectangle. Let us consider the rectangle $R \cap IR$. It is non empty as it is contains*r*. It is closed by *Theorem-5*.
Also, $(R \cap IR) \subset R$ as $R \not\subseteq IR$. Therefore $R \cap IR$ is a closed rectangle containing r which is a contradiction to such assumption that *R*is the smallest closed rectangle containing*r*. Therefore, $\sup(r) = \sup(R)$

Theorem 7. Let ρ be the minimum support value used for defining dense regions. Then any closed rectangle of support ρ will be a maximal dense region.

Proof*:* Suppose R is closed rectangle of support ρ and the region R is not maximal as shown in Figure 6.
Then, there is a region r such that $r \supset R$ and r is dense. Form the Theorem-6, support of r is the support of the smallest closed rectangle containing r.

Let, $\bar{R}$ be such a rectangle. Then $\bar{R} \supseteq r \supseteq R$ and $\sup(r) = \sup(\bar{R})$. Since, $\bar{R} \supset R$ and r is closed $\sup(\bar{R}) < \sup(R)$ contradicting our assumption that r is dense.Hence, R is a maximal dense region.
Form the *Theorem-6* and *Theorem-7* it is clear that for any minimum support threshold ρ, the closed rectangles of support ρ are the maximal dense regions. In case there is no closed rectangle of support ρ then the next integer greater than ρ say ρ' having closed rectangles will serve the purpose. If there are no such ρ' values then there will be no dense region.

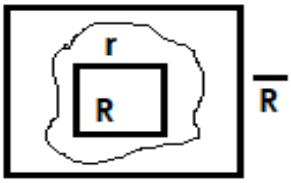

Figure 6. Maximal dense region R with support ρ

4. ACRMINER FOR COMPUTING ALL CLOSED RECTANGLES

The proposed algorithm uses a tree like data structure named as *ACR-Tree* to compute all closed rectangles incrementally. Since the algorithm is incremental it can process dynamic data. It does not require any preprocessing of input data. Whenever a new rectangle is given as input it automatically updates its data structure *ACR-Tree*. The input rectangles are given one by one in a sequence to the algorithm. Each record in RDB contains a rectangle $R = [X_1, X_2, Y_1, Y_2]$. Here, X_1 and X_2 refers to the left and right end point of the interval $[X_1, X_2]$ in X-domain. Similarly, Y_1 and Y_2 refers to the lower and upper end point of the interval $[Y_1, Y_2]$ in Y-domain. For the construction of *ACRMiner*, the *theorems* and *definitions* as discussed in *Section 3*are used. The proposed *ACRMiner* contains two types of lists of nodes- a header-list IR_LIST and a number of sub-lists. There is one sub-list *CR_LIST* for each node in the header-list *IR_LIST*. Any node in the header-list stores a 9-tuple $(X_1, X_2, Y_1, Y_2, area, freq, \sup, IRNext, CRNext)$, where X_1, X_2, Y_1 and Y_2 refers to the *left*, *right*, *lower* and *upper*

end points respectively of an input rectangle R respectively. Each node in *IR_LIST*is associated to a distinct input rectangle $R=[X_1, X_2, Y_1, Y_2]$ in dataset *RDB*. The field *area* stores the area of the rectangle R where $area=|(X_2-X_1)*(Y_2-Y_1)|$. The value of $area$ of a header-node is important in our proposed implementation because the newly created sub-list node will be added to the sub-list of the header node having smaller $area$. All the rectangles in the sub list of a header node are contained in the rectangle represented by that header node. A larger-area rectangle is more likely to intersect with a newly inputted rectangle than a smaller-area rectangle. If a large area rectangle contains more rectangles, the cost of rectangle intersection computation increases. However, all of the rectangles in that larger rectangle's sub-list may not intersect with the newly entered rectangle. To reduce the cost of computation of intersection of rectangles the sub-list of large area rectangle is kept short. For that reason, when a new node is created as a result of the intersection of a header node and the new input rectangle, the resulting node is added to the sub list of that header node whose rectangle at the node has the least area.

The field $freq$ stores the frequency count of the rectangle*R*in the input rectangle database. The field $\sup$ stores the support counts of the rectangle *R* in the same input rectangle database.*IRNext* and *CRNext* are two pointers associated with each header node e_H in *IR_LIST*. The pointer *IRNext* points to the node next to node e_H in *IR_LIST*. The pointer *CRNext* points to the node in the sub-list $^{CR\_LIST}e_H$ associated to e_H. Any node $e_L$in sub-list $^{CR\_LIST}e_H$ stores a 6-tuple $(X_1', X_2', Y_1', Y_2', count, \sup, LNext)$, where X_1', X_2', Y_1' and Y_2' refers to the rectangle $R'=[X_1', X_2', Y_1', Y_2']$. Each rectangle in a sub-list is a non-empty intersection of a number of input rectangles. The field $count$ stores this number. The pointer *LNext* points to nodenext to e_L in $^{CR\_LIST}e_H$ .The field $\sup$ stores the support counts of the rectangle R' in the same input rectangle database.

A node e_L in $^{CR\_LIST}e_H$ of a header node with rectangle R represents a rectangle which is a non-empty intersection of *R* with some rectangle in *IR_LIST* or with some rectangle in some sub-list $^{CR\_LIST}I$.

In the construction of *ACR-Tree*, whenever a new input rectangle comes it is inserted into the header list *IR_LIST* as a node. The newly inserted header node contains its calculated value of *area*, *sup* and *freq*. If the newly insterted input rectangle already exits as a node in the header list then only the *sup* and *freq*values are updated. Once the header node is inserted in *ACR-Tree* then the non-empty interestions between the newly inserted rectangle and the rectangle present at other existing header nodes are inserted into the sublist *CR_LIST* of the header node having the smallest *area* between them. The non-empty intersections of the newly inserted rectangle and the rectangles present at sublist of each header node is also inserted into the sublist of suitable header nodes. The *count* and *sup* values of all the respective nodes are also updated. All the rectangles present in header list and sublists are closed rectangles. The detail construction of the *ACR-Tree* is explained in section 5. Apictorial view of a sample structure of *ACR-Tree* is shown in Figure 7 where $Ri=[X_1^{R_i}, X_2^{R_i}, Y_1^{R_i} Y_2^{R_i}]$ for $1\le i\le n$ are the input rectangles. The nodes that are linked vertically represent the header list.Sublist of a header node is shown by the list of nodes linked horizontally. In Figure 7, $[X_1^{R_1}, X_2^{R_1}, Y_1^{R_1}, Y_2^{R_1}, \sup]$, $[X_{11}^{R_1^1}, X_{12}^{R_1^1}, Y_{11}^{R_1^1}, Y_{12}^{R_1^1}, \sup]$, $[X_{11}^{R_1^2}, X_{12}^{R_1^2}, Y_{11}^{R_1^2}, Y_{12}^{R_1^2}, \sup]$….., $[X_1^{R_2}, X_2^{R_2}, Y_1^{R_2}, Y_2^{R_2}, \sup]$,…….…… $[X_1^{R_n}, X_2^{R_n}, Y_1^{R_n}, Y_2^{R_n}, \sup]$ are all the closed rectangles along with their supports.

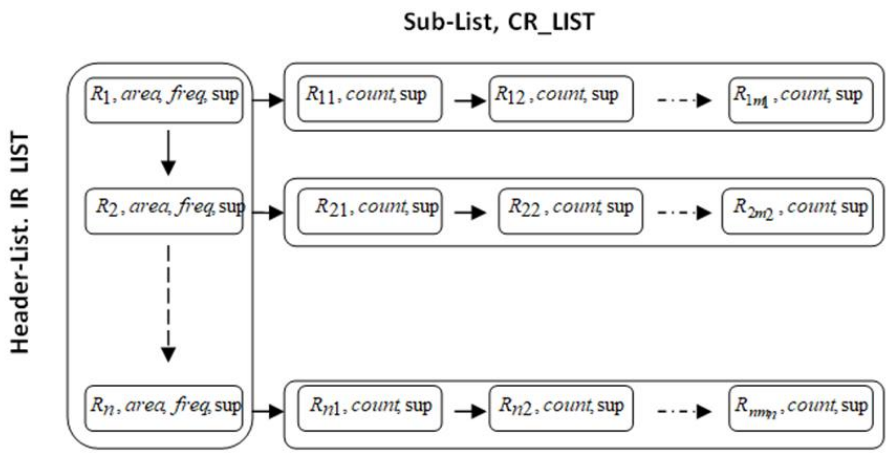

Figure 7. Pictorial Representation of ACR-Tree Data structure

5. CONSTRUCTION OF ACRMINER

The proposed algorithm*ACRMiner* for construction of *ACR-Tree*can be divided into three parts, viz. *Header-List–Update*, *Sub-List-Update*and *Final-Support-Update*. The first two parts are used to construct the *ACR-Tree* and the last part i.e. *Final-Support-Update* part is used to compute the support counts of the the nodes that are present in the constructed *ACRMiner*. In *Header-List-Update* part, the procedure *IR_LIST_Update* is used to upate the header list *IR_LIST* in *ACR-Tree*. A counter $I_{contain}$ is used which stores the sum of the frequency counts of input rectangles that contain the newly inserted rectangle$I$=[X_1,X_2,Y_1,Y_2].The procedure*IR_LIST_Update* also updates the supports of all existing header nodes. The second part i.e. *Sub-List-Update* uses aprocedure called *CR_LIST_Update* to update the sub-list *CR_LIST* associated with each node in *IR_LIST*. The *CR_LIST_Update* procedure is called when any new node *I* is added to the header-list *IR_LIST*.The procedure *CR_LIST_Update* is used for generation of new closed rectangles. The newly generated closed rectangles are added to the sub lists *CR_LIST* in *ACR-Tree*. The third part *i.e. Final-Support-Update* uses a procedure *FS_Update*. This procedure scans the *ACR-Tree*and updatesthe support of the nodes present in the sub lists *CR_LIST's* of *ACR-Tree*. All the nodes in *ACR-Tree* represent closed rectangles as each is a non-empty intersection of a number of input rectangles. However, more than one node may represent the same closed rectangle. In such cases, any one of them is chosen as all such nodes have same suppor count and the rest nodes are ignored. For an input rectangle I=[X_1,X_2,Y_1,Y_2] with frequency *f*, these threeparts involved in updating of *ACR-Tree* are discussed in depthin following sections.

5.1. Header-List-Update part

The procedure *IR_LIST_Update* updates the header list in *ACR-Tree*. The counter $I_{contain}$ is initialized to *zero* before the header-list modification starts for each newly inserted input rectangle. While updating the header list of *ACR-Tree*, when a new rectangle $I=[X_1, X_2, Y_1, Y_2]$ with frequency*f* isinputted then $I_{contain}$is updated as $I_{contain}=I_{contain}+freq$ for each rectangle with frequency $freq$ in theheader-list of*ACR-Tree* that contains *I*. Now, if the rectangle *I* is not present in the header list*IR_LIST*then $I=[X_1, X_2, Y_1, Y_2]$ is inserted as a node p at the end of the header-list IR_LIST. The frequency *freq* and support *sup* of the node p is set as $p.freq=f$ and $p..\sup=f+I_{contain}$ respectively. The area of the node p is also computed and it is set as $p.area=|(X_2-X_1)*(Y_2-Y_1)|$. If the rectangle*I* is already present at a node $e_H$in the header-list of *ACR-Tree* then the frequency and support of the node e_H is updated as $e_H.freq=e_H.freq+f$ and $e_H.\sup=e_H.\sup+f$ respectively. The supports of all other header nodes e_H representing rectangles that are contained in I are updated as $e_H.\sup=e_H.\sup+f$. Now, procedure *CR_LIST_Update* is called for modification of the sublist in *ACR -Tree*if any new node is added to the header list *IR_LIST*.

5.2. Sub-Lists-Update part

As stated earlier, the procedure *CR_LIST_Update* updates the sub-lists in *ACR-Tree*. When a new rectangle $I=[X_1, X_2, Y_1, Y_2]$ with frequency *f* is added as a node*p*into the header list then the sub lists of all the header nodes need to be updated. A node e_L in the sub lists of a header node e_H in *ACR-Tree* represenets a rectangle R' with support count *sup*. In*CR_LIST_Update*, all the header nodes e_H and their sub lists are traversed. The intersection of rectangle *I*at node *p* with the rectangles present in all the header nodes except *p* and their sub listsare computed and non-empty intersections are inserted into the sub list*i.e.* if $e_H \neq p$ is a header node in *ACR-Tree* and the rectangle *R* at node e_H has a non empty intersection with the rectangle *I* at *p*then it is selected for generation of new rectangles. If *R* is contained in *I* then the travseral of the nodes in the sub list of e_H is not required because all the rectangles in the sub list of headernodee_H are also contained by *I* and hence generatesno new rectangles. If rectangle *R* is not contained in rectangle *I* and has non empty intersection thena new rectangle is generated. This newly generated rectangle is added as a new node to the sub list of e_H or *I* depending on whichever have the rectangle with smaller area between these two. If both the rectangles at e_H and *I* have same area then the new rectangle is added as a node e_R to the sublist of e_H. The value of $count$ of e_R is set as 2 i.e. $e_R.count=2$. The justification for setting of $e_R.count=2$ is that e_R is a new rectangle formed by the intersection of two input rectangles at e_H and I. The support count of e_R is set as zero i.e. $e_R.\sup=0$. Now, all the nodes in the sublist of e_H are traversed. If e_L is a node in the sublist of e_H then intersection of I is taken with the rectangle R' at nodes e_L in the sub list of e_H having $e_L.count<4$ for generation of new rectangles. If this intersection generates a new closed rectangle R_{new} then it is added as a node e_K to the sub list of current header node e_H if its area of header node e_H is smaller or equal to the area of R i.e.

$e_H.area \leq I \cdot area$. Otherwise, e_K is added to the sub-list of I. Whenever such a new rectangle e_K is generated its counter *count* of node e_K is set to $e_k \cdot Count = e_L \cdot Count + 1$. This is because e_K is the result of intersection of e_L and input rectangle R is the intersection of $e_L \cdot count$ numbers of input rectangles. In sub-list modification, only the nodes e_L in the sublist of e_H having $e_L \cdot count < 4$ are considered for generation of new closed rectangles. This is justified because any closed rectangle is the intersection of at most four input rectangles (from The*orem-4* as mentioned in section 3). The supports of e_K set as $e_K.\sup = 0$

5.3. Final-Support-Update part

Finally, the procedure *FS_Update* is called for the modification of the support of the nodespresentin the sub lists. In this part, supports of the all the nodes in each sub list *CR_LIST* of *ACR-Tree* are updated by scanning the header nodes e_H*'s* in the header list *IR_LIST* of *ACR-Tree*. In *FS_Update*, for each node e_L in *CR_LIST* all the header-node sare visited and support of e_L is updated by adding the frequency of each header node e_H in *IR_LIST*that contains e_L. So, support of e_L is modified as $e_L \cdot \sup = e_L \cdot \sup + e_H \cdot freq$ for each such header node e_H. The support count of node $e_L$isinitialized as $e_L \cdot \sup = 0$ before its support computation.The algorithm for construction of ACR_Tree is given below.

Algorithm: *ACRMiner*- Constructs the *ACR-Tree*
Input: Rectangle Database, *RDB*
Output: *ACR-Tree* containing all closed rectangles with support and frency count

Step1: for each rectangle in RDB **do**
Step2: Update header list *IR_LIST* using procedure *IR_LIST_Update*
Step3: Update Sub list *CR_LIST* using procedure *CR_LIST_Update*
Step4: Update final support count of all the rectangles in the sub list *IR_LISTs* using procedure *FS_Update*
Step5: Report Costructted*ACR-Tree*

5.4. Example

Let us consider the following Rectangle database *RDB*as shown in Table 3 and construction of *ACRMiner* is elaborated as shown below with diagrams.

Table 3. Rectangle database, *RDB*

Record No	Rectangle, $I=[X_1,X_2,Y_1,Y_2]$	Frequency, *f*
I_1	1, 8, 2, 6	1
I_2	1, 4, 1, 3	1
I_3	5, 8, 2, 6	1
I_4	2, 7, 7, 9	1
I_5	5, 8, 2, 6	2

For the input rectangle *I*=[1, 8, 2, 6] with frequency *f=1*, first *IR_LIST_Update* procedure is called. Initialization of $I_{contain}=0$ is done and $I_{contain}$ for *I*is computed. Since, header list *IR_LIST* is empty and hence $I_{contain}=0$. The frequency, support and area of node, *I*=[1, 8, 2, 6, 28, 1, 1] is calculated as *I.freq=f*, $I.sup=f+I_{contain}$ and $I.area=|(X_2-X_1)*(Y_2-Y_1)|$. The input rectangle *I* is added to the header list *IR_LIST* as node, *I*=[1, 8, 2, 6, 28, 1, 1] (using *IR_LIST_Update*) where *I.area=28*, *I.freq=1* and *I.sup=1* is the area, frequency and support of the rectangle [1, 8, 2, 6] respectively which is represented by node *I*. Since, it is the only node in the header list, therefore no modification of the sub list is done. Finally, support counts of all the nodes in sub list are updated using the procedure *FS_Update*. Since the sub list is empty so updation process ends. Figure 8, shows the modified *ACR-Tree* after insertion of*I*=[1, 8, 2, 6] with frequency *f=1*.

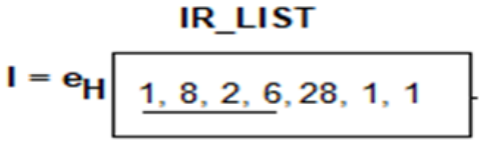

Figure 8. ACR-Tree after the insertion of *I*=[1, 8, 2, 6], frequency, *f=1*

For the input rectangle *I*=[1, 4, 1, 3] with frequency *f=1*, header node*I*=[1, 4, 1, 3] is added to the *ACR-Tree* using *IR_LIST_Update* procedure as node *I*=[1,4,1,3,6,1,1] where*I.area=6*, *I.freq=1+*$I_{conatin}$*=1+0=1* and *I.sup=1* is the area, frequency and support of the rectangle [1, 8, 2, 6] respectively which is represented by node *I*. Since, *I* is not contained by any of the recatmgles at header niodes and hence $I_{contain}=0$. Now, the sub-list modification is processed using *CR_LIST_Update* procedure following the steps as discussed above**.** Newly generated rectangle [1, 4, 2, 3] is added to the sub-list of the header node [1, 4, 1, 3, 6, 1, 1] as node e_R= [1, 4, 2, 3, 2, 0] where $e_R.count=2$ and $e_R.sup=0$ is the count and support of the rectangle [1, 4, 2, 3] respectively which is represented by node e_R. Finally, support counts of all the nodes in sub list are updated using the procedure *FS_Update*. There is one such node e_R= [1, 4, 2, 3, 2, 0] is present in the list. The support of this node is the sum of the frequencies of all the nodes that contains it. Hence, its support is set as $e_R.sup=1+1=2$. Figure 9, shows the modified *ACR-Tree* after insertion of *I*=[1, 4, 1, 3] with frequency*f=1*.

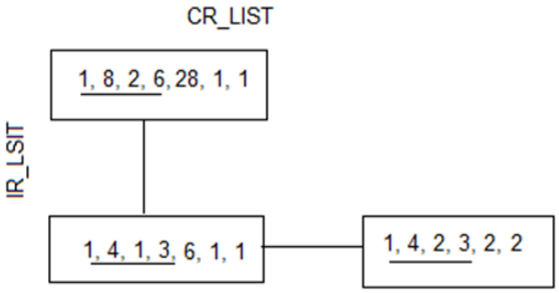

Figure 9. ACR-Tree after the insertion of *I*=[1, 4, 1, 3], frequency, *f=1*

Following the same procedure records I_3 and I_4 is inserted to the *ACR-Tree* and the tree is updated accordingly. Figure 10 and Figure 11 shows the updated *ACR-Tree*.

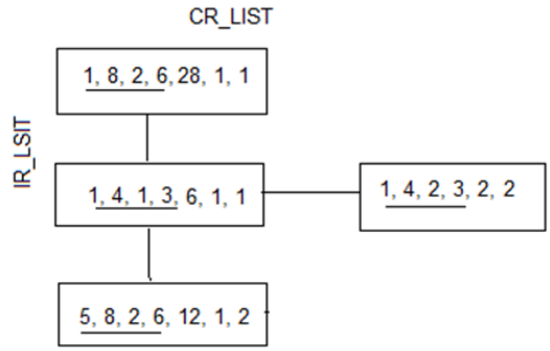

Figure 10. ACR-Tree after the insertion I=[5,8,2,6] with frequency, f=1

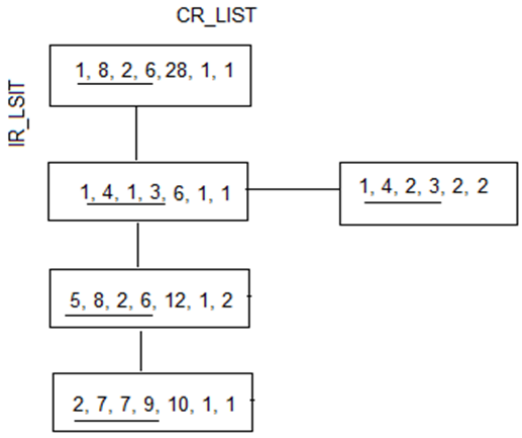

Figure 11. ACR-Tree after the insertion I=[2,7,7,9] with frequency, f=1

For the input rectangle *I*=[5, 8, 2, 6] with frequency *f=2*, the *ACR-Tree* is updated using the using *IR_LIST_Update* procedure. Since, therectangle [5, 8, 2, 6] is already present in the header list as node *I*=[5, 8, 2, 6, 12, 1, 2]. So, $I_{contain}$ count is ignored. Only the frequency and support of the header node representing the rectangle [5, 8, 2, 6] needs to updated. So, frequency and support of this header node are set as *I.freq=I.freq+f=1+2=3* and *I.sup=I.sup+f=2+2=4*. Now, procedure *FS_Update*is called forthe modifications of support coutnts of nodes in the sub lists *CR_LIST*.The final modified tree is as shown in Figure 12.

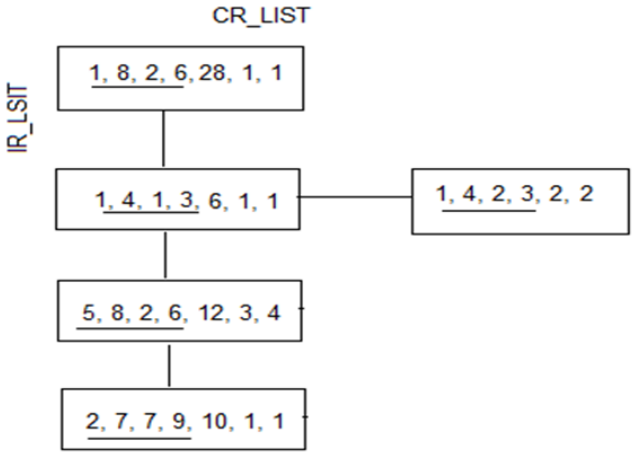

Figure 12. ACR-Tree after the insertion I=[5,8,2,6] with frequency, f=2

The output of *ACR-Miner*algorithm gives all closed rectangles together with their support countsuing the datastructure*ACR-Tree*. A total of 5 closed rectangles are found and their support counts are also computed and they areviz.{[1,8,2,6],1}, {[1,4,1,3],1}, {[1,4,2,3],2}, {[5,8,2,6],4} and {[2,7,7,9],1}**.** Now, if density threshold is considered as 2 then the resultant dense regions are: {[1,4,2,3],2}and {[5,8,2,6],4}

6. COMPLEXITY ANALYSIS OF THE PROPOSED ALGORITHM

In *ACR-Tree*, Header List *IR_LIST* is unsorted and new nodes are always added at the end of the Header List. The size of the Header List is equal to the number of input rectangles. The sub lists are also unsorted and new nodes are always added at the beginning of the lists. Linked list implementation is used for updation of *IR_LIST* and *CR_LIST's* in *ACR-Tree*. So, *O(n)*steps are required for header list updating for a given input rectanglewhere *n* is the number of the rectangles in the input data set. The sub list$CR_LIST_{eH}$of each header node ^{eH} is updated by computing the non empty intersections of the rectangles at each header node and at the correspondingsub list with the input rectangle. Since, any closed rectangle is the intersection of at most four input rectangles, so the maximum number of possible closed rectangles will be: $^{n}C_1 + ^{n}C_2 + ^{n}C_3 + ^{n}C_4 = O(n^4)$

Each rectangle in the sub lists is the intersection of a number of input rectangles. A count value is stored with each node in the sub lists to store this number. If this value is *4* (*four*) then no more intersections are taken with that rectangle. Therefore the total number of nodes in the sub lists is $O(n^4)$. Thus the worst case time complexity for updating of *ACR-Tree* for one input rectangle is $(O(n) + O(n^4))=O(n^4)$. If the dataset has *n* input rectangles then for inserting *n* input rectangles in *ACR-Tree*, the proposed algorithm will require $n * O(n^4) = O(n^5)$ time in worst case. Each rectangle in the sub lists is evaluated with all the nodes in the header list*IR_LIST*to update its support counts. Since, there are atmost$O(n^4)$ rectangles and number of the nodes in the header list can beatomost*n*, so the worst time complexity for updating of supports of nodes in sub lists of *ACR-Tree* for one input rectangle is $n* O(n^4)=O(n^5)$. Thus total worst case time complexity for the complete modification of *ACR-Tree*is $O(n^5)+ O(n^5)=2*O(n^5)= O(n^5)$. On average the algorithm will however perform better than this. For most of the input rectangles the number of intersection computed will be less. This is because if the intersection of the input rectangle with the rectangle at a header node is null then all rectangles in the corresponding sub list will also have empty intersection and so the sub list of the header node is not traversed. Also while adding nodes to sub lists, the header node with small rectangle size is taken so that the probability of occurrences of such cases increases. The sizes of the sub-lists will also be much smaller on average.

7. EXTENSION OF THE PROPOSED ALGORITHM TO HIGHER DIMENSIONAL SPACE

The problem that has been discussed so far in the 2-dimensional space can be extended easily to 3-dimensional space also. In 3-dimensional space the corresponding problem will be to mine closed frequent cubes from an input dataset of cubes. A cube can be represented as $C=[I_x, I_y, I_z]$, where I_x, I_y, I_z are one dimensional intervals in *X*, *Y* and *Z* domains respectively. Frequent cubes, closed frequent cubes and maximal frequent cubes are straight forward extensions of the corresponding concepts for intervals and rectangles. The theorems in *section 3* can be extended to higher dimensional space. A closed cube will be the intersection of six input cubes all of which may not be distinct. Computing the intersection of two rectangles involves computing the intersections of two pairs of intervals and for cubes it involves computing intersections of three pairs of intervals. Thus, computing the intersections of two hyper cubes is linear in the dimension of the underlying space. In the *ACR-Tree*data structure, the nodes will store cubes where each cube will be represented by three intervals. The number of closed cubes can be at most $O(n^6)$ and so the overall complexity will be $n^* O(n^6) = O(n^7)$.

8. EXPERIMENTAL EVALUATION OF PROPOSED ALGORITHM

The proposed algorithm is implemented by developing C++ programs in Linux PC having Intel Core i5 processor with 4 GB RAM. To validate the proposed method, the algorithm has been tested with three real life datasets *Fruit datasets, Vegetables data sets* and *Car datasets.* The proposed method is also tested with a number of synthetic datasets**.**

8.1. Results Applied on Fruit Dataset

The fruit dataset is used to test our proposed technique. This dataset is collected from *https://extension.umaine.edu/publications/4135e/.* This data set contains information about suitable temperature and humidity ranges (min-max) for 16 different fruits, which is recommended for suitable temperature and humidity storage conditions of the fruits [26]. Temperature and humidity ranges are depicted as X-domain and Y-domain respectively. The propsed algorithm was applied on this dataset and the result is shown in Table 4. A total of obtained 10 closed rectangles together with their support counts for the fruit data set.Table4 shows two closed rectangles [31, 31, 90, 90] and [32, 32, 90, 90] with support count of 13, indicating that they are dense regions with density value 13. The dense rectangle [31,31,90,90] shows that if the temperature and humidity values are kept fixed at 31 and 90 respectively then 13 numbers of fruits will be preserved. Similarly same number of fruits will be preserved if temperature and humidity values are kept fixed at 32 and 90 respectively.The other solutions such as [30, 31, 90, 95] and [31, 32, 90, 95] allow for some variation in temperature and humidity values and larger rectangles with density values less than 13 may be chosen for this.

Table 4. Closed Rectangles with support values generated from fruit dataset

Closed Rectangles	Support Count
[30, 40, 90, 95], [31, 32, 85, 85], [29, 31, 90, 95]	1
[30, 31, 90, 95]	3
[31, 32, 90, 95]	10
[31, 32, 90, 90]	11
[32, 32, 90, 95], [31, 31, 90, 95]	12
[32, 32, 90, 90], [31, 31, 90, 90]	13

Numbers of dense closed rectangles computed by the proposed method with varying density thresholds are shown in Table 5. Form the result it is found that there is no dense region for the density threshold value, ρ =14.

Table 5. No of dense closed rectangles with varying density thresholds, ρ on fruit data set

Density threshold, ρ (min)	No of Dense Closed Rectangles
2	7
4	6
8	6
10	6
12	4
14	0

8.2. Results Applied on Vegetable Dataset

The vegetable dataset is used to test our proposed technique. This dataset is also collected from *https://extension.umaine.edu/publications/4135e/*. This data set contains information about suitable temperature and humidity ranges (min-max) for 69 different vegetables, recommended for their suitable storage [26]. Temperature and humidity ranges are depicted as X-domain and Y-domain rectangles for evaluation of the proposed method. The proposed method generated 52 closed rectangles together with their support counts for this data set. Numbers of dense closed rectangular regions for varying density thresholds are shown in Table 6. Form the result, it is found that there is no dense region for the density threshold value, ρ =40.

Table 6. No of dense closed rectangles with varying density thresholds, ρ onVegetable data set

Density threshold, ρ (min)	No of Dense Closed Rectangles
5	15
10	8
15	8
20	8
25	6
30	3
35	1
40	0

Out of the 52 closed rectangles there are three closed rectangles viz. {[32,32,98,100],33}, {[32,32,99,100],34} and {[32,32,98,98],38} found which are dense rectangular regions having density value more than 30. The dense rectangle {[32,32,98,98],38} shows that if the temperature and humidity values are kept fixed at 32 and 98 respectively then 38 numbers of vegetables will be preserved. The other two solutions allow for some variation in temperature and humidity values, and larger rectangles with density values less than 30 may be chosen for this.

8.3. Results Applied on Car Dataset

The car interval dataset contains 33 car samples that are described by 8 interval features and one class level feature. The interval features are *Price*, *EngineCapacity*, *TopSpeed*, *Acceleration*, *Step*, *Length*, *Width*, and *Height* [24][25]. The samples are divided into 4 different class levels viz. *Utilitarian*, *Berlina*, *Sporting* and *Luxury* with 10, 8, 8 and 7 samples respectively. In this paper, two interval features *EngineCapacity* and *TopSpeed* features together are considered to assess the effectiveness of the proposed method where *EngineCapacity* represents X-interval and *TopSpeed* represents the Y-interval component of a rectangle. The outcome of the proposed method with various minimum density threshold values is shown in Table 7.

Table 7. No of dense closed rectangles with varying density thresholds, on Car data set

Density threshold, ρ (min)	No of Dense Closed Rectangles
1	243
2	220
3	181
4	135
5	90
6	55
7	29
8	9
9	0

The total number of closed rectangles generated is 243. The closed rectangle calculated for this dataset represents the number of car samples that share the same engine capacity and top speed range. For example, there are 76 such combinations of engine capacity and top speed range that are found in more than 4 car samples in the car dataset as shown in Table 7. Two such rectangular regions discovered are {[2171, 3199,226, 250],4}and {[2799, 3199,232,250],6}.

8.4. Results Applied on Synthetic Interval Dataset

The proposed method is tested on five synthetic datasets of varying sizes viz. *Syndataset1*, *Syndataset2*,*Syndataset3*, *Syndataset4*, *Syndataset5* and *Syndataset6*. The generation of synthetic datasets is discussed below.

8.4.1. Generation of Synthetic Dataset

The various synthetic datasetsused in this proposed algorithm *ACRMiner*isgenerated by using a synthetic data generator to generate the synthetic rectangle dataset. The input to the data generator is the number of transactions of rectangles (n) , frequency of rectangle trnsaction (f), maximum value of the left end point of the interval in x-domain ($l^x_{\max}$), maximum span of the intervals in x-domain ($span_x$), maximum value of the low end points of the intervals in y-domain ($l^y_{\max}$) and maximum span of the interval in y-domain ($span_y$). The span of an interval denotes the difference between thetwo points in a domain.In this synthetic data generator, a random number generator is used. This random number generator generates values for the left end poinst in the x-domain from 1 and $l^x_{\max}$. The span values generated for these intervalsarewhithin 1 to $span_x$. Similarly it generates values for low end points in the y-domain between 1 to $l^y_{\max}$ and the span of the interval between 1 and $span_y$. The synthetic datasets used in the porposed method are generated with $l^x_{\max}=200$, $span_x=100$, $l^y_{\max}=200$, $span_y=100$ and $f=1$.

This synthetic data generator uses a random number generator which gererates values for the left to generate the rectangle data set.

8.4.2. Results on Various Synthetic Dataset

The proposed method is applied to above five synthetic datasets. The number of closed rectangles computed and the time of execution of this method on these datasets are shown in Table 8.

Table 8. No of closed rectangles generated by the proposed method on various synthetic datasets

Dataset Name	Size of the Dataset	No of Closed Rectangles Generated	Execution Time
Syndataset1	1000	2624112	4.1767
Syndataset2	2000	4911136	14.247
Syndataset3	4000	14640928	62.318
Syndataset4	6000	19181947	141.2157
Syndataset5	8000	23687357	239.3524
Syndataset6	10000	25981481	351.8229

Figure 13, plots a graph of the proposed method's execution time versus dataset size. This demonstrates how the execution time of the proposed ACRMiner algorithm increases as the size of the input data increases.

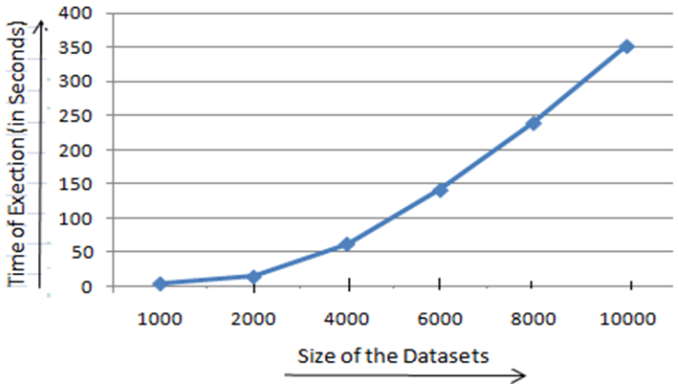

Figure 13. Time requirement of the proposed algorithm for varying size of synthetic dataset

8.4.3. Results of Varying Density Values on Synthetic Datasets

Because the density threshold value is crucial in deciding whether a region is dense or not, the proposed method is tested on various synthetic datasets to see how the output varies. To examine this variation, the synthetic datasets with the smallest and largest number of records, *Syndataset1* and *Syndataset6*, were used. The results of these experiments are shown in Table 9.

The results in Table 9, shows that the number of dense closed rectangles decreases as the density threshold increases, which is obvious.

Table 9. Result on synthetic dataset for different density thresold, ρ **(**min**)**

Dataset Name	Size of the Dataset	Density Threshold, ρ (min)	No of Closed Rectangles Generated
Syndataset1	1000	10	2603476
		20	2438929
		40	1476241
		60	397245
		80	40770
		100	57
		120	0
Syndataset6	10000	100	25564221
		200	23398811
		400	12888840
		600	1848229
		800	294230
		1000	794
		1200	0

The results in Table 9, shows that the number of dense closed rectangles decreases as the density threshold increases, which is obvious.

9. APPLICATIONS OF THE PROPOSE METHOD

The proposed method has application in a wide range of real-world problems. Some of the fields in which this method is useful are discussed in this section.

(1) *Meteorology:* Meteorology is the study of the atmospheric phenomena and the effects of weather on the atmosphere. There are various important factors such as rainfall, temperature, humidity, air quality, day time (sunrise and sunset time) etc. If two important parameters such as *temperature (max* and *min)* and *day time (sunrise* and *sunset time)* are selected for a particular location and period then it willreperesent a rectangular space. If the *tempature*and *day time* parameters are recorded for a number of days thenthere may be many overlapping rectangular regions available within the represented space. The algorithm proposed in this paper can be used to compute the density or suppor counts of these overlapping regions.From the density or support values of such regions one can find out suitable information about *temperature* and *day time* available on the majority of days for that location.This information may help tourists decide on the best time to visit this location.

(2) *Healthcare*: In healthcare, factors like blood sugar level, hemoglobin etc. contains crucial information about the helath condition of a patitient. For example, by considering factors like blood sugar level (lowest level and highest level) and hemoglobin count (max and min count) for patients suffering from diabetic, our proposed algorithm will provide information about blood sugar level and hemoglobin count common in most diabetic patients which may be useful for their effective treatments.

(3) *VLSI Design*: In VLSI design, placement of rectangular modules on a plane within a rectangle of minimum area without overlap is a crucial stage. The background of this process is the rectangle packing problem. The algorithm proposed in this paper can be used to find out the maximal overlapping regions of rectangles which will make VLSI design process faster.

(4) *Automatic Label Placement on Maps*: In cartography, label placement is a critical task in map production that requires a significant amount of manual work and time. Label placement is critical since it involves placing of labels in such a way that no two labelled map regions overlap. Too densely labelled maps are difficult to study and visualize. Finding the optimal placement of labels is expensive and the underlying problem is NP-Hard. In most algorithms the labels are placed initially by some procedure and after that the placements are improved by reducing the overlapping. The sizes of the labels may also be reduced. The algorithm proposed in this paper can be used in the initial placement of labels by extracting the overlapping rectangles. It can also be used to calculate the quality of a placement by calculating the total overlapping area.

(5) *Heatmap visualization*: Heatmap is a density-based data visualization technique used in big data visual analytics [30]. In the study of natural events such as storms, hurricanes, earthquakes etc., the affected areas are represented using bounding boxes or rectangles. Intensity values are calculated by the number of rectangles intersecting at a place. In such cases heatmaps are drawn to pictorially demonstrate the intensity of the event or magnitude of the phenomenon by using different colors i.e. different colors represent different intensity or magnitude values. Dark colors are usually used to represent high intensity values. Again, in many applications, the regions of interest are represented or approximated by rectangles. Different users have different regions of interest and then overlapping regions will indicate the regions that are of interest to many users. The algorithm proposed in this paper can be used for such

applications. Closed rectangles with high support values can be assigned dark colors. In other words, the darkness can be made proportional to the support values of the closed rectangles.

(6) *In Aquaculture Industry*: Water quality parameters that are commonly monitored in the aquaculture industry farming aquatic organisms include temperature, dissolved oxygen, pH, alkalinity, ammonia, and nitrites. Each organism has a range for these parameters which is conducive to their survival. If two important parameters are selected then the conductive ranges for the organisms are nothing but rectangles. If two rectangles corresponding to two organisms overlaps then the overlapping area will indicate the ranges of the parameters which are conductive for both the organisms. The algorithm proposed in this paper can be used for such applications. Closed rectangles with high support values will indicate ranges suitable for large number of organisms to survive.

10. CONCLUSION AND FUTURE WORK

In this paper we have proposed an incremental algorithm named as *ACRMiner* for computing all closed rectangles together with their support counts in a rectangle database and then based on a user defined threshold all the 2D closed regions are classified either as dense or sparse regions. The algorithm is incremental by nature and hence is applicable to dynamic data sets. The algorithm may generate duplicate closed rectangles and hence more space is needed. In case of duplicate closed rectangles, the rectangle with highest support will be considered. The proposed algorithm can correctly generate all closed rectangles together with their support counts from the given input dataset of rectangles using a tree like data structure called as *ACR-Tree*. The correctness of the algorithm follows from the theorems that have been proved. The effectiveness of the proposed algorithm is verified by testing on real-life datasets and synthetic data sets as well. Future work includes attempt to design a procedure so that generation of duplicates could be avoided. Future work also includes looking for methods to improve the performance of the proposed algorithm by using other suitable data structures.

REFERENCES

[1] A. E. Danganana nd, A. M. Sison, "An Improved Overlapping Clustering Algorithm to Detect Outlier", *Indonesian Journal of Electrical Engineering and Informatics*, vol. 6, no. 4, pp.401-409, 2018.

[2] U. Beyaztas, H. L. Shangand and Abdel-Salam G. Abdel-Salam, "Functional linear models for interval-valued data", *Communications in Statistics-Simulation and Computation*, vol. 51, no. 7, pp.3513-3532, 2020.

[3] A. Workman and J. J. Song, "Spatial analysis for interval-valued data", *Journal of Applied Statistics*, August 2023. (DOI: 10.1080/02664763.2023.2249636)

[4] R. Mavlyutovand and P. Cudre-Mauroux, "Managing Big Interval Data with CINTIA: the Checkpoint INTerval Array", *IEEE Transactions on Big Data*, vol. 7, no. 2, pp.285-298, June 2021.

[5] M. Izadikhah, R Roostaee and A. Emrouznejad, "Fuzzy Data Envelopment Analysis with Ordinal and Interval Data", *International Journal of Uncertainty Fuzziness and Knowledge-Based Systems*, vol. 29, no. 3, pp.385-410, June 2021.

[6] N. Sarma, "*Study and Design of Algorithms for Certain Problems in Interval Data Mining*". Doctoral Thesis. Department of Computer Science, Gauhati University, 2017.

[7] J. F. Allen, "Maintaining knowledge about temporal intervals", *Communications of the ACM*, vol. 26, no. 11, pp.832-843, 1983.

[8] N. J. Sarmah and A. K. Mahanta, "An efficient algorithm for mining closed frequent intervals", *International Journal of Knowledge Engineering and Data Mining*, vol. 5, no. 3, pp.222-240, 2018.

[9] A. K. Mahanta and M. Dutta, "Mining closed frequent intervals from interval data", *International. Journal of Applied Science and Advance Technology*, vol. 1, no. 1, pp.1-3, 2012.

[10] P. Papapetrou, G. Kollios, S. Sclaroff and D. Gunopulos, "Discovering frequent arrangements of temporal intervals", in *Fifth IEEE International Conference on Data Mining*, 2005. ICDM'05, pp. 354-361.

[11] D. Laskar, N. Sarmah and A.K. Mahanta, "SCI-Tree: An Incremental Algorithm for Computing Support Counts of all Closed Intervals from an Interval Dataset", *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 8 no. 9, pp. 233-242, 2019.

[12] I. Hazarika and A. K. Mahanta, "An Incremental Algorithm for Mining Closed Frequent Intervals", *In Advanced Computational and Communication Paradigms, Springer*, Singapore, pp. 63-73, 2018.

[13] N. J. Sarmah and A. K. Mahanta 2014, February. "An incremental approach for mining all closed intervals from an interval database", in*IEEE International Advance Computing Conference*, 2014. IACC'05, pp. 529-532.

[14] M. Dutta, M. Dutta and A. K. Mahanta, "Mining closed intervals in an interval database: An incremental method", in *IEEE International Conference on Electrical, Computer and Communication Technologies*, 2015. (ICECCT-2015) pp. 1-5.

[15] I. Hazarika, "*Study and Development of Data Mining Techniques for Classical and Symbolic Data*". Doctoral Thesis. Department of Computer Science, Gauhati Univesity.,2018.

[16] J. Lin, "Mining Maximal Frequent Intervals". 2003 *Proceedings of ACM symposium on Applied Computing*, 2003., pp. 426–431.
[17] D. I. Mazumdar, D.K. Bhattacharyya and M. Dutta, "Mining Minimal Infrequent Intervals", *Journal of Computer Science and Engineering*, vol. 179, no. 35, pp. 26-30, 2018.
[18] J. Edmonds, J. Gryz, D. Liang and R. J. Miller, "Mining for empty rectangles in large data sets". *Proceedings of the 8th International Conference on Database Theory*, United Kingdom (London), 2001., pp. 157-160.
[19] J. Edmonds, J. Gryz, D. Liang and R. J. Miller, "Mining for empty rectangles in large data sets", *Theoretical Computer Science*,. Vol. 296(3), pp. 435-452, 2003.
[20] L. P. Ku, B. Liu and W. Hsu, "*Discovering large empty maximal hyper-rectangle in multi-dimensional space*". National University of Singapore, Department of Information Systems and Computer Science, 1997.
[21] B. Liu, B., L.P. Ku and W. Hsu, W. "Discovering interesting holes in data", *Proceedings of Fifteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 1997. vol. 2, pp. 930–935.
[22] J. Lemley, F. Jagodzinski and R. Andonie, "Big holes in big data: A monte carlo algorithm for detecting large hyper-rectangles in high dimensional data". 2016, *IEEE 40th annual computer software and applications conference* (COMPSAC), vol. 1, pp. 563-571, June 2016.
[23] A. Datta and S. Soundaralakshmi, "An efficient algorithm for computing the maximum empty rectangle in three dimensions", *Information Sciences*, vol. 128 issue. (1-2), pp.43-65, 2000.
[24] HEDJAZI Lyamine. Accessed: Jun. 2019. [Online]. Available: http://lhedjazi.jimdo.com/useful-links
[25] F.D.A. De Carvalho, R. M. De Souza, M. Chavent and Y. Lechevallier, "Adaptive Hausdorff distances and dynamic clustering of symbolic interval data", *Pattern Recognition Letters*, vol. 27, no. 3, pp.167-179., 2006.
[26] https://extension.umaine.edu/publications/f wpcontent/uploads/sites/52/2015/04/4135.pdf
[27] N. Pasquier, Y. Bastide, R. Taouil and L. Lakhal, "Discovering frequent closed itemsets for association rules". *In International Conference on Database Theory*, 1999.pp. 398-416.
[28] I. Hazarika and A. K. Mahanta, "Mining maximal frequent rectangles". *Advances in Data Analysis and Classification*, vol. 16, issue 3, No. 5, pp. 593-616, 2022.
[29] J. Backer and J. M. Keil, "The bichromatic rectangle problem in high dimensions", in *21st Canadian Conference on Computational Geometry*, 2009. pp. 157–160.
[30] N E. Costa and T. Pechlivanoglou and M. Papageliset, "*OL-HeatMap: Effective Density Visualization of Multiple Overlapping Rectangles*", Big Data Research, vol. 25, article 100235, 2021, pp. 1-12.
[31] M. Bozorgi et al. "A Time-Efficient and Exploratory Algorithm for the Rectangle Packing Problem", *Intelligent Automation and Soft Computing*, vol. 31, no. 2, pp.885-898, 2022.
[32] A. Bedboudi, C. Bouras and Mohmed T. Kimour, "A Heterogeneous Population-Based Genetic Algorithm for Data Clustering", *Indonesian Journal of Electrical Engineering and Informatics*, vol. 5, no.3, pp.275-284, 2017.

BIOGRAPHY OF AUTHORS

DwipenLaskar, is a PhD scholar in the department of Computer Science, Gauhati University. He completed B.Sc. degree in 2002 from Gauhati University in Assam. He obtained the degree of M.Sc. in Computer Science from Gauhati University in 2005. In 2008, he completed his M.Tech. in information technology at Tezpur University in Assam.
Presently he is also working as an Assistant Professor in the Department of Computer Science in the Gauhati University since 2015. He also worked as an Assistant Professor in GIMT, Assam (Presently known as GirijanadaChowdhury University) from 2008 to 2015. His area of interest in research includes Data Mining, Machine Learning and Interval Data Mining and Pattern Recognition.

AnjanaKakotiMahanta, is presently working as Professor in department of Computer Science, Gauhati University. She obtained PhD in Computer Science in 1990. She visited the University of Warsaw for three months in 2007 as part of a bilateral exchange programme between the Polish Academy of Sciences and the Indian National Science Academy (INSA), where she worked on research projects with faculty members from the department of computer science. Her current area of research includes Algorithms and Data Mining. In her 36 years of teaching experience, she has guided and awarded 13 scholars. She had more than 73 research publications in various international, national journals and conferences.