Proceedings of Fifteenth International Joint Conference on Artificial Intelligence (*IJCAI-97*),
Nagoya, Japan, 1997, pp. 930-935.

Discovering Interesting Holes in Data

Bing Liu, Liang-Ping Ku and Wynne Hsu

Department of Information Systems and Computer Science
National University of Singapore
Lower Kent Ridge Road
Singapore 119260

Email: {liub, kulp, whsu}@iscs.nus.sg

A3bstract

Current machine learning and discovery techniques focus on discovering rules or regularities that exist in data. An important aspect of the research that has been ignored in the past is the learning or discovering of interesting holes in the database. If we view each case in the database as a point in a *k*-dimensional space, then a hole is simply a region in the space that contains no data point. Clearly, not every hole is interesting. Some holes are obvious because it is known that certain value combinations are not possible. Some holes exist because there are insufficient cases in the database. However, in some situations, empty regions do carry important information. For instance, they could warn us about some missing value combinations that are either not known before or are unexpected. Knowing these missing value combinations may lead to significant discoveries. In this paper, we propose an algorithm to discover holes in databases.

1 Introduction

Current machine learning and machine discovery techniques mainly focus on finding rules or formulas in data. They typically analyze each case (or tuple) in the database to induce or discover regularities that exist in the data cases. For example, a typical classification rule learning system (e.g., C4.5 [Quinlan, 1992]) induces a set of characteristic descriptions (or classification rules) from the cases in the database for some given classes. A clustering system (e.g., COBWEB [Fisher, 1987]) groups cases in the database into various similarity classes and derives a concept hierarchy. A scientific discovery system (e.g., BACON [Langley *et al.*, 1987]) typically discovers, among other things, mathematical formulas that fit the data. In this paper, we show that an important aspect of the research that has been ignored in the past is the discovering of large empty areas (or holes) in databases. If we view each case (or tuple) in the database as a point in a *k*-dimensional space, then a hole is simply a region in the space that contains no data point. In a continuous space, there always exist a large number of holes because it is not possible to fill up the continuous space with data points. The existence of large holes is, however, mainly due to the following two reasons:

1. The data cases collected are insufficient, resulting in some regions having no data point.
2. Certain value combinations are not possible. For example, in a particular domain, we have a database with two continuous attributes X and Y. Both X and Y can take values from 1 to 10. However, when $X > 5$, Y is always less than 4. In other words, there exists an empty area, i.e., a rectangular region defined by $5 < X \leq 10$ and $4 \leq Y \leq 10$.

Clearly, learning or discovering associative relationships (e.g., rules, formulas, etc.) that exist in data is important. In this paper, we argue that discovering the *missing* associations is also significant. For example, in a disease database we may find that certain symptoms and/or test values do not occur together, or when a certain medicine is used, some test values never go beyond certain range. Discovery of such information can be of great importance in medical domains because it could mean the discovery of a cure to a disease or even some biological laws.

It must be stressed that in many applications, producing the discovered rules alone does not provide the user with the complete information. For example, a particular organization has used a learning system to generate a set of rules from their database. One of the rules is:

If Compy_Size > 2 **then** Service = *Yes*.

This rule says that if the *Compy_Size* (company size) is greater than 2, the company uses the *service* provided by the organization. Assume the company size is partitioned into 10 categories. A close inspection of the database may reveal that no company, whose size is in the range of 4-8, uses the service. Hence, there is a hole in the data. Realizing the existence of this hole may lead this organization to probe into the possibilities of modifying its service or of doing more promotion to attract the medium size companies to use its service. In the case of one dimensional dataset, discovering the hole is simple. However, when the number of dimensions increases, the problem quickly becomes rather complex. To the best our knowledge, there is no existing technique that is able to perform this task. This paper proposes such a technique.

In general, a database contains a large number of holes because each case in the database is only a point in a *k*-dimensional space. Even if each attribute takes discrete or nominal values, it may be still quite difficult to fill the whole space. However not all holes are interesting. Most of them are not. The following types of holes are not interesting:

1. *Small holes*: they exist because there is only a limited number of cases in the database.
2. *Known impossible value combinations*: they exist because certain value combinations are not possible and this fact is known previously.

However, certain types of holes can be of great importance:

1. holes that represent impossible value combinations that are not known previously.
2. holes that indicate that the data collected within those areas are insufficient.
3. holes that are suspected by the user and need confirmation.

This paper proposes an algorithm that is able to find the holes in a k-dimensional continuous space, and sort them according to their sizes.

2 Preliminaries

The database that our algorithm works with is a normal database, which consists of the descriptions of N cases in the form of tuples. Each case in the database is described by w distinct attributes, $A_1, ..., A_i, ..., A_w$, so that in an instantiation of case description, an attribute A_i takes on the value $v_i \in domain(A_i)$. Some attributes take continuous or ordinal values, and we call these attributes continuous attributes. Other attributes take nominal values, and we call them discrete/nominal attributes.

Since the focus of this paper is on the space formed by continuous attributes, the proposed algorithm only uses the continuous attributes in its discovering process. In many situations, the user may not be interested in the holes that exist in the whole database, but only a segment of the database that satisfies certain requirements. Then, some preprocessing can be performed to extract the segment of the database. Assume the resulting segment of the database has k continuous attributes. The user then needs to specify the bounding (minimum and maximum) values for each attribute, denoted by min_i and max_i for $1 \leq i \leq k$. With these attributes and their bounding values, a k-dimensional continuous space S is defined, within which the data tuples (or points) in the segment of the database are contained.

In theory, a hole can be of any shape. In this paper, we restricts the shape to hyper-rectangles (holes of this shape are easily understood by the user). In particular, we are interested in the so-called *maximal hyper-rectangles* (MHR).

Definition: Given a k-dimensional continuous space S, where each dimension i ($1 \leq i \leq k$) is bounded by a minimum and maximum value (denoted by min_i and max_i). There exist n ($n \leq N$) data points (or tuples) in S. A *maximal hyper-rectangle* (MHR) is an empty hyper-rectangle that has no data point within its interior and has at least one data point on each of its $2k$ bounding surfaces. We call these points the *bounding points* of the hyper-rectangle. Each side i of the MHR is parallel to one axis of S and orthogonal to all the others.

The number of MHRs in a continuous space can be huge. However, we are only interested in those MHRs that are *sufficiently large* (or *significant*). The user can specify how to measure the size of a MHR and what size is considered *sufficiently large*. These are all application dependent. A simple way of measuring the size of a MHR is by its volume. As for what size is considered *sufficiently large*, we may use a threshold volume or a minimal length for each side of the MHR, or a combination of both.

Our *objective* is to find all the MHRs in the user-specified k-dimensional continuous space that satisfy the *sufficiently large* criterion and to rank them according to their sizes.

3 The Proposed Algorithm

3.1 Overview of the algorithm

The main idea is as follows. Given a k-dimensional continuous space S, and n points (or data cases) in S, we first start with one MHR, which occupies the entire space S. Then each point is incrementally added to S. At each insertion, we update the set of MHRs that have been found this far. The update is done as follows. When a new point is added, we identify all the existing MHRs that contain this point. These hyper-rectangles are no longer MHRs since they now contain a point within their interiors. Using the newly added point as reference, a new lower and upper bound for each dimension are formed to result in 2 new hyper-rectangles along that dimension. If these new hyper-rectangles are found to be sufficiently large, they are inserted into the list of existing MHRs, otherwise they are discarded.

3.2 The details of the algorithm

Given a point X, we denote $X(i)$ as the value of X along the i^{th} dimension. A MHR, H, is denoted as:

$$H = ((L_1, U_1), ..., (L_i, U_i), ..., (L_k, U_k))$$

where L_i and U_i are respectively the sets of lower and upper bounding points of H along the i^{th} dimension. Note that the lower (or the upper) bound of H is bounded by a set of lower (or upper) bounding points, rather than a single value. Let T denote a data structure that stores a collection of MHRs, and T supports the following functions:

1. *Insert*(T, H): it inserts the MHR H into T.
2. *Delete*(T, H): it deletes the MHR H from T.
3. *ContainmentSearch*(T, X): it returns a list of MHRs from T that contain the point X.

The data structure T can be implemented by first transforming the MHRs into $2k$-dimensional points [Preparata and Shamos, 1985] and then storing them in a Pseudo $2k$-d tree [Overmars and Leeuwen, 1982].

We also define a function *BigEnough*(H) which returns TRUE if the MHR H is considered to be *sufficiently large* (or significant). Note that *BigEnough*() must satisfy the following: if *BigEnough*(H') is true, then *BigEnough*(H) must be true for all H that contain H'.

For simplicity of notation in the algorithm, we let Sl_i and Su_i (for each dimension i) to be "*points*", where $Sl_i(i) = min_i$,

$Su_i(i) = max_i$, and $Sl_i(j) = Su_i(j)$ = undefined for $i \neq j$. Any comparison with $Sl_i(j)$ or $Su_i(j)$ (where $j \neq i$) is always true. Hence, Sl_i (or Su_i) serves as the *bounding plane* (which includes all the points on the plane) along the i^{th} dimension. The proposed algorithm is shown in Figure 1.

```
1  Algorithm FindMHR
2    Insert(T, (({Sl1}, {Su1}), ..., ({Sli}, {Sui}),
                ..., ({Slk}, {Suk})));
3    for each point X in the database do
4      RL = ContainmentSearch(T, X);
5      for each H=((L1,U1), ..., (Li,Ui), ..., (Lk,Uk)) in RL do
6        if X is on a surface of H then
7          insert X into the set of bounding points
               in that surface
8        else
9          Delete(T, H);
10         for each dimension i do
11           for each j ≠ i do
12             L′j = {l ∈ Lj | l(i) < X(i)};
13             U′j = {u ∈ Uj | u(i) < X(i)};
14             L″j = {l ∈ Lj | l(i) > X(i)};
15             U″j = {u ∈ Uj | u(i) > X(i)}
16           endfor;
17           if L′j and U′j not empty for all j ≠ i and
                BigEnough(H′ = ((L′1, U′1), ..., (Li, {X}),
                          ..., (L′k, U′k)) then
18             Insert(T, H′)
19           endif;
20           if L″j and U″j not empty for all j ≠ i and
                BigEnough(H″ = ((L″1, U″1), ..., ({X}, Ui),
                          ..., (L″k, U″k)) then
21             Insert(T, H″)
22           endif
23         endfor
24       endif
25     endfor
26   endfor
27   Sort and report all the MHRs in T according to their sizes;
```

Figure 1. The proposed algorithm

3.3 An example

We use an example to illustrate the working of the algorithm. Suppose we have a 2-dimensional space S (ABCD) as shown in Figure 2. The first MHR is the whole space, ABCD, which is described with:

$$H = ((\{Sl_1\}, \{Su_1\}), (\{Sl_2\}, \{Su_2\})))$$

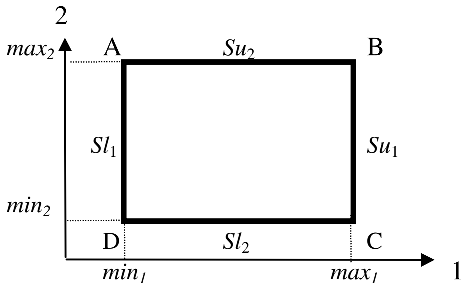

Figure 2. The original space or the first MHR

Line 3. We add the first point $X = P_1$ to the inside of ABCD (see Figure 3).

Line 4. Since in this case there is only one MHR (the original space ABCD) contains P_1 in T, thus

$$RL = \{((L_1, U_1), (L_2, U_2))\},$$

where $L_1 = \{Sl_1\}$, $U_1 = \{Su_1\}$, $L_2 = \{Sl_2\}$ and $U_2 = \{Su_2\}$.

Line 5. The first MHR to be considered is

$H = ((L_1, U_1), (L_2, U_2))$, i.e. ABCD in Figure 2.

Line 6. Since P_1 (Figure 3) is not on a surface of H, we proceed to Line 9.

Line 9. H is deleted since it is no longer a MHR. It will be replaced with smaller MHRs.

Line 10. We split H along each dimension. Dimension 1 is considered first.

Line 11-16. In this part, we look for the sets of bounding points along dimension 2. L'_2 and U'_2 are the sets of lower and upper bounding points for the new MHR that will be formed to the left of P_1. L''_2 and U''_2 are the sets of lower and upper bounding points for the new MHR to be formed to the right side of P_1. We obtain:

$L'_2 = \{Sl_2\}$, $U'_2 = \{Su_2\}$, and

$L''_2 = \{Sl_2\}$, $U''_2 = \{Su_2\}$.

Line 17-19. It forms the new MHR H' that uses P_1 as its upper bounding point on dimension 1, and L'_2 and U'_2 as its sets of lower and upper bounding points on dimension 2. Since in this case, both L'_2 and U'_2 are not empty, then

$$H' = (\{Sl_1\}, \{P_1\}), (\{Sl_2\}, \{Su_2\})),$$

which is AEFD in Figure 3. Assume H' is big enough. It is inserted into T (Line 18).

Line 20-22. This forms the other new MHR H'' that uses P_1 as the lower bounding point on dimension 1. We obtain,

$$H'' = (\{P_1\}, \{Su_1\}), (\{Sl_2\}, \{Su_2\})),$$

which is EBCF in Figure 3. Assume H'' is also big enough. It is inserted into T.

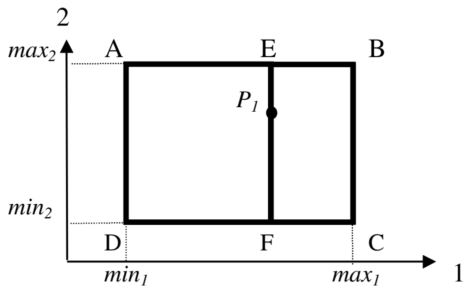

Figure 3. Splitting ABCD along dimension 1

Line 10-16. Next, we consider dimension 2. We have:

$L'_1 = \{Sl_1\}$, $U'_1 = \{Su_1\}$, and $L''_1 = \{Sl_1\}$, $U''_1 = \{Su_1\}$.

Line 17-22. H' and H'' are the two new MHRs formed due to P_1 splitting the original MHR along dimension 2 (see Figure 4). We then have,

$H' = ((\{Sl_1\}, \{Su_1\}), (\{Sl_2\}, \{P_1\}))$ and

$H'' = ((\{Sl_1\}, \{Su_1\}), (\{P_1\}, \{Su_2\}))$,

which are ABGH and HGCD in Figure 4 respectively.

This ends adding the first point and updating T with the new MHRs. Let us now add another point, P_2, into the space (see the location of P_2 in Figure 5).

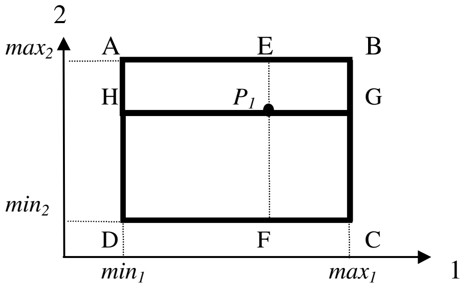

Figure 4. Splitting ABCD along dimension 2

Line 4. The set of MHRs in T containing P_2 is:

$RL = \{((\{Sl_1\}, \{P_1\}), (\{Sl_2\}, \{Su_2\})),$
$((\{Sl_1\}, \{Su_1\}), (\{Sl_2\}, \{P_1\}))\}.$

They are AEFD and HGCD respectively in Figure 4.

Line 5-9. We consider the first MHR,

$H = ((\{Sl_1\}, \{P_1\}), (\{Sl_2\}, \{Su_2\})),$

which is AEFD in Figure 5. Since P_2 is not on a surface of H, H is deleted.

Line 10. Assume dimension 1 is considered first.

Line 11-16. The bounding points obtained are:

$L'_2 = \{Sl_2\}$, $U'_2 = \{Su_2\}$, and $L''_2 = \{Sl_2\}$, $U''_2 = \{Su_2\}$.

Line 17-22. We obtain

$H' = ((\{Sl_1\}, \{P_2\}), (\{Sl_2\}, \{Su_2\}))$ and
$H'' = ((\{P_2\}, \{P_1\}), (\{Sl_2\}, \{Su_2\})),$

which are AKMD and KEFM respectively in Figure 5. Assume they are big enough. They are inserted into T,

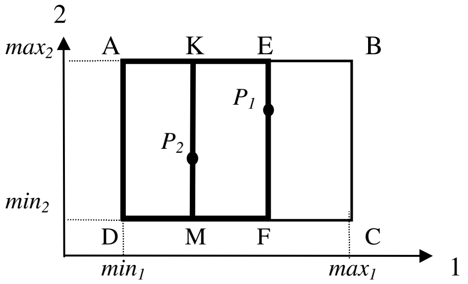

Figure 5. Splitting AEFD along dimension 1

Line 10-16. Consider dimension 2 w.r.t. the same MHR,
$H = ((\{Sl_1\}, \{P_1\}), (\{Sl_2\}, \{Su_2\}))$. We obtain,

$L'_1 = \{Sl_1\}$, $U'_1 = \{\}$, and $L''_1 = \{Sl_1\}$, $U''_1 = \{P_1\}$.

Note that in this case, $U'_1 = \{\}$ because the coordinate of P_1 along dimension 1 is greater than the coordinate of P_2 along dimension 1. See OF in Figure 6 (there is no point on OF). This means that there is no upper bounding point along dimension 1.

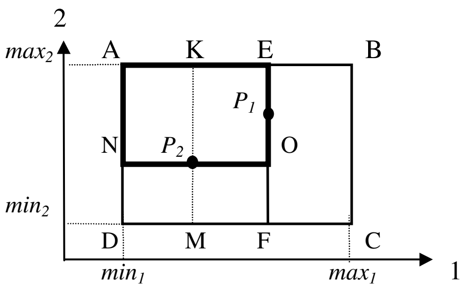

Figure 6. Splitting AEFD along dimension 2

Line 17-22. There is no H' because $U'_1 = \{\}$. H' should be (if it exists) NOFD in Figure 6, but NOFD is not a MHR. Its space will be occupied later by another MHR. Here we only have H'',

$H'' = ((\{Sl_1\}, \{P_1\}), (\{P_2\}, \{Su_2\})),$

which is AEON in Figure 6.

Line 5. We now go back to work on the other MHR,

$H = ((\{Sl_1\}, \{Su_1\}), (\{Sl_2\}, \{P_1\}))\},$

which is HGDC in Figure 4 or 7.

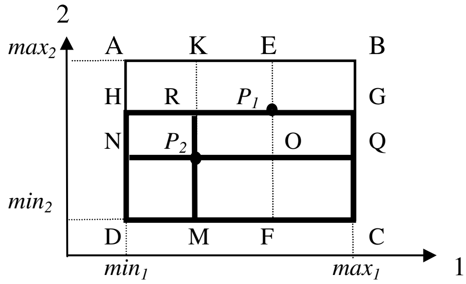

Figure 7. Splitting HGCD

Line 6-21. After going though the similar process as the above, we obtain 3 new MHRs,

$((\{P_2\}, \{Su_1\}), (\{Sl_2\}, \{P_1\}))$: RGCM in Figure 7,
$((\{Sl_1\}, \{Su_1\}), (\{Sl_2\}, \{P_2\}))$: NQCD in Figure 7,
$((\{Sl_1\}, \{Su_1\}), (\{P_2\}, \{P_1\}))$: HGQN in Figure 7.

Note that HRMD is not a MHR because there is no bounding point on HR, however, HRMD is inside the MHR AKMD formed earlier. NOFD, which was not included in any new MHR when we consider AEFD, is now inside NQCD.

The final of sets of MHRs in T are RGCM, NQCD, HGQN, AKMD, KEFM, AEON, ABGH, EBCF respectively in Figure 7.

3.4 Proof of correctness

A sketch of the proof of the correctness of the algorithm is presented next. We prove that the algorithm FindMHR produces *all possible* MHRs that have sizes accepted by the *BigEnough*() function. Hence, we need to show that any hyper-rectangle reported by FindMHR *must be maximal* and accepted by *BigEnough*(), and any given MHR that is accepted by *BigEnough*() in the space *S must be reported*. The former is easy to show as our algorithm checks each hyper-rectangle before it is inserted into *T*. Hence we focus our attention on proving the latter.

The detailed proof is too long to be presented here, instead we give an outline of the proof (interested reader may refer to [Ku *et al.*, 1997] for the detailed proof). Let us first assume that *BigEnough*() accepts MHR of all sizes, i.e., no pruning is done. Our task is to prove that the algorithm FindMHR finds all possible MHRs, i.e., the data structure *T* stores all possible MHRs. The proof will proceed in an inductive manner, showing at the base case, when there is no point inserted, there is only 1 MHR which occupies the whole space. Then at each iteration, if *T* already stores all MHR before the next point *X* is inserted, by inserting *X* and performing all the necessary updates, *T* contains all resultant MHRs. This in turn is decomposed into several parts:

1. Show that those MHRs in T not containing X will not be affected (which justifies the use of the function *ContainmentSearch*()).
2. Show that those MHRs in T containing X will not be maximal anymore (which justifies why they are deleted).
3. Show that the new MHRs must be inside the union of the MHRs found by *ContainmentSearch*() (which justifies why the new MHR are generated only from these MHR found).
4. Show that the new MHRs must touch X and all possible MHRs that touch X will be reported (which justifies the way the new MHRs are constructed).
5. Show that no identical MHRs can be generated using this algorithm (which justifies why we do not check for duplicates).

A final point is that in our algorithm, the newly generated MHR is always contained within the MHR from which it is derived from, hence it is safe to discard any MHR that is not accepted by *BigEnough*() (by the definition of *BigEnough*()). With this, the correctness of the algorithm FindMHR is proved.

3.5 Run time complexity

Each insert and delete operation on T can be implemented in $O((\log m)^2)$ amortized time, where there are $O(m)$ number of MHRs stored in T. Containment search in T is done in $O(\log m + C)$ where there are $O(C)$ number of MHRs returned [Overmars and Leeuwen, 1982].

A loose bound on the total number of MHR (i.e., m) is $O(n^{2(k-1)})$ since a possible MHR can be constructed by choosing $2(k-1)$ points from the n points, to bound the k-1 dimensions, and the bounding points for the last dimension is simply decided from the geometry. We also assume that the number of bounding points per MHR is small, thus the operation to compute L'_j, U'_j, L''_j, L''_j is almost constant.

With this assumption, the algorithm runs in $O(n(\log m + C) + nC(\log m)^2 + nCk(\log m)^2 + nCk^2)$ which simplifies to $O(nCk(\log m)^2 + nCk^2)$. Since C is of $O(m)$ and m is of $O(n^{2k-2})$, the run time is simplified to $O(n^{2k-1}k^3(\log n)^2)$.

Although this complexity is high, in practice the number of *sufficiently large* MHRs is not many. Thus, computational cost is usually not a problem.

4 Experimental Results

The proposed algorithm is implemented in C++. Two sets of experiments have been conducted to test the efficiency and effectiveness of the algorithm in discovering the MHRs. The first set of experiments aims to show that the running time of the algorithm is reasonable. All the datasets are randomly generated. Some areas are intentionally left empty. But we have no knowledge about the existence of holes in other areas. Here, we only present the results from the 3D datasets because they are more representative of the real-life situations. Holes of more than 3D are hard to understand (interested reader may refer to [Ku *et al.*, 1997] for experimental results using higher dimensional datasets). Let X, Y and Z denote the 3 dimensions. The resolution of X, Y and Z is of an accuracy of up to 1/100. We run the first set of experiments on 3 different datasets. The lower and upper bounds for X and Y in all the three datasets are 0.00-23.00 and 0.00-17.00 respectively. The bounds for Z are 0.00-20.00, 0.00-70.00 and 0.00-150.00 respectively for the three datasets. The planted empty areas are of the same size, 9, 6 and 9 along X, Y and Z dimensions respectively. Table 1 summarizes the run time results of the three datasets (running on Digital Alpha 8400 under normal loading conditions). Note that T (which stores all the MHRs) is currently implemented as a linked list.

Table 1. Results of the first Set of Experiments.

Data set	No. of points	Running time *a*:*b* – *a* mins, *b* secs		No. of MHRs found	
		8–4–8	4–2–4	8–4–8	4–2–4
1	1427	0 : 02	0 : 06	85	2299
2	5064	0 : 15	1 : 05	162	6925
3	10702	1 : 10	6 : 17	260	13141

Column 1 indicates the dataset number. Column 2 shows the number of data points in each dataset. Column 3 gives the running times for finding all the MHRs that satisfy the minimal length requirements along the 3 dimensions. This column is further divided into two sub-columns. One of them shows the running times when the minimal lengths along X, Y and Z dimensions of the MHRs are 8, 4 and 8 respectively (i.e., the bounds used by *BigEnough*()). The other shows the running times when the minimal length is reduced by half along each dimension (i.e., 4-2-4). Column 4 gives the number of MHRs discovered by the algorithm in each of the two situations. From the table, we see that when the minimal size of the MHR decreases, the time taken to find all the sufficiently large MHRs increases. We also see that as the number of data points increases, the time taken to find all the large MHRs also increases. In general, however, the relationship between the number of data points and the time taken to find all the large MHRs is complex because there are other factors that play a role, e.g., the density and/or the distribution of the data points, and the number of large MHRs that exist in the dataset. In all experiments, the running times are reasonable. In spite of the large number of discovered MHRs (both planted and unknown), many of them actually represent the same general regions with slight variations (slightly different sets of bounding points). In practice, post-processing can be performed to extract those general empty regions.

In the second set of experiments, we use a real-life disease dataset. This dataset has 7 continuous attributes, and 713 data points. The algorithm is run 8 times using different attribute combinations, i.e., combining 2, or 3 or 4 attributes. The minimal size of the MHRs in each experiment is specified by a doctor. The running times of all the tests are within 1 or 2 seconds.

In these experiments, a number of interesting holes are discovered. For example, it is suspected that if the systolic blood pressure (SBP) of a subject is high, then his/her disystolic blood pressure (DBP) is also high. A hole is discovered in the region of high SBP and low DBP. This hole confirms the suspected fact. Some holes are quite unexpected.

For instance, the doctor has a strong belief that the higher the SBP, the more likely the subject will get the disease. However, it is found that between the age of 18-50 (the upper bound of age is 69), there is no subject whose SBP is higher than 145 (the upper bound is 231) and is diagnosed to have the disease.

5 Related Work

To the best of our knowledge, no existing algorithm is able to find interesting holes in a multi-dimensional database. Although there are algorithms in geometry [Chazelle *et al.*, 1986; Orlowski, 1990] that can find empty rectangles in the 2D space, these algorithms cannot be extended to the multi-dimensional space.

Most current research in machine learning and machine discovery focuses on finding rules or formulae that exist in data. Our work is different from rule induction [e.g., Quinlan, 1992] because rule induction is not concerned with empty areas. It typically groups the empty areas with the data areas in order to arrive at some generalized rules.

Typical existing scientific discovery systems discover qualitative and numeric laws from data. Examples of well-known systems include ABACUS [Falkenhainer and Michalski, 1986], BACON [Langley *et al.*, 1987], FAHRENHEIT [Zytkow, 1990], and IDS [Nordhausen and Langley, 1993]. They are different from our work because our algorithm is targeted at discovering those empty areas, which may represent impossible value combinations.

Conceptual clustering systems [e.g., Fisher, 1987] typically partition the data cases into similar classes and form concept hierarchies. Again, they are not concerned with those empty areas that do not contain any data.

In data mining research, many techniques have been proposed to discover regularities in data [Fayyad *et al.*, 1996]. They are similar to those above for machine learning. Here, we would like to mention specifically the association rule discovery technique in [Agrawal *et al.*, 1993]. This technique discovers associations that exist in the database whose attributes are all nominal (or discrete) attributes. For this type of databases, the concept of MHR does not apply. The equivalent concept of MHR in the nominal case is *missing associations*. Since the algorithm in [Agrawal *et al.*, 1993] can find all the existing value associations in a database. Using a simple technique (e.g., generate and test), it is possible to find all the missing associations (without going through the database again). However, the problem is the efficiency and the representation of the missing associations. In our future work, we will study this problem. This paper only focuses on discovering those significant MHRs in a continuous space.

6 Conclusion

This paper argues that although discovering rules or regularities that exist in data is important, in many situations, discovering of large holes in the database is also interesting. An algorithm that is able to discover holes in the continuous space, also known as *maximal hyper-rectangles* (MHR), is proposed and implemented. We believe this algorithm will be useful in scientific discovery and data mining.

Acknowledgments

We would like to thank Dr. Hing-Yan Lee, Ms. Hwee-Leng Ong and Ms. Angline Pang from Information Tehcnology Institute, and Dr. Ke-Qing Gong and Dr. King-Hee Ho from National University Hospital for many useful discussions, for providing us the databases, and for their help in the testing of our system.

References

[Agrawal *et al.*, 1993] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD-1993*, pages 207-216, 1993.

[Chazelle *et al.*, 1986] B. Chazelle, R.L. Drysdale, D.T. Lee. Computing the largest empty rectangle. *SIAM Journal of Computing*, 15(1):300-315, 1986.

[Falkenhainer and Michalski, 1986] F. Falkenhainer and R. Michalski. Integrating quantitative and qualitative discovery: the ABACUS system. *Machine Learning*, 1(4):367-401, 1986.

[Fayyad *et al.*, 1996] U. Fayyad, G. Piatesky-Shapiro & P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, pages 37-54, 1996.

[Fisher, 1987] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139-172, 1987.

[Ku *et al.*, 1997]. L. P. Ku, B. Liu, and W. Hsu. *Discovering large empty maximal hyper-rectangles in multi-dimensional space.* Technical Report, Dept. of ISCS, National University of Singapore, 1997.

[Langley *et al.*, 1987] P. Langley, H. Simon, G.Bradshaw, and J. Zytkow, *Scientific discovery: computational explorations of the creative process*, The MIT press, 1987.

[Liu and Hsu, 1996] B. Liu and W. Hsu, Post-analysis of learned rules, *AAAI-96*, pages 828-834, 1996.

[Nordhausen and Langley, 1993] B. Nordhausen and P Langley. An integrated framework for empirical discovery. *Machine Learning* 12(1/2/3):17-48, 1993.

[Orlowski, 1990] M.Orlowski. A new algorithm for the largest empty rectangle problem. *Algorithmica*, 5:65-73, 1990.

[Overmars and Leeuwen, 1982] M. Overmars and J. Leeuwen. Dynamic multi-dimensional data structures based on Quad- and K-D trees. *Acta Informatica*, 17:267-285, 1982.

[Preparata and Shamos, 1985] F. Preparata and M.I Shamos. *Computational geometry-An introduction*. Springer,1985.

[Quinlan, 1992] J. Ross Quinlan. *C4.5*: *program for machine learning*. Morgan Kaufmann, 1992.

[Zytkow, 1990] J. Zytkow. Deriving basic laws by analysis of processes and equations. In P. Langley & J. Shrager (eds.), *Computational models of scientific discovery and theory formation*. Morgan Kaufmann, 1990.