



An Efficient Algorithm for Finding Dense Regions for Mining Quantitative Association Rules

WANG LIAN

Faculty of Information Technology, Macao University of Science and Technology
Avenida Wai Long, Taipa, Macao
lwang@must.edu.mo

DAVID W. CHEUNG AND S. M. YIU

Department of Computer Science, The University of Hong Kong
Pokfulam Road, Hong Kong
<dcheung><smyi>@cs.hku.hk

(Received February 2004; accepted March 2005)

Abstract—Many algorithms have been proposed for mining boolean association rules. However, very little work has been done in mining quantitative association rules. Although we can transform quantitative attributes into boolean attributes, this approach is not effective and is difficult to scale up for high-dimensional cases and also may result in many imprecise association rules. Newly designed algorithms for quantitative association rules still are persecuted by the problems of nonscalability and noise. In this paper, an efficient algorithm, DRMiner, is proposed. By using the notion of “density” to capture the characteristics of quantitative attributes and an efficient procedure to locate the “dense regions”, DRMiner not only can solve the problems of previous approaches, but also can scale up well for high-dimensional cases. Evaluations on DRMiner have been performed using synthetic databases. The results show that DRMiner is effective and can scale up quite linearly with the increasing number of attributes. © 2005 Elsevier Ltd. All rights reserved.

Keywords—Data mining, Quantitative association rules, Dense regions, Density measure, Algorithms

1. INTRODUCTION

Data mining, the effective discovery of correlations among the underlying data in large databases, has been recognized as an important area for database research and has also attracted a lot of attention from the industry as it has many applications in marketing, financial, and retail sectors. One commonly used representation to describe these correlations is called *association rules* as introduced in [1]. In this model, the set $I = \{i_1, i_2, \dots, i_m\}$ is a collection of items or attributes. The database DB consists of a set of transactions, where each transaction is a subset of items in I . An association rule is an implication of the form $X \Rightarrow Y$ with $X, Y \subseteq I$ and $X \cap Y = \emptyset$.

The research of the second author is supported by a grant from the Research Grant Council of Hong Kong. The authors would like to thank the anonymous reviewers for their very useful comments and insights which enhance the readability of the paper and enlighten us with some interesting future directions for the research.

The meaning of the rule is that a transaction containing items in X will likely contain items in Y so that marketing strategy can be derived from this implication, for instance. To determine whether an association rule is interesting, two measures are being used: *support* and *confidence*. An association rule, $X \Rightarrow Y$, has support $s\%$ in DB if $s\%$ of transactions in DB contain items in $X \cup Y$. The same association rule is said to have confidence $c\%$ if among the transactions containing items in X , there are $c\%$ of them containing also items in Y . So, the problem is to find all association rules which satisfy predefined minimum support and minimum confidence constraints.

In this setting, attributes which represent the items are assumed to have only two values and thus are referred as *Boolean* attributes. If an item is contained in a transaction, the corresponding attribute value will be 1, otherwise the value will be 0. Many interesting and efficient algorithms have been proposed for mining association rules for these Boolean attributes, for examples, *a priori* [1], DHP [2], and partition algorithms [3] (see also [4–8]). However, in a real database, attributes can be *quantitative* and the corresponding domains can have multiple values or a continuous range of values, for examples, age, salary. By considering this type of attributes, association rules like this one, $(30 \leq \text{age} \leq 39)$ and $(50000 \leq \text{salary} \leq 79999) \Rightarrow (100000 \leq \text{loan} \leq 300000)$, will be desirable. To handle these quantitative attributes, in this paper, a new measure called *density* will be introduced. This new measure, together with the support and confidence measures, will lead to an efficient and scalable algorithm, DRMiner, for mining quantitative association rules.

1.1. Motivation for a Density Measure

The motivation for a new *density* measure can best be illustrated by an example. Assuming that we have two quantitative attributes, A and B (see Figure 1). Each transaction in the database is mapped to a data point in this two-dimensional space using the corresponding values of the attributes as coordinates. Each unit in the grid of Figure 1 is called a cell. We want to find all the association rules of the form $A \subseteq [x_1, x_2] \Rightarrow B \subseteq [y_1, y_2]$, where $x_1, x_2 \in \{0, a1, a2, a3, a4, a5, a6\}$ with $x_2 > x_1$ and $y_1, y_2 \in \{0, b1, b2, b3, b4, b5\}$ with $y_2 > y_1$. And we further set the support threshold to five points and the confidence threshold to 50%.

We can obviously obtain the following rules:

- $A \subseteq [a1, a2] \Rightarrow B \subseteq [b2, b3]$ (1),
- $A \subseteq [a2, a5] \Rightarrow B \subseteq [0, b5]$ (2),
- $A \subseteq [0, a6] \Rightarrow B \subseteq [0, b5]$ (3).

One can easily see that with only the support and confidence measures, as long as a range has the minimum support, any larger range containing this range will also satisfy the support measure

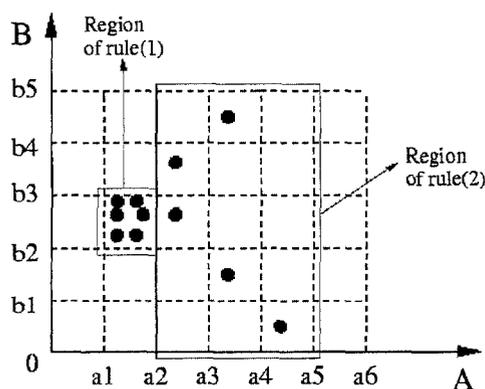


Figure 1 Example of some quantitative rules.

threshold. Similarly, by enlarging the two ranges in both dimensions, it is very *likely* that the new ranges will satisfy the confidence constraint as well. This can lead to many “useless” rules.

- *Trivial Rule:* Rules (2) and (3) will be considered as useless and not interesting because Rule (3) covers all possible values of both attributes while Rule (2) covers almost all possible values of both attributes.
- *Redundant Rule* According to the above observation, from Rule (1), we can have all kinds of rules in the form $A \subseteq [z1, z2] \Rightarrow B \subseteq [u1, u2]$ where $[a1, a2] \subseteq [z1, z2]$ and $[b2, b3] \subseteq [u1, u2]$. In this particular example, we can see that all these rules satisfy both the support and confidence constraints. However, these rules, when compared with Rule (1), are not very interesting because the increase in support of these rules is relatively a lot less than the increase in the sizes of the ranges.

From the above example, intuitively, Rule (1) is much more preferable than both Rules (2) and (3). The reason is that the density of the region representing Rule (1) is much higher than the density of the regions representing Rules (2) and (3) (see Figure 1). Hence, if density is defined as a new measure, it is easy to get rid of the trivial and redundant rules.

In a real application, when we map a database to a multidimensional space, we can notice that the data points (transactions) exhibit a “dense-regions-in-sparse-regions” property. In other words, the space is sparse but not uniformly so. That is, the data points are not distributed evenly throughout the multidimensional space. According to this kind of distribution and the density measure we have just introduced, the problem of mining quantitative association rules can be transformed to the problem of finding regions with enough density (*dense regions*) and then map these dense regions to quantitative association rules. Note that generating association rules from dense regions is straight-forward. For example, if a dense region on dimensions $A[a1, a2]$, $B[b1, b2]$, $C[c1, c2]$ is identified, then rules like “ $A[a1, a2]$ and $B[b1, b2] \Rightarrow C[c1, c2]$ ” or “ $B[b1, b2]$ and $C[c1, c2] \Rightarrow A[a1, a2]$ ” can be generated easily. So, in this paper, we mainly focus on efficiently discovering dense regions.

Our dense region discovery is a three-step process

- (1) We first map all database transactions into a high-dimensional space, which is partitioned into cells, and use a k - d tree to hold all non-empty cells in the space. In this step, the size of leaf node is carefully controlled so as to ensure that no dense region can be entirely located in one leaf node, therefore one leaf node can only contain one part of a dense region and that part must touch the boundaries of the leaf node. This property will facilitate the discovery of dense regions in the later steps.
- (2) For each leaf node, we may have a region that covers a possible part of a dense region (we call it a *dense region cover*). In this step, we identify and self merge the dense region cover set. The purpose of this merging is to make sure that dense regions are entirely located in this merged cover set.
- (3) Based on the merged cover set, we then identify the dense regions.

1.2. Related Work

It is natural to consider dense region discovering as a clusterization problem [9]. There are many existing clustering algorithms that base on different techniques or models. For examples, [10–12] are density-based; [13] is cell or grid-based; [14,15] use randomized sampling; and [16] combines classical partition- and density-based techniques. However, general clustering algorithms may not be very appropriate for solving our problem. In general, most of the above algorithms are designed to discover clusters with arbitrary shapes, however, what we need is hyperrectangular clusters for generating sensible association rules. So, these methods are not directly applicable.

Although hyperrectangular-shape clusters can be obtained by using specific distance measure such as the Manhattan distance, the density of the clusters cannot be guaranteed. If the region of the minimum bounding box of a cluster does not satisfy the density threshold, further processing is

required. One possibility is to shrink the bounding box on some dimensions. However, shrinking may not be able to achieve the required density. Another approach is to split and recluster the clusters by using some recursive clustering techniques. Besides the high cost of performing recursive clustering, splitting could break the dense clusters into a large number of small clusters, which eventually would trigger costly merges. Finally, absorbing a few sparse points during the clusterization process could disturb tremendously the density of the found clusters. The problem is that a clustering algorithm inherently does not distinguish the points that belong to some dense regions from the points in the sparse regions. The only exception is [14] that produces similar clusters as our method does, but its time complexity is exponential in the number of dimensions, so is not practical in general as usually the practical cases have high dimensions. Moreover, most clustering algorithms require the number of clusters that need to be found as input, however, this number is not available in this case.

Some others tried to apply the image analysis techniques to solve the problem. However, the number of dimensions that can be handled is restricted to two, or at most three, while it is easy to get more than four in real applications. Decision tree classifier is another approach, however, due to the problem of efficiency, it does not provide an effective solution. For example, the SPRINT classifier [17] generates a large number of temporary files during the classification process. This causes numerous I/O operations and demands large disk space. To make things worse, every classification along a specific dimension may cause a splitting on some dense regions resulting in serious fragmentation of the dense regions. Costly merges are then needed to remove the fragmentation. In addition, many decision tree classifiers cannot handle large data set because they require all or a large portion of the data set to reside permanently in memory.

There are a few other works trying to solve this mining problem for quantitative attributes. In [18], the authors proposed an algorithm which is an adaptation of the *a priori* algorithm for quantitative attributes. It partitions each quantitative attribute into consecutive intervals using *equi-depth* bins. Then adjacent intervals may be combined to form new intervals in a controlled manner. From these intervals, *frequent itemsets* (c.f. *large itemsets* in *a priori* algorithm) will then be identified. Association rules will be generated accordingly. The problems with this approach is that the number of possible interval combinations grows exponentially as the number of quantitative attributes increases, so it is not easy to extend the algorithm to higher-dimensional cases. Besides, the set of rules generated may consist of redundant rules for which they present a "greater-than-expected-value" interest measure to identify the interesting ones.

Lent *et al.* also proposed an algorithm for mining quantitative attributes [19]. Their idea is to combine similar association rules to form interesting quantitative association rules using the technique of clustering. The algorithm will map the whole database into a two-dimensional array with each entry representing an interval in each of the dimensions. Entries with enough support will be marked, then a greedy clustering algorithm using "bitwise AND" operation is used to locate the clusters. The drawback is that the algorithm is sensitive to noise. Although an image processing technique, called *low-pass filter*, is used to remove these noises, the algorithm is still sensitive to noise and noise is unavoidable in a real database. Also, the algorithm is basically designed for two quantitative attributes, so again it is not trivial how to extend the algorithm to an efficient one for higher-dimensional cases. There are other approaches that solve some special cases of mining quantitative association rules [20–22]. For examples, [20,21] only allow one quantitative attribute to appear on the right-hand side of the association rule while [22] can only discover rules with two quantitative attributes on the left-hand side of the rule.

The noise problem and the problem of redundant rules of these approaches can be handled by the density measure in our approach. Also, our DRMiner does work on the general case of mining quantitative association rules and can be used in higher-dimensional cases with scalable performance. It is hoped that this new approach can give more insights on this mining problem. The rest of the paper is organized as follows. Some preliminary definitions will be given in Section 2. Section 3 will describe the algorithm for discovering dense regions. Section 4 shows

the time and space complexity of our method. Section 5 evaluates our method by synthetic data, and Section 6 concludes the paper.

2. SOME PRELIMINARY DEFINITIONS

In the *a priori* algorithm, two steps are used to produce association rules. In Step 1, frequent itemsets are identified based on the support threshold. Then, in Step 2, association rules are generated from those frequent itemsets that also satisfy the confidence threshold. In a similar manner, we also produce meaningful quantitative association rules in two steps. In Step 1, hyperrectangular regions (to be defined below) are generated with respect to both support and density thresholds. In Step 2, among the regions identified in Step 1, we select those regions that also satisfy the confidence threshold. Then, association rules are generated from the selected hyperrectangular regions. Step 2 is trivial, so we focus on Step 1. To complete Step 1, there are two available strategies on how to use the density measure. The first approach is to generate all possible hyperrectangles based only on the support threshold. Then, use the density threshold to do the pruning. The other alternative is to adopt a new approach, which use both support and density thresholds in the process of generating hyperrectangles. In fact, the first approach is inefficient and even inapplicable, because it has to generate a huge amount of useless hyperrectangles for pruning, and because of the fact that the number of range combination grows exponentially with the number of dimensions and the number of partitions of each dimension, so in many cases, it is impossible to get all possible hyperrectangles. On the contrary, the second approach is more efficient, because the pruning is carried out as the hyperrectangles generation is processed. Note that the hyperrectangles identified in this step will be checked against the confidence threshold in Step 2. Before we discuss the details, we first give some useful definitions.

DEFINITION 1. A quantitative association rule is represented as follows: $A_1 \subseteq [a_1, b_1] \wedge A_2 \subseteq [a_2, b_2] \wedge \dots \wedge A_{n-1} \subseteq [a_{n-1}, b_{n-1}] \Rightarrow A_n \subseteq [a_n, b_n]$ ¹ where $A_i (1 \leq i \leq n)$ is a quantitative attribute. And $[a_i, b_i]$ is a range on the attribute A_i and $a_i \leq b_i$.

A quantitative attribute usually contains values on a continuous range, partitioning provides a useful way to manage these values and to speed up the processing. To start with, similar to other grid-based approaches, we first partition the range of each quantitative attribute into subranges. The d -dimensional space which is composed by attributes A_1, A_2, \dots, A_d can then be regarded as a set of small regions which we call them cells. The partitioning is to facilitate a data structure (k - d tree) to store information on the useful regions for further processing. Furthermore, we are working on a high-dimensional space rather than the original database, so from now on we will refer "dimension i " as attribute i in the following context.

DEFINITION 2. Given a relational database with d attributes, and all its transactions mapped into a d -dimensional space such that each transaction is a data point in this d -dimensional space with its i^{th} ($1 \leq i \leq d$) attribute value be the index on i^{th} dimension, a i th hyperrectangular region r is defined by d ranges, namely $[a_1, b_1], [a_2, b_2], \dots, [a_d, b_d]$ where $[a_i, b_i]$ is a range on the i^{th} attribute with $a_i \leq b_i$ for all $1 \leq i \leq d$. A hyperrectangular region with volume $v_r =$ the number of cells in the region, density $\rho_r = (\text{number of points})/v_r$ is called a dense region if:

- (1) $\rho_r \geq \rho_{\min}$ (a user specified density threshold).
- (2) each cell in r has its density $\geq \rho_{\text{low}}$ (a user specified density threshold and $\rho_{\min} > \rho_{\text{low}}$).

In Definition 2, Condition (1) is intuitive. For Condition (2), let us recall Figure 1, suppose ρ_{\min} is two points in one cell. It is clear that $([a_1, a_2], [b_2, b_3])$ is a dense region and $([a_1, a_2], [0, b_3]), ([a_1, a_2], [b_1, b_4]), ([a_1, a_2], [b_2, b_5]) \dots$ are also dense regions. They are all regions containing both $([a_1, a_2], [b_2, b_3])$ and some empty cells. In other words, a region with high

¹It is easy to extend this definition to include more than one attribute on the right-hand side of the rule.

density can swallow empty (or sparse cells) to produce lots of dense regions. This contradicts to our original purpose. Therefore we introduce Condition (2), that is, we do not want high density regions to swallow cells with very low density.

REMARK. Note that if the density of a region is a lot higher than ρ_{\min} , it is still possible for a high density region to swallow some cells with smaller density (which is higher than ρ_{low}). One quick solution is to prune back the border areas that have density which is relatively a lot lower than the density of the core region. Also, it is recommended that the values of ρ_{\min} and ρ_{low} should not be set to have a big difference in real applications.

3. DISCOVERING DENSE REGIONS

Recall that in our approach, finding dense regions is decomposed into three steps, in this section we will first formulate the dense region discovery problem as an optimization problem. Then, we show the details of the three steps.

3.1. Problem Statement of Discovering Dense Regions

We formulate the dense region discovering problem as follows. Let $S = D_1 \times D_2 \times \dots \times D_d$ be a d -dimensional space such that, for $1 \leq i \leq d$, $D_i = \{x \mid x \in A_i, L_i \leq x < H_i\}$ is a range in a totally ordered domain A_i , bounded above and below by H_i and L_i , respectively. This space contains a set of data points $D = \{v_1, v_2, \dots, v_n\}$, where D is a subset of S . The space S is partitioned into equal size cells such that the *cell length* on the i^{th} dimension is c_i . That is, the i^{th} dimension is divided into $cn_i = (H_i - L_i)/c_i$ equal intervals. We use $\text{CP} = \langle c_1, c_2, \dots, c_d \rangle$ to denote a *cell-based partition* of S . We use cell as the basic unit to reference the coordinates of regions. We use $r = [(l_1, l_2, \dots, l_d), (h_1, h_2, \dots, h_d)]$ to denote a region representing a subspace whose projection on the i -dimension is the interval $[L_i + c_i l_i, L_i + c_i h_i]$. The volume of a region is the total number of data points that the region can hold. The density of a region r is the number of data points that fall in r divided by the volume of r . We denote the volume of a region r by v_r , and its density ρ_r . And ρ_{\min} is the density requirement for dense regions. Because the final rules we mined should satisfy the support threshold, so the volume of a dense region corresponding to a rule should also be large enough to satisfy the support threshold. Hence we give v_{\min} as the volume threshold a dense region should satisfy. ρ_{low} is another density threshold that each cell should satisfy in a dense region. This is because we do not want to combine empty or near empty cells in a dense region. Let sup_{\min} and N be the support threshold and the total number of points, respectively. Then, we require that $v_{\min} \geq \text{sup}_{\min} * D$.

Given a d -dimensional space S , a set of data points D in S , a cell based partition CP on S , together with three input thresholds ρ_{\min} , ρ_{low} , and v_{\min} , discovering the dense regions in S is to solve the following optimization problem.

Table 1. Problem statement of discovering dense regions.

Objective	To find a set of dense regions (r_1, r_2, \dots, r_n) such that $\sum v_{r_i}$ is maximized
Constraints	$\rho_{r_i} \geq \rho_{\min}, (i = 1, \dots, n)$ $v_{r_i} \geq v_{\min}, (i = 1, \dots, n)$ all cells cl in v_{r_i} have $\rho_{\text{cl}} \geq \rho_{\text{low}}, (i = 1, \dots, n)$.

3.2. The DRMiner Algorithm for Discovering Dense Regions

In this section, we describe an efficient heuristic-based algorithm to locate the dense regions. Note that our algorithm does not guarantee an optimal solution to the optimization problem in Section 3.1, but is efficient enough to locate dense regions for the purpose of identifying quantitative association rules.

A dense region is a connected set of cells each of which has a density higher than ρ_{low} . In the following, we will call this type of cells *admissible cells*. Among the admissible cells, those

that have density higher than ρ_{\min} are called *dense cells*. Also, any cell which is not empty is called a *valid cell*. Hence, a dense region must be a connected set of admissible cells. One way to discover dense regions is to use a multidimensional array to represent all the cells in the space and store the number of data points of each cell in the array. Then the whole space can be traversed along all possible directions to locate and grow dense regions. This is very inefficient, the array in general would be very large and could not be stored in the memory. Also, many cells in the array are empty and contribute nothing to the mining of dense regions. Another approach is to use a tree-like data structure to store and index all the valid cells.² This would require much less memory space; however, locating neighboring cells in a dense region may need to traverse many nodes on the tree. So, neither of these two approaches is an ideal solution. We have integrated these two approaches into the much more efficient algorithm DRMiner.

The following are the three main steps of the DRMiner.

- (1) A cell-based k - d tree is built to store the valid cells in the space together with the number of points in each cell. We have observed that if the leaf nodes in the tree are small enough, then every dense region must touch some boundaries of some leaf nodes. Therefore, dense regions can be grown from the boundaries of the leaf nodes.
- (2) A *dense region cover set* (a set with several dense regions) is then grown inside some leaf nodes from their boundaries, and then this cover set is self merged across boundaries. Subsequent search of the dense regions is restricted in this merged cover set. *Most importantly, the size of this merged cover set will be in the order of that of the dense regions and hence is much smaller than the whole space.*
- (3) The cells in each cover can then be traversed to find out the exact dense regions in the cover. Note that most leaf nodes corresponding to sparse regions would not be involved in the covers grown in Step (2). This effectively prune away most of the sparse regions in the searching in Step (3) above

The algorithm DRMiner has two important merits.

- (a) it can identify very efficiently a set of small subspaces, the cover set, for finding the dense regions;
- (b) the searching is limited in each cover separately;

there is no need to traverse between covers. In the following, we will explain the techniques in DRMiner in details.

Step 1. Build k - d Tree

We build a k - d tree to store the valid cells in the space S . For every point in the set of data points D , the cell which it belongs to will be inserted on the tree. Besides the cell coordinates, the tree also keeps track of the number of points in each cell. A k - d tree is very suitable for our dense region discovering problem, because every node splitting is done along one dimension. Hence, the resulted nodes are in fact rectangular regions in the space such that the union of all the leaf nodes covers the whole space. The following theorem defines the splitting criterion on our k - d tree.

THEOREM 1. *Let v_{\min} be the minimum volume of a dense region, and v_{cl} be the volume of a cell. Let R be a hyperrectangle in S . If the number of admissible cells in R is less than v_{\min}/v_{cl} , then no dense region can be completely contained in R .*

PROOF. Since the volume of a dense region must be larger than v_{\min} , it must contain at least v_{\min}/v_{cl} admissible cells. Hence no dense region can be completely contained in R .

Following Theorem 1, we split a leaf node in the k - d tree whenever it has more than v_{\min}/v_{cl} admissible cells By doing that, we can guarantee that a dense region will always touch some

²This index is built on valid rather than admissible cells just because we not only use this index for discovering dense region, but also use it to support generating rules, so all points must be in this index.

boundaries of some leaf node, i.e., it can never be contained in a single node without touching any boundary. Figure 2 shows the case that a dense region is split across four nodes on a k - d tree. Note that we follow the standard routine of a general k - d tree to choose the splitting attribute. That is, we randomly give an order to all attributes, then follow this order to do the splitting one by one. If the last attribute is reached, we may start a new round.

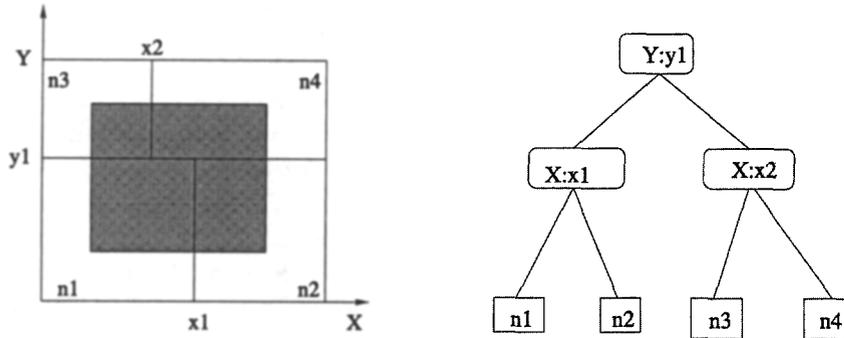


Figure 2. Dense regions split across boundaries

Step 2. Grow and Self Merge Dense Region Cover Set on Boundaries

Let R be the region associated with a leaf node of the k - d tree which has been built previously to store the valid cells. Let A be the set of admissible cells in R . (In the following, when we say a dense region in R , we mean the part of a dense region that falls in R .) If the minimum bounding box $MBOX(A)$ of A does not touch any boundary of R , then according to Theorem 1, R cannot contain any dense region, and hence can be ignored in the mining of dense regions. On the other hand, if this is not the case, then we will grow covers from the boundary to contain the dense regions in R .

Suppose $MBOX(A)$ does intersect with some boundaries of R . Let k be a boundary (a $d - 1$ -dimension hyperplane) of R . Let $B \subseteq A$ be the set of admissible cells touching k . Let P be the projection of $MBOX(B)$ on k . It is straight forward to see that the projection on k of any dense region in R which touches k will be contained in P . Let X be the axis perpendicular to k . (Note that X has been divided into intervals by the cell partition.) Let $F = \{I \mid I \text{ is an interval on } X, \exists \text{ a cell } c \in A - B, \text{ such that the projection of } c \text{ on } X \text{ is } I\}$. Let T be the maximal connected set of intervals in F which touches the boundary k . (The existence of T is guaranteed by B since it touches k .) The region $P \times T$ is called the *dense region cover* grown from k in R .

THEOREM 2. *Let C be a dense region cover grown from a boundary k in a region R . If r is a dense region in R which touches k , then $r \cap R \subseteq C$.*

PROOF. It follows directly from the description of dense region cover.

Figure 3 is an example of finding dense region cover in a leaf node. The filled cells are the admissible cells in the node. Assume the boundary (X, x_1) is the boundary from which the cover will be grown. P is the projection on the boundary, and T is the maximal connected set of intervals touching the boundary. $P \times T$ is the cover from the boundary (X, x_1) . Note that the cover contains all dense regions which touch the boundary (X, x_1) .

Figure 4 is the procedure `GrowDenseRegCover` used to compute the dense region cover set in a leaf node of the k - d tree. In line 8 of `GrowDenseRegCover`, the procedure `CellOnBoundary` returns the admissible cells in A that touch the boundary k and stores them in B . If B is not empty, then a dense region cover will be grown from k in line 10 with the procedure `GrowCover` according to the definition of dense region cover. After a cover has been grown, all admissible cells in the cover will be removed from the set of admissible cell A . Before the procedure is repeated

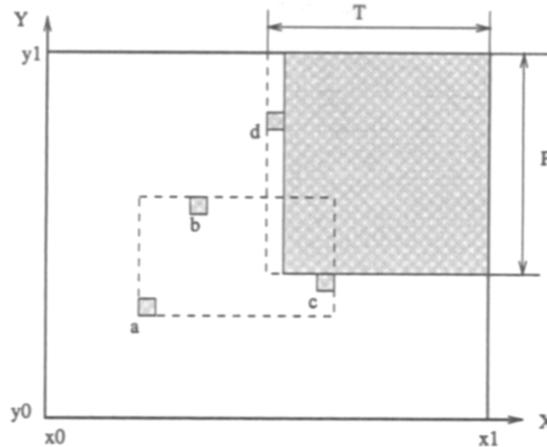


Figure 3 Finding dense region cover in the leaf node.

on another boundary, the minimum bounding box of the remaining admissible cells not covered yet will be tested against all boundaries in Step (6). If it does not touch any boundary, then no more cover is needed; and the procedure returns the found covers. This is also illustrated in Figure 3: once the cover $P \times T$ is removed, the MBOC of the remaining admissible cells, (cells a , b , c), would not touch any boundary; hence, no more cover is needed.

```

/* PROCEDURE : GrowDenseRegCover */
/* Input : N: a leaf node of k-d tree, with a set of valid cells */
/* Output. Dc. dense region cover set in N */
1) A = { admissible cells in N },
2) let R be the region associated with N,
3)  $\Theta$  = set the boundaries of R,
4) Dc =  $\emptyset$ ,
5) for each boundary  $k \in \Theta$ , do {
6)   if MBOC(A) does not touch any boundary in  $\Theta$  then
7)     return Dc, /* dense region cover set have been found */
8)   B = CellOnBoundary(A, k), /* B contains all cells in A that touch k */
9)   if (B  $\neq \emptyset$ ) then {
10)    D = GrowCover(A, B, R, k); /* grow cover from k */
11)    Dc = Dc  $\cup$  {D}; /* insert the cover found into Dc */
12)    A = A - {c | c  $\in$  D}; /* remove cells in D from A */
13)    if (A =  $\emptyset$ ) then return Dc,
14)   }
15) }

```

Figure 4 The procedure of GrowDenseRegCover

Merge dense region cover set

Since the k -d tree may split a dense region across several nodes, the dense region cover set of the leaf nodes need to be merged at the split boundary. For example, in Figure 2, the dense region has been separated into four pieces at the split positions of the k -d tree. After the dense region cover sets in n_1 and n_2 have been found, they will be merged along the split position at $X = x_1$. The result of merging will be attached to the nonleaf nodes at $(X : x_1)$. Subsequently, it will be merged further with the cover sets from n_3 and n_4 , and the resulting cover of the whole dense region will be attached to the node at $(Y : y_1)$.

When merging cover set from two sibling nodes on the k - d tree, each node may have more than one cover touching the same boundary as a result of previous merging. In that case, we will merge any two from the two nodes which touch each other on the boundary and the cover will be extended to their minimum bounding box. This merging will be performed recursively until no more merging can be done between the two nodes. This merging procedure will guarantee that a dense region will not be divided up into different cover set.

The DRMiner algorithm

We have described the first two steps of DRMiner. In Figure 5, we presented the whole algorithm.

DRMiner has three main steps. The first step (lines 1–4) is to read data points from D and build the k - d tree to store all valid cells. The second step (lines 5–8) is to grow and merge the dense region cover set from the nodes on the tree. In general, we can assume that all the valid cells can be counted in the memory, because the number of cells is much smaller than the number of data points. We will have to use chunking to handle the cells if the memory is not enough. This will be discussed in Section 4. In the third step (lines 9–12), we search for dense regions within each cover found. We use an array to store all the cells in a cover and use a greedy procedure FindDenseRegion to scan the cover and grow dense regions from the array. Since the cover set has volumes in the same order as the dense regions they contain, it is much smaller than the volume of the whole space. Hence, in general, we can assume that the array storing the cells in a cover can be built in the memory. Again, we discuss how to handle the case of not enough memory in next section. What remains to be discussed is the procedure FindDenseRegion.

```

/* Input: S. whole space; D. data points, CP: cell based partition;
     $\rho_{\min}$ ,  $\rho_{\text{low}}$ ,  $v_{\min}$ : thresholds,
     $O(D'_1, D'_2, \dots, D'_d)$ : dimension order.
/* Output: dense regions in S. */
/* Step 1: build the  $k$ - $d$  tree */
1)  $Tr$  = Initialize the  $k$ - $d$  tree,
2) for each point  $p \in D$  do {
3)   if the cell  $c$  containing  $p$  has not been inserted on  $Tr$ , then insert  $c$  in  $Tr$ ;
4)   increase the count of  $c$  by 1 ,
    /* leaf nodes of  $Tr$  are split according to the criterion defined in Theorem 1; */
}
/* Step 2: grow dense region cover set on the  $k$ - $d$  tree */
5) for every leaf node of  $N$  of  $Tr$  do
6)   GrowDenseRegCover( $N$ ),
    /* grow dense region cover set in  $N$  */
7) for every non-leaf node of  $Tr$ , merge the dense region cover set of its children;
8) assign the resulted dense region cover set to  $DC$ ,
/* Step 3. search dense regions in the dense region cover set in  $DC$  */
9)  $Dr = \emptyset$ ,
10) for each dense region cover  $dc \in DC$  do {
11)    $dr = \text{FindDenseRegion}(dc, O)$ ; /* find dense region in  $dc$ ,  $O$  is the dimension order */
12)    $Dr = Dr \cup dr$ ,
13) }
14) return  $Dr$ ,

```

Figure 5 DRMiner.

```

/* Procedure · FindDenseRegion */
/* Input. C: a dense region cover ,
      O. selected dimension order  $O(D'_1, D'_2, \dots, D'_d)$  */
/* Output. DR . set of dense regions in C */
1) build an array A to store all the cells in C,
2)  $DR = \emptyset$ ,
3) for each cell c in A scanned in the order O, do{
4)   if (c is not inside any dense region in DR and  $\rho_{cl} \geq \rho_{min}$ ) then {
5)      $r = \text{GrowRegion}(c)$ ; /* grow dense region r from the seed c */
6)     if  $V(r) \geq V_{min}$  then
7)        $DR = DR \cup \{r\}$ ,
8)     }
9) }
10) return DR,

/* Procedure · GrowRegion */
/* Input. c a seed ;  $O(D'_1, D'_2, \dots, D'_d)$ . dimension order,
/* Output. r : a dense region in C */
1)  $r := c$ ;
2) repeat{
3)   for k from  $D'_1$  to  $D'_d$  do{ /* k is the dimension index */
4)     dir = 1 ; /* first grow on the positive direction */
5)     repeat{
6)       repeat{
7)          $\delta := \delta(r, k, \text{dir})$ , /* get the increment */
8)         if (all cells in  $\delta$  are admissible and  $\rho(r \cup \delta) \geq \rho_{min}$  )
9)           then  $r = r \cup \delta$ , /* add the increment to the dense region */
11)        } until (r cannot be expanded anymore),
12)        dir := dir - 2, /* grow on the negative direction (dir = -1) */
13)      }until dir < -1;
14)    } /* for loop end */
15) }until (r does not change),
16) return r;

```

Figure 6 Procedure of FindDenseRegion.

Step 3. Find Dense Regions (Procedure FindDenseRegion)

Suppose C is a dense region cover. We store all the cells in C (including cells with no data point) in a multidimensional array so that we can scan the cells in C to search for dense regions. The order of scanning in C is determined by a preselected dimension ordering $O(D'_1, D'_2, \dots, D'_d)$, i.e., first on dimension D'_1 , then on D'_2 , etc. During the scanning, FindDenseRegion first locates a *seed cell*, which is a dense cell, then uses the seed to grow a maximal dense region along the dimension order. After a dense region is found, FindDenseRegion repeats the searching in the reminding cells of C until all cells have been scanned. Figure 6 is the procedure FindDenseRegion.

In line 5 of FindDenseRegion, the procedure GrowRegion is called to grow a dense region from a seed. It grows the region in the same order as the scan order: first in the positive direction of D'_1 ; then in the negative direction of D'_1 ; then in the positive direction of D'_2 ; etc.; until all dimensions and directions are examined. It iterates this growing process on all directions until no expansion can be found on any dimension. In the first dimension, GrowRegion grows a dense region by adding cells to the seed. Once after the first dimension, it grows by adding trunks of cells to the seed. We call the trunk of cells added to the dense region in each step an *increment*. If a

dense region $r = [(a_1, a_2, \dots, a_d), (b_1, b_2, \dots, b_d)]$ grows into the positive direction of dimension k , we denote the *increment* by $\delta(r, k, 1)$. Similar, the increment of r on the negative direction of dimension k is denoted by $\delta(r, k, -1)$. GrowRegion determines the increment $\delta(r, k, \text{dir})$ of r on dimension k as the trunk $[(u_d, \dots, u_2, u_1), (v_d, \dots, v_2, v_1)]$, where

$$\begin{aligned} u_i &= a_i, & v_i &= b_i, & \text{if } 1 \leq i \leq d, \quad i \neq k; \\ u_k &= a_k, & v_k &= b_k + 1, & \text{if } \text{dir} = 1; \\ u_k &= a_k - 1, & v_k &= b_k, & \text{if } \text{dir} = -1. \end{aligned}$$

In line 7 of GrowRegion, the increment first grows into the positive direction, then the negative direction (line 12). It repeats this on all dimensions until r cannot grow anymore (line 15). Note that the growing is limited in the region defined by the cover. Since the cover set are much smaller than the whole space, running FindDenseRegion is much more efficient than scanning the whole space for dense regions. Note that using different orders in growing the dimensions in FindDenseRegion may result in different dense region configuration. However, the total volumes resulted from different orderings should be very close.

3.3. Generate Quantitative Association Rules

Now we can transform dense regions to quantitative association rules. Since density threshold is already satisfied, we only need to consider whether support and confidence thresholds are satisfied. From the definition, a quantitative association rule is of the form: $A_1 \subseteq [a_1, b_1] \wedge A_2 \subseteq [a_2, b_2] \wedge \dots \wedge A_{n-1} \subseteq [a_{n-1}, b_{n-1}] \Rightarrow A_n \subseteq [a_n, b_n]$ where A_i ($1 \leq i \leq n$) is a quantitative attribute. It is obvious that the rule defines a dense region by the ranges for each A_i . Let this dense region be denoted by R_n . And let the orthographic projection of R_n on the hyperplane formed by dimensions A_1, A_2, \dots, A_{n-1} be denoted by R_{n-1} . Because of orthographic projection, the length of R_{n-1} on i^{th} dimension is the same as that of R_n , where $1 \leq i \leq n - 1$. Now the support of this potential association rule that generated from R_n is the number of points falling in R_n , and the confidence of this potential rule is the number of points falling in R_n over the number of points falling in R_{n-1} . If both of the support and confidence requirements are satisfied, then a quantitative association rules is successfully generated from the dense region R_n . See an example in Figure 7. We can see a three-dimensional box R_n in the three-dimensional space, and a two-dimensional shadow below it which is R_{n-1} , that is, the orthographic projection of R_n on the plane formed by dimensions A_1 and A_2 .

As we perform the same checking on all the dense regions, we can get all the quantitative association rules.

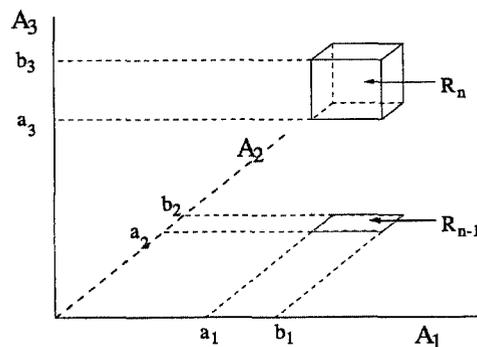


Figure 7. Example of counting support and confidence.

4. TIME COMPLEXITY AND MEMORY LIMITATION IN DISCOVERING DENSE REGIONS

In this section, we will first give the time complexity of algorithm DRMiner, then we show how to handle the problem of memory limitation.

4.1. Complexity of DRMiner

In this section, we will analyze the complexity of DRMiner. Let N be the number of data points, and N_c be the number of valid cells. Also, let the number of leaf node on the k - d tree T_r be N_d , and C_p be the average number of valid cells in a leaf node. According to the splitting criterion and Theorem 1, $C_p \leq v_{\min}/v_{cl}$.

The complexity of inserting a point into the k - d tree consists of two parts .

- (1) the time to locate a leaf node which is $O(\log(N_x))$, where N_x is the number of leaf nodes;
- (2) the time to insert the point in the node which is in the order of $d \times \log(C_p)$, where d is the number of dimensions.

Hence the complexity of building the k - d tree is in the order of $N \times (\log(N/C_p) + d \times \log(C_p))$. Since C_p is bounded by v_{\min}/v_{cl} , $O(N \times \log N)$ is an upper bound on the time complexity to build the k - d tree.

The complexity of computing the dense region cover set in a leaf node is determined by the number of admissible cells in the node which is bounded by C_p . The cost of finding touching admissible cells for a boundary, minimum bounding box for a set of admissible cells, and computing a cover from a boundary, are all linear to the number of admissible cells. Therefore the time for computing the dense region cover set is $O(N_d \times d \times C_p)$, which is bounded by $O(d \times N)$.

Since the k - d tree is a binary tree, the number of nodes that require to perform dense-region-cover merging is at most $N_d - 1$. The complexity of the merge in each node is $O(d \times N_x \log(N_x))$, where N_x is the size of cover set involved in each merge. In general, N_x is very small, and is bounded in the worst case by the number of dense regions which is also a small number. Therefore, the complexity of the merging is bounded by $O(d \times N)$.

The procedure FindDenseRegion needs to scan all the admissible cells once in every cover. For each admissible cell, it will at most check all its $2d$ neighboring cells. Hence the complexity of finding the dense regions in a cover is $O(d \times N_c)$, where N_c is the number of cells in the cover. Since the volume of the cover set is on the same order as that of the dense regions, the time to find all dense regions is bounded by $d \times N$.

In summary, the complexity for the second and third steps is linear to N and d . The dominating cost is in the building of the k - d tree, which is bounded by $O(N \times \log(N/C))$. This shows that DRMiner is a very efficient algorithm.

4.2. Space Complexity Analysis

In general, we work on the cell level, that is, we only store information on valid cells, not every data points. Therefore, the memory space required to store k - d is $O(C)$, where C is the total number of valid cells, which should not be too big in most cases.³ This also depends on the cell size. However, if there is not enough memory, the tree can be stored on disk, and the computing of dense region cover set can be performed separately on different branches. Cover sets from different branches can be merged afterwards.

In the last step of finding dense regions from their cover set, since dense regions have high density and their cover set have the same order of capacity, the array built from a cover in general should not be too big to fit in the memory. However, if it happens that a cover cannot be fully contained in the available memory, then chunking can be used to partition the cover.

³In the worst case when each cell contains only one data point, the total number of valid cells C is the same as the total number of data points.

dense regions can be computed in each chunk separately [28]. At the end, an additional step of merging the dense regions found from the chunks is required.

5. EXPERIMENTAL RESULTS

Some experiments have been carried out to assess the performance of DRMiner. All the experiments are performed on a Sun Sparc 5 workstation running Solaris 2.6 with 64 M main memory. Here we use synthetic data to evaluate the performance of DRMiner. Details of synthetic data generation will be discussed in Section 5.1. The evaluation of discovering dense regions will be shown in Section 5.2

5.1. Generation of Synthetic Database

In the performance studies, we use synthetic database to evaluate the performance of DRMiner. The main parameters for synthetic database generation are listed in Table 2. The databases that we used for the experiments are generated by a two-step procedure. The procedure is governed by several parameters, which gives the user control over the structure and distribution of the generated data tuples. In the first step of the procedure, a number of potential non-overlapping dense regions are generated. In the second step, points are generated with in each potential dense region, as well as the remaining space. For each generated point, a tuple corresponding to that point is generated.

Table 2. Input parameters of data generation.

Parameter	Meaning
d	no. of dimensions
L_i	length of dimension i
ρ_s	density of the sparse region
m	average multiplicity for the whole space
N_{dr}	no. of dense regions
\bar{l}_i	average length of dense regions in dimension i
σ_i	standard deviation of the length of d.r. in dimension i
$\bar{\rho}_{dr}$	average density of dense regions
\bar{m}_{dr}	average multiplicity for the dense regions

The data for the experiments are generated by a two-step procedure. The user first specifies the number of dimensions (d) and the length (L_i) of each dimension of the multidimensional space in which data points and dense regions are generated. In the first step, a number (N_{dr}) of non-overlapping hyper-rectangular regions, called “*potentially dense regions*”, are generated. The lengths of the regions in each dimension are carefully controlled so that they follow a normal distribution with the mean (\bar{l}_i) and variance given by the user.

In the second step, data points are generated in the potential dense regions as well as the whole space, according to the density parameters ($\bar{\rho}_{dr}$) specified by the user. Within each potential dense region, the generated data points are distributed randomly. Each data point is next used to generate one or more tuples, which are inserted to an initially empty database. The average number of tuples per space point is specified by the user.

This procedure gives the user flexible control on the number of dimensions, the lengths of the whole space as well as the dense regions, the number of dense regions, the density of the whole space as well as the dense regions, and the size of the final database.

5.1.1. Step 1. Generation of potentially dense regions

This step takes several parameters as shown in Table 2. The first few parameters determine the shape of the multidimensional space containing the data. The parameter d specifies the number

of dimensions of the space, while the values L_i ($i = 0, 1, 2, \dots, d - 1$) specify the length of the space in each dimension. Valid coordinate values for dimension i are $[0, L_i)$. Thus, the total volume of the space v_{DCS} is given by

$$v_{DCS} = \prod_{i=0}^{d-1} L_i. \quad (1)$$

The parameter ρ_s is the average density of the sparse region, which is the parts of the space not occupied by any dense regions. Density is defined as the number of tuples divided by the total hyper-volume. On average, each point corresponds to m tuples in the final database. This parameter is called the “multiplicity” of the whole space. Therefore, the number of data tuples generated, N_t , will be

$$N_t = m \cdot N_p, \quad (2)$$

where N_p is the total number of distinct points in the space.

The next parameter N_{dr} specifies the total number of potentially dense regions to be generated. The potentially dense regions are generated in such a way that overlapping is avoided. The length of each region in dimension i is a Gaussian random variable with mean \bar{l}_i and standard deviation σ_i . Thus, the average volume of each potentially dense region is

$$\bar{v}_{dr} = \prod_{i=0}^{d-1} \bar{l}_i. \quad (3)$$

The position of the region is a uniformly distributed variable, so that the region will fit within the whole multidimensional space. If the region so generated overlaps with other already generated regions, then the current region is shrunk to avoid overlapping. The amount of shrinking is recorded, so that the next generated region can have its size adjusted suitably. This is to maintain the mean lengths of the dense regions to be \bar{l}_i . If a region cannot be shrunk to avoid overlapping, it is abandoned and another region generated instead. If too many attempts have been made without successfully generating a new region which does not overlap with the existing ones even after shrinking, the procedure aborts. The most probable cause for this is that the whole space is too small to accommodate so many nonoverlapping potentially dense regions of such large sizes.

To each potentially dense region are assigned two numbers—the density and the average multiplicity. The density of each potentially dense region is generated so that it follows a Gaussian random variable with mean $\bar{\rho}_{dr}$ and standard deviation $\bar{\rho}_{dr}/20$. This means that on average, each potentially dense region will have $\bar{\rho}_{dr} \cdot \bar{v}_{dr}$ points generated in it. The average multiplicity of the region is a Poisson random variable with mean \bar{m}_{dr} . These two assigned values are used in the next step of the data generation procedure.

5.1.2. Step 2. Generation of points and tuples

The next step takes in the potentially dense regions generated in Step 1 as parameter, and generates points in the potentially dense regions as well as the whole space. Tuples are then generated from these generated points according to the multiplicity values.

To generate the data, a random point in the whole space is picked. The position of the point is determined by uniform distribution. The point is then checked to see if it falls into one of the potentially dense regions. If so, it is added to that region. Otherwise, it is added to the list of “sparse points”. This procedure is repeated until the number of points accumulated in the sparse point list has reached the desired value $\rho_s(v_{DCS} - N_{dr} \cdot \bar{v}_{dr})$.

Next, each potentially dense region is examined. If it has accumulated too many points, the extra points are dropped. Otherwise, uniformly distributed points are repeatedly generated within that potentially dense region until enough points (i.e. $\bar{\rho}_{dr} \cdot \bar{v}_{dr}$) have been generated. After

this, all the points in the multidimensional space have been generated according to the required parameters as specified by the user. The total number of points generated is the sum of the number of points generated in the sparse region as well as the dense regions. Thus,

$$\begin{aligned} N_p &= \rho_s(v_{DCS} - N_{dr} \cdot \bar{v}_{dr}) + \bar{\rho}_{dr} \cdot \bar{v}_{dr} \\ &= \rho_s \cdot v_{DCS} + N_{dr} \cdot \bar{v}_{dr} \cdot (\bar{\rho}_{dr} - \rho_s). \end{aligned} \quad (4)$$

Finally data tuples are generated from the generated points. For each point in a potentially dense region, a number of tuples occupying that point is generated. This number is determined by an exponentially distributed variable with mean equal to the value assigned as “multiplicity” for that region in the previous step. For each point in the sparse list, we also generate a number of tuples. But this time, the number of tuples is determined by an exponentially distributed variable with a mean which achieves an overall multiplicity of m for the whole space, so that equation (2) is satisfied. From equations (1), (2), (3), and (4), we get

$$N_t = m \cdot \left(\rho_s \cdot \prod_{i=0}^{d-1} L_i + N_{dr} \cdot (\bar{\rho}_{dr} - \rho_s) \cdot \prod_{i=0}^{d-1} \bar{l}_i \right). \quad (5)$$

So, the total number of tuples (N_t) generated can be controlled by adjusting the parameters. Thus, the size of the database can be easily controlled.

5.2. Evaluation of Discovering Dense Regions

In this section, we will show the experimental result of discovering dense regions, and our focus is the speed of DRMiner with different dimension number and different percentage of sparse points and other factors. The speed is measured by the time used to discovering dense region. Our expectation is the speed of DRMiner can be scalable under different conditions.

Here, ρ_{min} , ρ_{low} , and v_{min} are all inputs to experiments. To simplify the experiment, we use a default cell volume of 20 for the cells in DRMiner. In this section, we simply refer the average dense region density $\bar{\rho}_{dr}$ as the dense region density, and the minimum density threshold ρ_{min} as the density threshold. We also set $\rho_{low} = \rho_{min}/2$ and $v_{min} = 4096$ and $\rho_{min} = \bar{\rho}_{dr}$, in all the experiments below expect other specification.

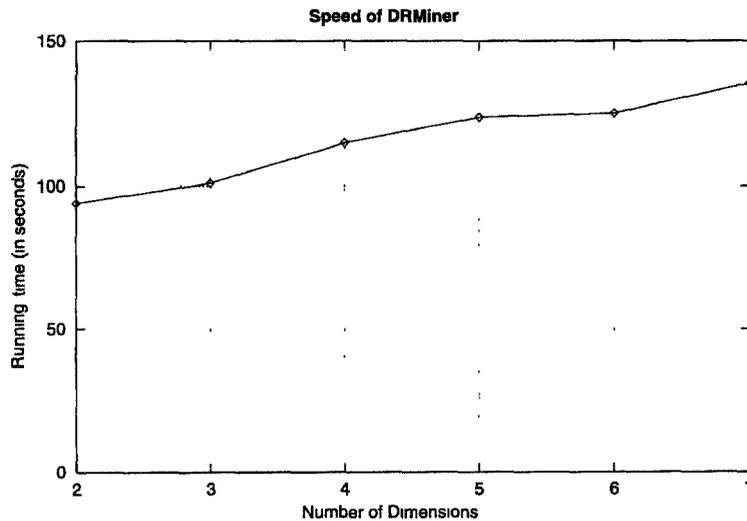


Figure 8. Speed of DRMiner on different number of dimensions.

5.2.1. Effect of DRMiner on different number of dimensions

In these set of experiments, we fixed the volume of the d -dimensional space and increased the number of dimensions from two to seven. The d -dimensional space has a volume of 2×10^{10} with different dimensional lengths. $N_{dr} = 10$, $\bar{\rho}_{dr}=20\%$. The average volume of a potential dense region is 5×10^5 , and the number of data points in the whole d -dimensional space is about one million, in which about 5% are sparse points.

Figure 8 clearly shows that the speed of DRMiner does not increase exponentially as the number of dimensions increases. This is what we have expected from the complexity analysis.

5.2.2. Effect of DRMiner on different number of sparse points

In Section 5.2.1, we have studied the performance of DRMiner on high-dimensional case. Here, we investigate the speed of DRMiner for different percentage of sparse points. Here, we fix the number of dimension to 3 and the three-dimensional space has a volume of 2×10^{10} with different dimensional lengths. $N_{dr} = 10$, $\bar{\rho}_{dr}=20\%$. And the number of points in dense regions is about one million. We increase the percentage of sparse points in the total number of points from 1% to 100%, Figure 9 shows the result.

In Figure 9, we can see that the speed of DRMiner is slightly affected by the percentage of sparse points when the sparse point percentage is less than 20% and not greatly affected w.r.t. 50%. This is because although the number of sparse points in this space is increased, this increasing does not greatly affect the density of sparse region. And this means that the density difference between dense region and sparse region is still great, so the dense region cover will not be enlarged significantly, this in turn only leads to slightly increasing of the time cost of DRMiner. However, when most of the points becomes sparse points, the time consuming increase sharply.

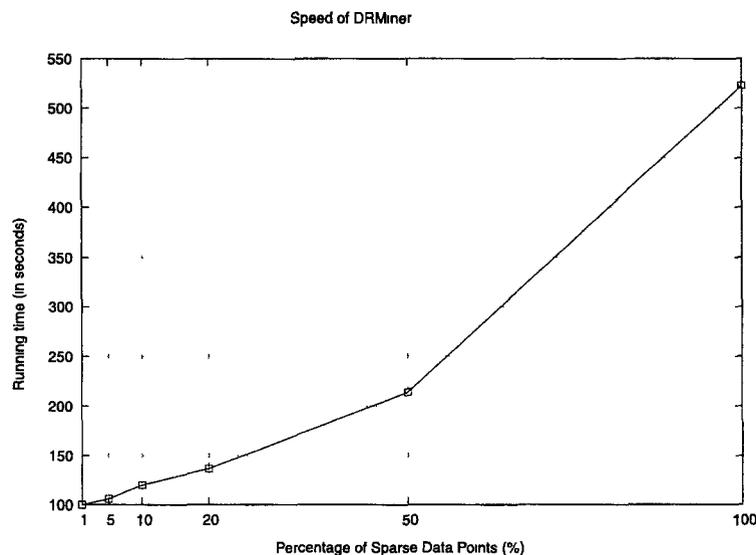


Figure 9 Speed of DRMiner on different number of dimensions

5.2.3. Effect of DRMiner on larger database size

We also studied the effect of increasing the size of our input database. Here, we fixed the number of dimension to three, and $N_{dr} = 10$, the size of the whole space is 2×10^{10} . The size of Database changes from containing 0.5 millin data points to containing four million data points, while the portion of the sparse points is fixed to 5% to the total number of points. Figure 10 shows the result.

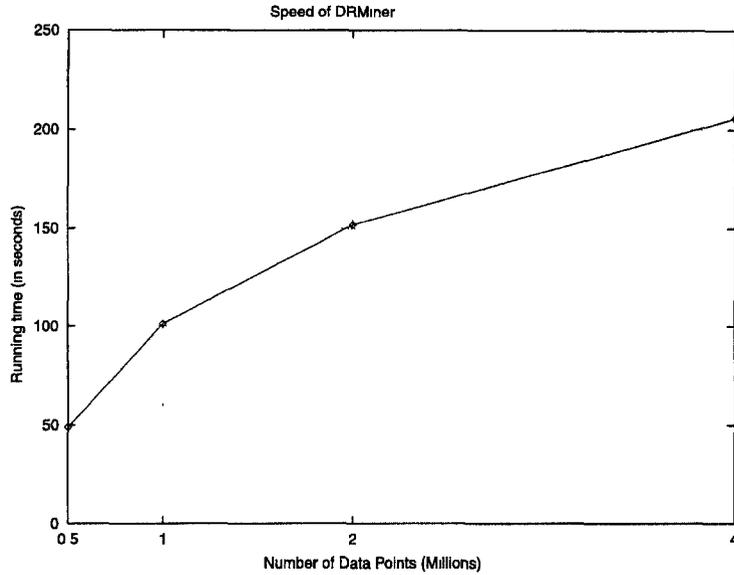


Figure 10. Speed of DRMiner with different database size.

From Figure 10, we can notice that the speed increases linearly as the number of points increases, which is comparable to our analysis in Section 4.

5.2.4. Effect of DRMiner on different number of dense regions

Besides above experiments, we also test the speed performance of DRMiner with different number of dense regions from 5 to 100. Again, these experiments are in a three-dimensional space with the total volume of it is 2×10^{10} , and the total number of points is around one million with 5% of them are sparse points. Figure 11 gives the result.

In Figure 11, we can see the speed is decreased as the number of dense regions increasing. The reason is first: the size of each dense region becomes smaller. This reduces the amount of splitting of the dense regions across the $k-d$ tree nodes; and second as the size of dense region decreased, the size of dense region cover set becomes smaller accordingly. This in turn speed up the dense region growing in these dense region cover set.

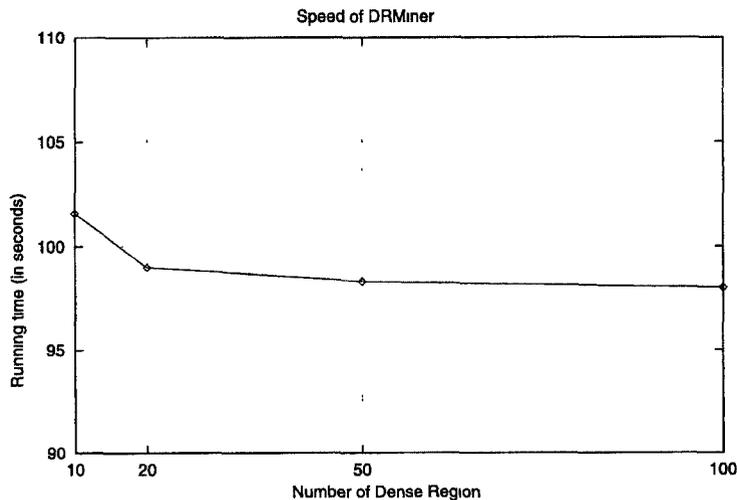


Figure 11 Speed of DRMiner with different number of dense regions

REMARK. Beside the above experiments, we have also examined the effect of uniformly distributed data on our algorithm. We summarized the findings as follows. The following data sets are considered:

- (1) data set with uniform density $\leq \rho_{low}$;
- (2) data set with uniform density $\geq \rho_{min}$; and
- (3) data set with uniform density which is between ρ_{low} and ρ_{min} .

For Data Sets (1) and (2), our algorithm works fine, but our algorithm may take a long time to complete (or even fails) because it is likely that the dense region may cover the whole hyperspace.

6. CONCLUSION

In this paper, we have introduced a “density” measure for mining quantitative association rules. Using this density measure and an efficient algorithm for locating dense regions, an efficient algorithm, DRMiner, is developed for quantitative attributes. This DRMiner not only solves the problems of previous algorithms, but also can scale up well for high-dimensional cases as supported by the preliminary experimental results. In fact, the techniques presented in this paper can also be applied in other areas. For example, DRMiner is useful in indexing OLAP data for reducing the query response time [23].

A major weakness of our method is that it requires the user to specify many thresholds, this may be difficult in many cases. However using our algorithm is still valuable to get to know more about the data. Another problem unsolved is the dimension curse, we have done extensive experiments in less than ten-dimensional space, and notice in case of above ten dimension, our method performs poor. This is because that k - d -tree can not efficiently handle high-dimensional data. Therefore we will try to seek help from other more robust multidimensional index structures in the future.

There are several other possible directions for future research regarding this work. As mentioned in the paper, based on the current definition of a dense region, it may be possible that for an extremely high density region to swallow regions with relatively low density. It is expected that this may create a problem if the densities of the dense regions vary a lot. The impact of this needs more investigation. There can be a number of possible solutions to this problem. We can either expand each region as large as possible according to the current definition, then prune back the border areas that have density that are relatively a lot lower than the density of the core region. Or we can refine the definition to make sure that only regions with relatively similar density can be merged. Note that the algorithm we developed in the paper is a heuristic-based approach, thus does not guarantee an optimal solution to the optimization problem stated in Section 3. It is interesting to develop an algorithm (possibly using a dynamic programming approach) that can output an optimal solution to the problem. Also, the algorithms may not perform very well in some extreme cases, for example, for data set with uniform density which is between ρ_{low} and ρ_{min} . It is worthwhile to have an algorithm which fits for more cases.

REFERENCES

1. R. Agrawal and R. Srikant, Fast algorithms for mining association rules, In *Proceedings of the 20th International Conference on Very Large Databases*, Santiago, Chile, September 1994, pp. 487-499.
2. J.S. Park and M.-S. Chen and P.S. Yu, An effective hash-based algorithm for mining association rules”, In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Jose, CA, May 1995, pp. 175-186.
3. A. Savasere, E. Omiecinski and S. Navathe, An efficient algorithm for mining association rules in large databases, In *Proceedings Of the 21st International Conference on Very Large Databases*, Zurich, Switzerland, September 1995, pp. 432-444
4. R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer and A. Swami, An interval classifier for database mining applications, In *Proceedings of the 18th International Conference on Very Large Databases*, Vancouver, Canada, August 1992, pp. 560-573.

- 5 D.W. Cheung, V.T. Ng, A W Fu and Y.J. Fu, Efficient mining of association rules in distributed databases", Special Issue in *Data Mining. IEEE Transactions on Knowledge and Data Engineering* 8 (6), 911-022, (1996)
- 6 D W Cheung and Y Xiao, Effect of data skewness in parallel mining of association rules, In *Proceedings of the 2nd Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Melbourne, Australia, April 1998, pp. 291-314
7. J. Han and Y. Fu, Discovery of multiple-level association rules from large databases, In *Proceedings Of the 21st International Conference on Very Large Databases*, Zurich, Switzerland, September 1995, pp. 420-431.
8. H. Toivonen, Sampling large databases for association rules, In *Proceedings of the 22nd International Conference on Very Large Databases*, Mumbai, India, September 1996, pp. 134-145.
- 9 S Sarawagi, Indexing OLAP data, *Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society* 20 (1), 432-444, (1997).
10. M. Ester, H Kriegel, J Sander and X Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, August 1996, pp 226-231
11. M Ankerst, M. Breung, H. Kriegel and J Sander, OPTICS. Ordering points to identify the clustering structure, In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 49-60, (1999).
- 12 X Wang and H. Hamilton, DBRS: A density-based spatial clustering method with random sampling, In *Proceedings of the 3rd Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Seoul, Korea, May 2003, pp 563-575
- 13 W. Wang, J Yang and R. Muntz, STING: A statistical Information grid approach to spatial data mining, In *Proceedings of the 23rd International Conference on Very Large Databases*, Athens, Greece, August 1997, pp 186-195
- 14 R. Agrawal, J. Gehrke, D. Gunopulos and P. Raghavan, Automatic subspace clustering of high dimensional data for data mining applications, In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Seattle, WA, 1998, pp 94-105.
15. R. Ng and J Han, Efficient and effective clustering methods for spatial data mining, In *Proceedings of the 20th International Conference on Very Large Databases*, 1994, pp 144-155
- 16 A Denton, Q. Ding, W. Perrizo and Q. Ding, Efficient hierarchical clustering of large data sets using P-trees, In *Proceedings of the 15th International Conference on Computer Applications in Industry and Engineering (CAINE'02)*, San Diego, CA, November 2002, pp. 138-141.
- 17 J Shafe, R. Agrawal and M Mehta, SPRINT: A scalable parallel classifier for data mining, In *Proceedings of the 22nd International Conference on Very Large Databases*, Bombay, India, September 1996, pp 544-555.
- 18 R Srikant and R Agrawal, Mining quantitative association rules in large relational tables, In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Montreal, Canada, June 1996, pp. 1-12.
- 19 B. Lent, A. Swami and J Widom, Clustering association rules, In *Proceedings of International Conference on Data Engineering*, Birmingham, U K., April 1997, pp 220-231
- 20 Y Aumann and Y. Lindell, A statistical theory for quantitative association rules, *Journal of Intelligent Information Systems* 20 (3), 255-283, (2003).
21. G Webb, Discovering associations with numeric variables, In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001, pp 383-388.
22. T Fukuda, Y Morimoto, S. Morishita and T Tokuyama, Data mining with optimized two-dimensional association rules, *ACM Transactions on Database Systems (TODS)* 26 (2), 179-213, (2001).
23. D W. Cheung, B. Zhou, B Kao, H Kan and S.D. Lee, Towards the building of a dense-region based OLAP system, *Data and Knowledge Engineering* 36, 1-27, (2001).