Summarizing Data Cubes Using Blocks

Yeow Wei Choong, Anne Laurent, Dominique Laurent

Summarizing Data Cubes Using Blocks*

Yeow Wei Choong
HELP University College - Kuala Lumpur - MALAYSIA
choongyw@help.edu.my
Anne Laurent
LIRMM UMR CNRS 5506 - Université Montpellier II
Montpellier - FRANCE
laurent@lirmm.fr
Dominique Laurent
ETIS UMR CNRS 8051 - Université de Cergy-Pontoise
Cergy-Pontoise - FRANCE
dominique.laurent@dept-info.u-cergy.fr

Abstract

In the context of multidimensional data, OLAP tools are appropriate for the navigation in the data, aiming at discovering pertinent and abstract knowledge. However, due to the size of the data set, a systematic and exhaustive exploration is not feasible. Therefore, the problem is to design automatic tools to ease the navigation in the data and their visualization. In this paper, we present a novel approach allowing to build automatically blocks of similar values in a given data cube that are meant to summarize the content of the cube. Our method is based on a levelwise algorithm (*a la* Apriori) whose complexity is shown to be polynomial in the number of scans of the data cube. The experiments reported in the paper show that our approach is scalable, in particular in the case where the measure values present in the data cube are discretized using crisp or fuzzy partitions.

Keywords: Multidimensional Databases, OLAP, Data Summarization, Levelwise Algorithms, Fuzzy Partitions.

*Work partially supported by the STIC-Asia project *EXPEDO* (funded by the French Ministère des Affaires Étrangères and the French Embassy in Malaysia).

1 Introduction

As stated by Bill Inmon in 1990, "*A data warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process*" [13]. As data warehouses are devoted to intensive decision-oriented querying, classical relational database management systems are known to be not suitable in this framework. To cope with this problem, the multidimensional model of databases has been proposed by E.F. Codd more than 10 years ago in [7].

In the context of multidimensional databases, data are considered as belonging to multidimensional tables, the so-called *data cubes* or simply *cubes*, defined over several *dimensions* and in which *measure* values are associated to one value in each dimension. On-Line Analytical Processing (OLAP) has become a major research issue, aiming at providing users with tools for querying data cubes.

Querying a cube is known to be a tedious process because, as data are often voluminous, an exhaustive exploration is not possible. Therefore, it is often the case that users wish to have a rough idea of the content of a cube in order to identify relevant data. In other words, summarizing the content of a data cube is one of the major needs of users. The OLAP operators called *roll-up* and *drill-down* are commonly used to this end. These operators allow to explore the data cube according to different levels of granularity defined on dimensions: while rolling-up according to one or several dimensions displays the data at a lower level of details, drilling-down has the reverse effect of displaying the data at a higher level of details. However, it should be noticed that these operators work based on predefined hierarchies on dimensions, and thus, do not allow to summarize a data cube based on its actual content, *i.e.,* the measure values.

In this paper, we propose an approach to automatically summarize a data cube by computing sub-cubes, which we call *blocks*, that *mainly* contain the same measure value. It is important to note that in this work, we do not consider the option of computing blocks containing *exclusively* the same measure value, which is very restrictive and thus, would prevent from obtaining relevant summaries.

The way we characterize that a block b of a cube C *mainly* contains the measure value m can be outlined as follows: assuming two user-given thresholds σ and γ, called the *support* and the *confidence* thresholds, respectively, b *mainly* contains m if the ratio of the number of occurrences of m in b over the cardinality of C is greater than or equal to σ, and if the ratio of the number of occurrences of m in b over the cardinality of b is greater

than or equal to γ. These two ratios are called *support* and *confidence* of b for m, and as we shall see later, the support and the confidence thresholds are respectively related to the minimum size and to the purity of the block. Moreover, as measure values are numerical, it can be relevant to consider close values as equal. We take such an option into account in the computations of support and confidence by considering two kinds of partitioning of the set of measure values present in the data cube, namely *crisp* and *fuzzy* partitoning.

As in either case the computation of blocks as roughly described above is NP-hard, we propose a levelwise algorithm *a la* Apriori and the experiments reported in this paper show that our method is scalable even for data cubes with large cardinalities and large numbers of dimensions. However, it is important to note that the price to pay for scalability is the non completeness of our method, *i.e.,* a cube C may contain blocks with supports and confidences greater than or equal to the corresponding thresholds, but that are not output. We shall discuss this important point in details later in the paper.

The set of all blocks computed by our approach is considered as a *summary* of the cube. In our previous work, we have argued that blocks can be associated with *rules* (see [4]), and that they can serve as a basis for an efficient visualization of the cube (see [5]). It should be noticed that, since the computed blocks (obtained after partitioning or not) *mainly* contain a given measure value, it might be the case that two or more blocks overlap. This important feature of our approach is taken into account in [4] by considering fuzzy rules, and in [5] by defining a policy to display the most relevant block among all those that overlap. In this paper, we do not address the issues of computing rules or of visualizing the blocks. Instead, we focus on the computation of the blocks in the following respects:

1. Based on the fact that the method presented in [4] is not complete, we enhance our approach and we show some partial completeness results in this new framework.

2. As in practice, the measure values contained in a cube are numerical, we study the impact of discretizing these values, using crisp or fuzzy partitions.

3. We report experiments conducted on randomly generated data sets that show that our approach is still scalable for large data cubes with a high number of dimensions.

The following example illustrates our approach.

Example 1 *Let us consider the cube C displayed in Figure 1. This cube is defined over two dimensions, namely $CITY$ and $PRODUCT$, and contains measure values standing for the quantity of a given product sold in a given city. For instance, it can be seen that the quantity of product $P1$ sold in city $C1$ is 6 units.*

Considering a support threshold $\sigma = \frac{1}{12}$ and a confidence threshold $\gamma = \frac{2}{3}$, our approach generates the blocks as represented in Figure 1. These are defined as follows in our formalism:

- $b_1 = [C1, C1] \times [P1, P2]$ *for value 6, because the support and the confidence for 6 are respectively $\frac{1}{12}$ and 1,*
- $b_2 = [C1, C3] \times [P3, P4]$ *for value 8, because the support and the confidence for 8 are respectively $\frac{1}{6}$ and $\frac{2}{3}$,*
- $b_3 = [C3, C4] \times [P1, P3]$ *for value 5, because the support and the confidence for 5 are respectively $\frac{1}{6}$ and $\frac{2}{3}$,*
- $b_4 = [C4, C6] \times [P3, P4]$ *for value 2, because the support and the confidence for 2 are respectively $\frac{5}{24}$ and $\frac{5}{6}$.*

First, we note that, for the measure value 6, the block defined by $b = [C1, C2] \times [P1, P2]$ has a support and a condidence equal to $\frac{1}{8}$ and $\frac{3}{4}$, respectively. Therefore, this illustrates the non completeness of our approach since $\frac{1}{8} \geq \sigma$ and $\frac{3}{4} \geq \gamma$.

Now, assume that instead of integer values, the cube of Figure 1 contains numbers that represent the quantity of sales in thousands of units for each city and each product. In this case, it is likely that, for instance, in place of 6 for city $C1$ and products $P1$ and $P2$, the cube contains values 5.996 and 6.002. In this case, computing blocks based on the exact *measure value would not give the block b_1, although the corresponding measure values are close to each other. To cope with this problem, we consider that the set of measure values can be partitioned so as to yield relevant blocks. For instance, in our example, this partitioning could be defined by considering for every integer X that measure values in $[(X-1).500, X.500[$ are equal to X. Moreover, we generalize partitionings to* fuzzy *partitionings, so as to consider that $[(X-1).500, X.500]$ and $[(X-2).500, (X+1).500]$ are respectively the* support *and the* kernel *of the bin corresponding to the fuzzy notion* about X thousands*.*

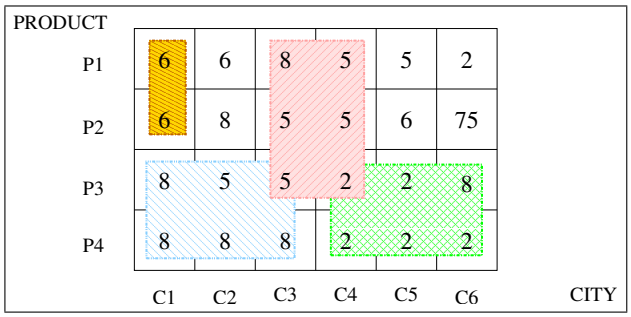

Figure 1: A Data Cube and the Associated Blocks

We mention that building blocks from a data cube facilitates its visualization, and the more the relevance of the blocks, the better the representation quality.

The issue of data cube representation has been addressed in [6], where the authors show that a data cube has several representations among which some are more relevant than others, according to user-specified criteria. In [6], the criterion is that the measure is ordered in an increasing manner over all dimensions, and representations optimizing this criterion are studied.

In the present paper, we consider as relevant the repesentations where same measure values are grouped to form blocks as large as possible. However, in this paper, contrary to [6], our goal is not to compute relevant representations according to this criterion; in what follows, the representation of the data cube is assumed to be fixed, and the blocks are computed in this particular representation. In this setting, it is relevant to use the blocks computed by our approach in order to assess the quality of the representation of the cube. More precisely, this quality can be related to the following criteria:

- the proportion of elements in the cube that are included in the blocks (the higher the proportion, the less elements not covered by the rules),
- the number of blocks built (the more blocks there are, the more heterogeneous data are),
- the number of blocks in comparison with the number of measure values (if several blocks are built for the same measure value m, then the different occurrences of m are displayed in non contiguous areas of the cube),

- the number of overlappings between blocks and their sizes (the higher the number of overlapping blocks, the more mixed the data).

The paper is organized as follows: Section 2 introduces the basic definitions concerning multidimensional databases and blocks, including interval based- and fuzzy interval based-blocks. Section 3 presents the algorithms to build blocks from multidimensional databases and the corresponding complexity issues, as well as a thourough discussion about the completeness of our approach. Section 4 presents a method based on cell neighborhood to improve the completeness of our approach. Section 5 reports on the experiments performed on synthetic data and on a real data set. Section 6 presents the related work from the literature, and in Section 7, we conclude the paper and we outline further research directions.

2 Multidimensional Databases and Blocks

2.1 Basic Definitions

Although no consensual definition has emerged for now concerning data representation and manipulation, a multidimensional database is generally defined as being a set of *data cubes* (hereafter *cubes*). A cube can be seen as a set of cells and a cell represents the association of a *measure* value with one member in each *dimension.* Moreover, *hierarchies* may be defined over dimensions, for instance to describe sales in function of states and not of cities. Since hierarchies are not considered in the present paper, we do not include these in our definition.

Definition 1 - Cube. *A k-dimensional* cube*, or simply a cube, C is a tuple $\langle dom_1, \ldots, dom_k, dom_m, m_C \rangle$ where*

- *$dom_1, \ldots, dom_k$ are k finite sets of symbols for the members associated with dimensions $1, \ldots, k$ respectively,*
- *let dom_{mes} be a finite totally ordered set of measures. Let $\perp \notin dom_{mes}$ be a constant (to represent null values). Then $dom_m = dom_{mes} \cup \{\perp\}$,*
- *m_C is a mapping: $dom_1 \times \ldots \times dom_k \rightarrow dom_m$.*

A cell c of a k-dimensional cube C is a $(k+1)$-tuple $\langle v_1, \ldots, v_k, m \rangle$ such that for every $i = 1, \ldots, k$, v_i is in dom_i and where $m = m_C(v_1, \ldots, v_k)$. m is called the *content* of c and c is called an m-cell. Moreover, for every

$i = 1, \ldots k$, dom_i is called the *domain* of dimension d_i and an element v_i in dom_i is called a *member value*.

We recall from [6] that a given cube can be represented in several ways, based on the ordering of the member values in each set dom_i. For example, figures 1 and 2 display two different representations of the cube C considered in Example 1. Although we do not consider the issue of computing particular representations, the notion of representation, as defined below, plays an important role in the present approach.

PRODUCT

P4	8	8	8	2	2	2	
P2	5	6	8	5	6	75	
P1	8	6	6	5	5	2	
P3	5	8	5	2	2	8	
	C3	C1	C2	C4	C5	C6	**CITY**

Figure 2: Another Representation of the Cube of Figure 1

Definition 2 - Representation. *A* representation *of a k-dimensional cube C is a set $R = \{rep_1, \ldots, rep_k\}$ where for every $i = 1, \ldots, k$, rep_i is a one-to-one mapping from dom_i to $\{1, \ldots |dom_i|\}$.*

In this paper, we consider a *fixed* k-dimensional cube C and a *fixed* representation of C, $R = \{rep_1, \ldots, rep_k\}$.

Now, given a fixed representation of C, $R = \{rep_1, \ldots, rep_k\}$, for every dimension d_i, v_1 and v_2 in dom_i are said to be *contiguous* if $rep_i(v_1)$ and $rep_i(v_2)$ are consecutive integers, *i.e.* if $|rep_i(v_1) - rep_i(v_2)| = 1$. Moreover, if $rep_i(v_1) \leq rep_i(v_2)$, the *interval* $[v_1, v_2]$ is the set of all contiguous values between v_1 and v_2, *i.e.,* $[v_1, v_2] = \{v \in dom_i \mid rep_i(v_1) \leq rep_i(v) \leq rep_i(v_2)\}$.

2.2 Blocks

In our approach, we define a block of C as follows.

Definition 3 - Block. *A* block *b is a set of cells defined over a k-dimensional cube C by $b = \delta_1 \times \ldots \times \delta_k$ where δ_i are intervals of contiguous values from dom_i, for $i = 1, \ldots, k$.*

Note that in the previous definition a block is specified by exactly one interval per dimension. In the case where an interval would not be specified on a

dimension d_i, the corresponding interval δ_i is set to $[rep_i^{-1}(1), rep_i^{-1}(|dom_i|)]$, which is denoted by ALL_i.

For example, if we consider the cube of Figure 1, the interval $[C1, C3]$ is associated with the block $[C1, C3] \times ALL_{PRODUCT}$ where $ALL_{PRODUCT}$ is the interval $[P4, P1]$.

Definition 4 - Block Overlapping. *Two blocks b and b' are said to* overlap *if they share at least one cell,* i.e., *if $b \cap b' \neq \emptyset$.*

It is easy to see that two blocks $b = \delta_1 \times \ldots \times \delta_k$ and $b' = \delta'_1 \times \ldots \times \delta'_k$ overlap if and only if for every dimension d_i, $\delta_i \cap \delta'_i \neq \emptyset$. As stated in the following definition, in our formalism, a slice is defined as a particular block.

Definition 5 - Slice. *Let v_i be a member value in dom_i. The* slice *of C associated with v_i, denoted by $\mathcal{T}(v_i)$, is the block $\delta_1 \times \ldots \times \delta_k$ such that $\delta_i = \{v_i\}$, and for all $j \neq i$, $\delta_j = ALL_j$.*

Given two member values v_1 and v_2 in the same domain dom_i, the slices $\mathcal{T}(v_1)$ and $\mathcal{T}(v_2)$, are said to be contiguous *if v_1 and v_2 are contiguous,* i.e., *if $|rep_i(v_1) - rep_i(v_2)| = 1$.*

Referring to Figure 1, the slices $\mathcal{T}(P3)$ and $\mathcal{T}(P4)$ are contiguous since $P3$ and $P4$ are contiguous in the considered representation.

It is important to note that the notion of contiguous cells (or slices) depends on the representation of the cube that is being considered. Indeed, two member values (or slices) can be contiguous in a given representation of C but *not* contiguous in another representation of C. For instance, considering the cube C of Example 1, the member values $C2$ and $C3$ are contiguous in the representation of C displayed in Figure 1, but are *not* contiguous in the representation of C displayed in Figure 2.

We now define the following *specificity relation* between blocks of a given cube C.

Definition 6 - Specificity Relation. *Let $b = \delta_1 \times \ldots \times \delta_k$ and $b' = \delta'_1 \times \ldots \times \delta'_k$ be two blocks. b' is said to be* more specific *than b, denoted by $b \sqsubseteq b'$, if for every $i = 1, \ldots, k$, $\delta_i \neq \delta'_i \Rightarrow \delta_i = ALL_i$.*

For instance, in the cube of Figure 1, for $b = [C1, C3] \times ALL_{PRODUCT}$ and $b' = [C1, C3] \times [P3, P4]$, we have $b \sqsubseteq b'$ since the intervals defining b and b' satisfy the above definition.

It can be seen that the relation $\sqsubseteq$ as defined above is a partial ordering over the set of all blocks of the cube C. Given a set of blocks B, the

maximal (respectively minimal) elements of B are said to be *most specific* (respectively *most general*) in B. Most specific blocks and most general blocks are called *MS-blocks* and *MG-blocks*, respectively.

Moreover, it can be easily shown that if b and b' are two blocks such that $b \sqsubseteq b'$, then $b' \subseteq b$.

2.3 Support and Confidence of a Block

The support and the confidence of a given block b are defined according to the content of the cells in b. In order to comply with our discussion in the introductory section (see Example 1), we consider three different criteria in this respect: (i) single measure values, (ii) partition based measure values, and (iii) fuzzy partition based measure values.

In order to define the support and the confidence of a block for a given measure value, we introduce the following notation: let b be a block and m a measure value, $Count(b, m)$ denotes the number of m-cells in b, and $|b|$ denotes the total number of cells in b. In particular, $|C|$ denotes the total number of cells in the whole cube C.

Definition 7 - Support. *The* support *of a block b from C for a measure value m is defined as:*

$$supp(b, m) = \frac{Count(b, m)}{|C|}.$$

Considering a user-given minimum support threshold σ and a measure value m, a block b such that $supp(b, m) \geq \sigma$ is called σ-frequent for m.

Definition 8 - Confidence. *The* confidence *of a block b for a measure value m is defined as:*

$$conf(b, m) = \frac{Count(b, m)}{|b|}.$$

As argued in the introductory section, considering separately all measure values present in the cube can lead to consider non relevant blocks, which will be very small. For instance, in a cube containing billions of cells and where the measure values range from 1 to $1,000$ with very few repetitions, almost $1,000$ values have to be considered separately. Alternatively, in this case, 5 and 5.2 are likely to be considered as similar measure values and thus, should be processed as such.

In order to take this important point into account, we propose two ways to build blocks, based on *intervals* of measure values, on the one hand, and on *fuzzy intervals*, on the other hand. In these cases, the support and the confidence of a block are defined as follows.

Definition 9 - Interval Support and Confidence. *The* interval support *of a block b in C for a measure value interval $[m1, m2]$ is defined as:*

$$i\_supp(b, [m1, m2]) = \frac{iCount(b, [m1, m2])}{|C|}$$

where $iCount(b, [m1, m2])$ is the number of m-cells in b such that $m \in [m1, m2]$. Similarly, the interval confidence *of b for $[m1, m2]$ is defined as:*

$$i\_conf(b, [m1, m2]) = \frac{iCount(b, [m1, m2])}{|b|}.$$

When considering fuzzy intervals instead of intervals, counting cells in a block b with respect to a fuzzy interval φ can be computed according to the following methods ([9]):

1. The Σ-*count* sums up the membership degrees of all cells of b.
2. The *threshold-count* counts those cells of b whose membership degree is greater than a user-defined threshold.
3. The *threshold-Σ-count* sums up those cell membership degrees that are greater than a user-defined threshold.

In what follows, given a fuzzy interval φ and a cell c with content m, we denote by $\mu(c, \varphi)$ the membership value of m in φ. Moreover, given a block b, $\Sigma^f_{c \in b} \mu(c, \varphi)$ denotes the count of cells in b whose content is in φ, according to one of the three counting methods mentioned above. In this case, the support and confidence of a block b are defined as follows.

Definition 10 - Fuzzy Support and Confidence. *The* fuzzy support *of a block b in C for a fuzzy interval φ is defined as:*

$$f\_supp(b, \varphi) = \frac{fCount(b, \varphi)}{|C|}$$

where $fCount(b, \varphi) = \Sigma^f_{c \in b} \mu(c, \varphi)$. Similarly, the fuzzy confidence *of b for φ is defined as:*

$$f\_conf(b, \varphi) = \frac{fCount(b, \varphi)}{|b|}.$$

2.4 Properties

We first show that the support is anti-monotonic with respect to $\sqsubseteq$, in either of the three cases defined above.

Proposition 1 *For all blocks b and b' such that $b \sqsubseteq b'$, we have:*

1. *For every measure value m, $supp(b', m) \leq supp(b, m)$.*
2. *For every interval of measure values $[m_1, m_2]$, $i\_supp(b', [m1, m2]) \leq i\_supp(b, [m1, m2])$.*
3. *For every fuzzy interval of measure values φ, $i\_supp(b', \varphi) \leq i\_supp(b, \varphi)$.*

PROOF: If b and b' are two blocks such that $b \sqsubseteq b'$ then we have that $b' \subseteq b$. Moreover, in this case, we have $Count(b, m) \leq Count(b', m)$, $iCount(b, [m_1, m_2]) \leq iCount(b', [m_1, m_2])$ and for every fuzzy counting method given above, $fCount(b, \varphi) \leq fCount(b', \varphi)$. Therefore, the proposition follows from the definitions of the support, which completes the proof. □

In our levelwise algorithm for computing blocks given in the next section, Proposition 1 is used in the following way in the case of single measure values (the other two cases being similar): given a block b, a measure value m and a support threshold σ, b is *not* σ-frequent for m if there exists a block b' such that $b' \sqsubset b$ and b' is not σ-frequent for m.

The following proposition, of which the easy proof is omitted, shows that the support (respectively confidence) of blocks based on intervals and fuzzy intervals are greater than the support (respectively confidence) blocks based on single values.

Proposition 2 *For every block b and every measure value m, let m_1 and m_2 be measure values such that $m \in [m_1, m_2]$, and let φ be a fuzzy interval such that $kernel(\varphi) = [m_1, m_2]$. Then, for any of the three fuzzy counting methods Σ^f, we have:*

$$Count(b, m) \leq iCount(b, [m_1, m_2]) \leq fCount(b, \varphi).$$

As a consequence:

- $supp(b, m) \leq i\_supp(b, [m_1, m_2]) \leq f\_supp(b, \varphi)$ *and*
- $conf(b, m) \leq i\_conf(b, [m_1, m_2]) \leq f\_conf(b, \varphi)$.

3 Algorithms

In this paper, our goal is to discover blocks whose support and confidence are greater than or equal to user specified thresholds. To this end, our method is based on a levelwise Apriori-like algorithm [2], for scalability reasons. In this section, we first present the algorithms for the discovery of blocks in the case of single values, and then we show how the cases of (fuzzy) interval based measure values can be processed.

Roughly speaking, in the case of single measure values, our method works as follows: for every single measure value m in C do the following

1. For every $i = 1, \ldots, k$, compute all maximal intervals I of values in dom_i such that, for every v in I, the slice $\mathcal{T}(v)$ is σ-frequent for m.

2. Combine the intervals in a levelwise manner as follows: at level l ($2 \leq l \leq k$), compute all σ-frequent blocks $b = \delta_1 \times \ldots \times \delta_k$ such that exactly l intervals defining b are different than ALL. Assuming that all blocks σ-frequent for m have been computed at the previous levels, this step can be achieved in much the same way as frequent itemsets are computed in the algorithm Apriori.

3. Considering the set of all blocks computed in the previous step, sort out those that are not MS-blocks and those having a confidence for m less than γ.

It should be clear from Definition 7 that a block can be frequent only if it contains at least $\sigma \,.\, |C|$ cells. Similarly, it follows from Definition 8 that, for a given confidence threshold γ, a block b is output only if it contains at least $\gamma \,.\, |b|$ cells containing the measure value m. Therefore, when fixing the support and the confidence thresholds, the user actually determines thresholds concerning the size and the purity of the blocks (s)he wishes to obtain.

3.1 Block Generation for Single Measure Values

In the following algorithms, MS- or MG-blocks are computed according to the user's specification. Algorithm 1 performs step 1 above, while Algorithm 2 performs steps 2 and 3 above.

Referring to the cube of Example 1, the supports for measure value 8 of all slices of the cube is displayed in Figure 3, while Figure 1 depicts all blocks output by Algorithm 2.

Algorithm 1: Computation of $\mathcal{L}_1(m)$

Data: A k-dimensional data cube C, a measure value m, a support threshold σ

Result: The set of intervals $\mathcal{L}_1(m)$, the set of corresponding blocks $\mathcal{B}_1(m)$

$\mathcal{L}_1(m) \leftarrow \emptyset$

foreach *dimension* d_i, $i = 1, \ldots, k$ **do**

  $int(m,i) \leftarrow \emptyset$

  $currentInterval \leftarrow [NIL, NIL]$

  foreach $j = 1, \ldots, |dom_i|$ **do**

    $s \leftarrow supp(\mathcal{T}(rep_i^{-1}(j)), m)$

    if $s < \sigma$ **then**

      if $currentInterval = [\alpha, NIL]$ *where* $\alpha \neq NIL$ **then**

        /* close the current interval at position $j-1$, and set the current interval to the empty interval */

        $int(m,i) \leftarrow int(m,i) \cup \{[\alpha, rep_i^{-1}(j-1)]\}$

        $currentInterval \leftarrow [NIL, NIL]$

    else

      if $currentInterval = [NIL, NIL]$ **then**

        /* start a new current interval at position j */

        $currentInterval \leftarrow [rep_i^{-1}(j), NIL]$

      if $j = |dom_i|$ *and* $currentInterval = [\alpha, NIL]$ *where* $\alpha \neq NIL$ **then**

        $int(m,i) \leftarrow int(m,i) \cup \{[\alpha, rep_i^{-1}(j)]\}$

  $\mathcal{L}_1(m) \leftarrow \mathcal{L}_1(m) \cup int(m,i)$

$\mathcal{B}_1(m) \leftarrow \{b = \delta_1 \times \ldots \times \delta_k \mid (\exists i)(\delta_i \in int(m,i))$ and $(\forall j \neq i)(\delta_j = ALL_j)$ and $conf(b,m) \geq \gamma\}$

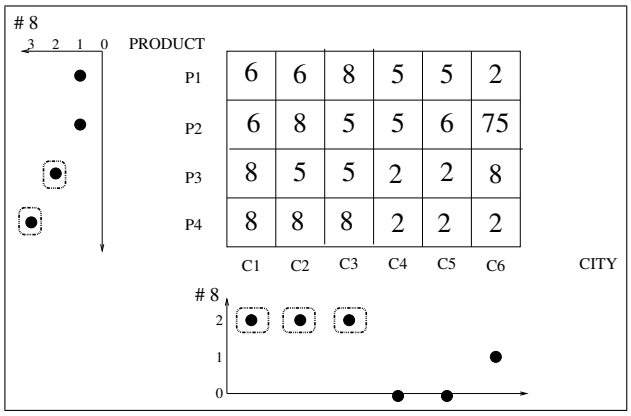

Figure 3: Occurences of measure value 8

$b_1 = [P1, P2] \times [C1, C1]$ for value 6, $b_2 = [P3, P4] \times [C1, C3]$ for value 8
$b_3 = [P1, P3] \times [C3, C4]$ for value 5, $b_4 = [P3, P4] \times [C4, C6]$ for value 2

Note that there are two overlappings: one between b_2 and b_3, and one between b_3 and b_4.

3.2 Processing Interval-Based Blocks

In this section, we consider the computation of blocks when intervals and fuzzy intervals are considered, instead of single measure values. In this case, the following modifications must be made in the two algorithms given previously:

1. The supports and confidences must be computed accordingly. That is, in the case of interval based measure values, *supp* and $conf$ must be replaced by i_supp and i_conf, respectively, and in the case of fuzzy interval based measure values, *supp* and $conf$ must be replace by f_supp and f_conf, respectively.

2. In Algorithm 2, the most outer loop must range over the set of intervals (fuzzy or not according to the considered case), instead of over the set of all single measure values.

On the other hand, when considering intervals or fuzzy intervals, a preprocessing task must be applied on the data in order to discretize the measure values into (fuzzy) intervals. This discretization process can be auto-

Algorithm 2: Discovery of MS-blocks

Data: A k-dimensional data cube C, a measure value m, a support threshold σ, and a confidence threshold γ.

Result: The set of blocks $\mathcal{B}$ associated with C

foreach *measure value* m *from* C **do**

  Compute $\mathcal{L}_1(m)$ and $\mathcal{B}_1(m)$

  for $l = 2$ *to* k **do**

    $\mathcal{B}_l(m) \leftarrow \emptyset$

    Generate from $\mathcal{L}_{l-1}$ all candidates $\delta_{i_1} \times \ldots \times \delta_{i_l}$ such that $(\forall p, p' \in [1, l])(i_p \neq i_{p'})$. Let $\mathcal{L}_l(m)$ be this set.

    Pruning: Delete from $\mathcal{L}_l(m)$ all candidates $\delta_{i_1} \times \ldots \times \delta_{i_l}$ such that $(\exists p \in \{1, \ldots, l\})(\delta_{i_1} \times \ldots \times \delta_{i_{p-1}} \times \delta_{i_{p+1}} \times \ldots \times \delta_{i_l} \notin \mathcal{L}_{l-1})$

    foreach *remaining candidate* $\delta_{i_1} \times \ldots \times \delta_{i_l}$ **do**

      Let b be the block $\delta_1 \times \ldots \times \delta_k$ where $\delta_p = \delta_{p_j}$ if dimension d_p has been treated and $\delta_p = ALL_p$ otherwise

      if $supp(b, m) < \sigma$ **then**

        Remove $\delta_{i_1} \times \ldots \times \delta_{i_l}$ from $\mathcal{L}_l(m)$

      else

        if $conf(b, m) \geq \gamma$ **then** $\mathcal{B}_l(m) \leftarrow \mathcal{B}_l(m) \cup \{b\}$

  $\mathcal{B}(m) \leftarrow \{b \in \bigcup_{l=1}^{l=k} \mathcal{B}_l(m) \mid b \text{ is an MS-block}\}$

  /* If MG-blocks are to be computed then replace MS with MG is the previous statement */

$\mathcal{B} \leftarrow \bigcup_m \mathcal{B}(m)$

matically performed, provided that the user defines the number of intervals (s)he wants to consider.

Denoting by N this number of intervals, and assuming that m_b (respectively m_t) is the bottom value (respectively the top value) of the measure values, $[m_b, m_t]$ can be divided into n intervals either in an equi-width manner (*i.e.,* the widths of all intervals are equal), or in an equi-depth manner (*i.e.,* all intervals cover the same number of cells). These intervals are denoted by $[bot_i, top_i]$ for $i = 1, \ldots, N$, and we note that for every $i = 2, \ldots, N$, $bot_i = top_{i-1}$.

Then, in the case of fuzzy intervals, we consider N trapezoidal membership functions $\mu_1, \ldots, \mu_N$ such that:

- $[bot_1, top_1]$ and $[bot_1, top_2]$ are respectively the kernel and the support of μ_1,
- $[bot_N, top_N]$ and $[bot_{N-1}, top_N])$ are respectively the kernel and the support of μ_n,
- $[bot_{i-1}, top_{i+1}]$ and $[bot_i, top_i]$ are respectively the support and kernel of μ_i, for $i = 2, \ldots, N-1$.

It should also be noted from Proposition 2 that blocks based on intervals and fuzzy intervals of measure values are larger than blocks based on single values.

3.3 Complexity Issues

In this section, we show that our method for computing blocks is polynomial in time with respect to the size of the cube C, but not complete, in the sense that blocks that fullfil the threshold conditions might not be output by our algorithms.

Let m be a measure value. In Algorithm 1, the cube is scanned once for each dimension d_i. Thus, this step requires k scans of the cube C. Regarding the complexity of Algorithm 2, at each level, the whole cube is scanned at most once, since the intervals produced by Algorithm 1 for a given measure value do not overlap. As in Algorithm 2, at most k iterations are processed, its execution requires at most k scans of the cube C. As a consequence, the computation of all frequent blocks associated to a given measure value m is in $O(k \,.\, |C|)$. As computing the confidence of a block does not require the scanning of the cube (because the size of a block is the product of the sizes of the intervals defining this block), the time complexity of Algorithm 2 is in $O(k \,.\, |C|)$.

Hence, in the case of single measure values, the computation of all blocks is in $O(k \,.\, |C|^2)$, because C contains at most $|C|$ distinct measure values. We note that although polynomial, the complexity of our method is not linear. On the other hand, in the cases of (fuzzy) intervals of measure values, if we denote by N the number of these intervals, then the computation of blocks is in $O(k \,.\, |C| \,.\, N)$, *i.e.,* linear with respect to the size $|C|$ of C. The experiments reported in Section 5 show the influence of the three factors (*i.e.,* k, $|C|$ and N) on the computation time.

As mentioned in the introductory section, the general problem of computing blocks of in a cube is known to be NP-hard, and the polynomial method proposed in the paper computes an approximation of the solution, meaning that our method is not complete. As an example, we recall from Example 1 that the block $b = [C1, C2] \times [P1, P2]$ is not an output, although its support and confidence are greater than the support and confidence thresholds, respectively. This is due to the fact that the slice $\mathcal{T}(C2)$ is not frequent for 6, which implies that the interval $[C1, C2]$ is not computed by Algorithm 1.

However, the following example shows that this is not the only reason for non completeness. More precisely, even if completeness would mean the computation of *all* blocks $b = \delta_1 \times \ldots \times \delta_k$ such that, for a given measure value m,

1. $supp(b, m) \geq \sigma$ and $conf(b, m) \geq \gamma$, *and*
2. $(\forall i = 1, \ldots, k)(\forall v \in \delta_i)(supp(\mathcal{T}(v), m) \geq \sigma)$

then Example 2 below shows that our method is not complete either.

Example 2 *Consider the cube C as shown in Figure 3.3 in which blocks are to be computed according to support and confidence thresholds equal to* $1/12$ *and* 80%*, respectively.*

The computation of $\mathcal{L}_1(8)$ using Algorithm 1 returns the intervals $[C1, C6]$ and $[P1, P4]$ because all slices contain at least two 8-cells. However, since the confidence for 8 of the block $[C1, C6] \times [P1, P4]$ is equal to $15/24$, which is less than 80%, this block is not computed by Algorithm 2. On the other hand, it is easy to see that the two 8-blocks $[C1, C3] \times [P3, P4]$ and $[C4, C6] \times [P1, P2]$ satisfy the two items above.

In the next section, we study how to modify Algorithm 1 so as to take into account situations as in Example 2 above. Then we study the completeness of this modified method.

PRODUCT

P1	5	5	5	8	8	8	
P2	5	5	5	8	8	8	
P3	8	8	8	6	6	6	
P4	8	8	8	6	6	6	
	C1	C2	C3	C4	C5	C6	**CITY**

Figure 4: The Cube for Example 2

4 Refining the Computation of Blocks

In this section, we take into account the cell neighborhoods in order to enhance the completeness of our method.

4.1 Modified Computation of Blocks

In this section, we consider single measure vales. The modified algorithm (referred to as Algorithm 3) generates intervals of member values based not only on the support but also on the *neighbors* of the cells. Intuitively, cells are considered as neighbors if they share one side in the representation. For instance, in Figure 3.3 the cells $\langle P2, C3, 5\rangle$ and $\langle P2, C4, 8\rangle$ are neighbors.

Definition 11 - Cell Neighborhood. *Two distinct cells* $c = \langle v_1, \ldots, v_k, m\rangle$ *and* $c' = \langle v'_1, \ldots, v'_k, m'\rangle$ *are* neighbors *if there exists a unique* i_0 *in* $\{1, \ldots, k\}$ *such that:*

- $|rep_{i_0}(v_{i_0}) - rep_{i_0}(v'_{i_0})| = 1$ *and*
- *for every* $i = 1, \ldots, k$ *such that* $i \neq i_0$, $v_i = v'_i$.

We note that in a k-dimensional cube, a cell has at most $2 \,.\, k$ neighbors. Moreover, considering a slice $\mathcal{T}(v)$ where v is a member value of the domain dom_i of dimension i, let v^- and v^+ be the member values of dom_i such that $rep_i(v^-) = rep_i(v) - 1$ and $rep_i(v^+) = rep_i(v) + 1$, respectively.

Clearly, every cell c in $\mathcal{T}(v)$ has *exactly one* neighbor in each of the slices $\mathcal{T}(v^-)$ and $\mathcal{T}(v^+)$. Given a measure value m, we denote by $n(v^-, m)$, respectively $n(v^+, m)$, the number of m-cells in $\mathcal{T}(v)$ whose neighbor in $\mathcal{T}(v^-)$, respectiveley in $\mathcal{T}(v^+)$, is also an m-cell. Then, we define $neighbors(v^-, m)$ and $neighbors(v^+, m)$ as follows:

$$neighbors(v^-, m) = \frac{n(v^-,m)}{Count(\mathcal{T}(v),m)} \quad \text{and} \quad neighbors(v^-, m) = \frac{n(v^+,m)}{Count(\mathcal{T}(v),m)}.$$

Intuitively, $neighbors(v^-, m)$ and $neighbors(v^+, m)$ are respectively the ratios of m-cells in a given slice having m-cells as neighbors in the previous slice, respectively in the next slice.

Based on this notation, our method works roughly as follows: We assume that, in addition to the support and confidence thresholds, we are given a new threshold called the *neighbor threshold*, denoted by ν. When scanning dimension i for a given measure value m, let v be the member in dom_i of which the slice is being considered. If an interval of the form $[V, NIL]$ where $V \neq NIL$ is under construction and if the support of $\mathcal{T}(v)$ is greater than or equal to the support threshold, then:

- If $neighbors(v^-, m) < \nu$, then the interval $[V, v^-]$ is output and the computation of the new interval $[v, NIL]$ is considered.
- If $neighbors(v^+, m) < \nu$, then the interval $[V, v]$ is output and the computation of the new interval $[v^+, NIL]$ is considered.
- Otherwise, the next slice, *i.e.,* the slice defined by v^+, is considered for the interval $[V, NIL]$.

In the remainder of the paper, we call Algorithm 2.1 the algorithm obtained from Algorithm 2 by replacing Algorithm 1 with Algorithm 3 for the computation of $\mathcal{L}_1$.

Example 3 *We illustrate Algorithm 3 using the cube of Figure 3.3 and we consider the same thresholds as in Example 2, that is:* $\sigma = 1/12$ *and* $\gamma = 80\%$. *Moreover, let the neighbor threshold* ν *be* 60%.

In this case, for dimension $CITY$ *and measure value* 8, *Algorithm 3 starts with* $[NIL, NIL]$ *as the value for currentInterval and processes the first slice* $\mathcal{T}(C1)$. *As its support is greater than* $1/12$, *and as* $j = 1$, *only* n^+ *is computed, and is found equal to* 1. *Therefore the value of currentInterval is set to* $[1, NIL]$ *and the next slice,* i.e., $\mathcal{T}(C2)$ *is processed. In this case,* n^+ *and* n^- *are computed and both are found equal to* 1. *Thus, the slice* $\mathcal{T}(C3)$ *is processed.*

At this stage, we find $n^- = 1$ *and* $n^+ = 0$. *Since* $0 \leq \nu$, *the interval* $[C1, C3]$ *is output, the value of currentInterval is set to* $[C4, NIL]$ *and the slice* $\mathcal{T}(C4)$ *is processed. Now, we find* $n^- = 0$ *and* $n^+ = 1$. *As in this case,* $j = \alpha$ *in Algorithm 3, no computation is done and the slice* $\mathcal{T}(C5)$ *is processed, which does not lead to any change. Finally the processing of the slice* $\mathcal{T}(C6)$ *results in the interval* $[C4, C6]$, *since* $|dom_{CITY}| = 6$.

It can be seen that, for dimension $PRODUCT$ *and measure value* 8, *the computation is similar and outputs the two intervals* $[P1, P2]$ *and* $[P3, P4]$.

Algorithm 3: Modified Computation of $\mathcal{L}_1(m)$

Data: A k-dimensional data cube C, a measure value m, a support threshold σ, a neighbor threshold ν

Result: The set of intervals $\mathcal{L}_1(m)$, the set of corresponding blocks $\mathcal{B}_1(m)$

$\mathcal{L}_1(m) \leftarrow \emptyset$

foreach *dimension* d_i, $i = 1, \ldots, k$ **do**

  $int(m,i) \leftarrow \emptyset$

  $currentInterval \leftarrow [NIL, NIL]$

  foreach $j = 1, \ldots, |dom_i|$ **do**

    $s \leftarrow supp(\mathcal{T}(rep_i^{-1}(j)), m)$

    if $j < |dom_i|$ **then**

      $n^+ \leftarrow neighbors(rep_i^{-1}(j+1), m)$

    if $j > 1$ **then**

      $n^- \leftarrow neighbors(rep_i^{-1}(j-1), m)$

    if $s < \sigma$ **then**

      if $currentInterval = [\alpha, NIL]$ *where* $\alpha \neq NIL$ **then**

        $int(m,i) \leftarrow int(m,i) \cup \{[\alpha, rep_i^{-1}(j-1)]\}$

        $currentInterval \leftarrow [NIL, NIL]$

    else

      /* $s = supp(\mathcal{T}(rep_i^{-1}(j)), m) \geq \sigma$ */

      if $currentInterval = [NIL, NIL]$ **then**

        /* start a new current interval at position j */

        $currentInterval \leftarrow [rep_i^{-1}(j), NIL]$

      if $currentInterval \neq [NIL, NIL]$ **then**

        /* $currentInterval = [\alpha, NIL]$ */

        if $j > 1$ *and* $j \neq \alpha$ **then**

          if $n^- < \nu$ **then**

            $int(m,i) \leftarrow int(m,i) \cup \{[\alpha, rep_i^{-1}(j-1)]\}$

            $currentInterval \leftarrow [rep_i^{-1}(j), NIL]$

        if $j < |dom_i|$ **then**

          if $n^+ < \nu$ **then**

            $int(m,i) \leftarrow int(m,i) \cup \{[\alpha, rep_i^{-1}(j)]\}$

            $currentInterval \leftarrow [j+1, NIL]$

        else

          /* $j = |dom_i|$ */

          $int(m,i) \leftarrow int(m,i) \cup \{[\alpha, rep_i^{-1}(j)]\}$

  $\mathcal{L}_1(m) \leftarrow \mathcal{L}_1(m) \cup int(m,i)$

$\mathcal{B}_1(m) \leftarrow \{b = \delta_1 \times \ldots \times \delta_k \mid (\exists i)(\delta_i \in int(m,i))$ and $(\forall j \neq i)(\delta_j = ALL_j)$ and $conf(b,m) \geq \gamma\}$

Therefore, for measure value 8*, we obtain* $\mathcal{L}_1(8) = \{[C1, C3],\ [C4, C6],\ [P1, P2],\ [P3, P4]\}$.

Regarding the computation of the blocks, it is easy to see that, in this example, Algorithm 2.1 computes the two blocks $[C1, C3] \times [P3, P4]$ *and* $[C4, C6] \times [P1, P2]$*, since their confidence is* 1.

4.2 Neighbors and Intervals

In this section, we study the impact of considering neighbors for blocks defined according to intervals or fuzzy intervals of measure values. We note that, in the case of (fuzzy) intervals, dealing with cell neighborhood does not affect Algorithm 3, provided that counting neighbors is defined accordingly.

In the case of intervals of measure values, given a member value v and an interval $[m_1, m_2]$, we denote by $i\_n(v^-, [m_1, m_2])$, respectively $i\_n(v^+, [m_1, m_2])$, the number of cells in $\mathcal{T}(v)$ whose content is in $[m_1, m_2]$ and whose neighbor in $\mathcal{T}(v^-)$, respectively in $\mathcal{T}(v^+)$, is a cell whose content is in $[m_1, m_2]$. Then, $i\_neighbors(v^-, [m_1, m_2])$ and $i\_neighbors(v^+, [m_1, m_2])$ are defined as follows:

$$i\_neighbors(v^-, [m_1, m_2]) = \frac{i\_n(v^-, [m_1, m_2])}{iCount(\mathcal{T}(v), [m_1, m_2])} \text{ and}$$

$$i\_neighbors(v^+, [m_1, m_2]) = \frac{i\_n(v^+, [m_1, m_2])}{iCount(\mathcal{T}(v), [m_1, m_2])}.$$

Similarly to the case of intervals, dealing with cell neighborhood for fuzzy intervals is modified as follows. We first recall from [14] that assessing the fact that the contents of two cells c and c' both belong to a given fuzzy interval φ, denoted by $\mu(c, \varphi) \otimes \mu(c', \varphi)$, can be done according to the following *t-norms*:

1. Probalistic t-norm: $\mu(c, \varphi) \otimes \mu(c', \varphi) = \mu(c, \varphi).\mu(c', \varphi)$
2. Zadeh's t-norm: $\mu(c, \varphi) \otimes \mu(c', \varphi) = \min(\mu(c, \varphi), \mu(c', \varphi))$
3. Lukasiewicz's t-norm: $\mu(c, \varphi) \otimes \mu(c', \varphi) = \max(\mu(c, \varphi) + \mu(c', \varphi) - 1, 0)$.

Given one of the above t-norms and a member value v and a fuzzy interval φ, we denote by $f\_neighbors(v^-, \varphi)$ and $f\_neighbors(v^+, \varphi)$ the following:

$$f\_neighbors(v^-, \varphi) = \frac{\Sigma^f_{c \in \mathcal{T}(v)} \mu(c, \varphi) \otimes \mu(c^-, \varphi)}{fCount(\mathcal{T}(v), \varphi)}, \text{ and}$$

$$f\_neighbors(v^+, \varphi) = \frac{\Sigma^f_{c \in \mathcal{T}(v)} \mu(c, \varphi) \otimes \mu(c^+, \varphi)}{fCount(\mathcal{T}(v), \varphi)}$$

where c^- and c^+ are the neighbors of c in $\mathcal{T}(v^-)$ and $\mathcal{T}(v^+)$, respectively.

Therefore, it should be clear that in the case of intervals (respectively fuzzy intervals) of measure values, in Algorithm 3, *supp* and *neighbors* should respectively be replaced by *i_supp* and *i_neighbors* (respectively by *f_supp* and *f_neighbors*).

4.3 Completeness Properties

In this section, we study the completeness of our approach in the case of single measure values. In particular, we show that if we consider a cube that can be partitioned into non-overlapping blocks containing the same measure value, then Algorithm 2.1 computes these blocks.

First, we show that, that for *limit thresholds,* our approach is complete for *any* cube C, in the sense that Algorithm 2.1 outputs blocks that represent *exactly* the content of C. In what follows, we call *limit thresholds:*

- a support threshold σ such that $0 < \sigma \leq \frac{1}{|C|}$,
- a confidence threshold γ such that $\gamma = 1$,
- a neighbor threshold ν such that $\nu = 1$.

Before giving the corresponding theorem, we note that considering limit thresholds implies the following:

- A block b is frequent for m if and only if b contains at least one m-cell.
- A block b such that $conf(b, m) \geq \gamma$ contains only m-cells.
- If in Algorithm 3, $\mathcal{L}_1(m)$ is computed according to $\nu = 1$, then for every m-cell c in a block b returned by Algorithm 2.1, all neighbors of c that belong to b are also m-cells.

Now, the completeness of our approach can be stated as follows, in the case of limit thresholds.

Theorem 1 *Let C be a k-dimensional cube. Then for limit thresholds, Algorithm 2.1 outputs a set of blocks $\mathcal{B}$ such that for every cell $c = \langle v_1, \ldots, v_k, m\rangle$ in C, there exists* one and only one *block b in $\mathcal{B}$ associated with m that contains c.*

PRODUCT

P1	5	5	5	5	6	6	
P2	5	8	5	5	6	6	
P3	5	8	8	5	6	6	
P4	8	8	8	8	6	6	
	C1	C2	C3	C4	C5	C6	**CITY**

Figure 5: The Cube for Example 4

PROOF: Let us first consider a cell $c = \langle v_1, \ldots, v_k, m\rangle$ in C. Since we assume that $0 < \sigma \leq \frac{1}{|C|}$, every slice $\mathcal{T}(v_i)$ is frequent for m. Therefore, according to Algorithm 3, each v_i is in an interval δ_i of $\mathcal{L}_1(m)$. Let us consider the block $b = \delta_1 \times \ldots \times \delta_k$ that, clearly, contains c.

Since we assume limit thresholds, all neighbors of c in b are m-cells, and thus, so are *all* cells of b. As a consequence, $conf(b, m) = 1$ and thus b is output by Algorithm 2.1, which shows that there exists at least one block b in $\mathcal{B}$ associated with m that contains c.

Assuming that two such blocks b and b' can exist implies that b and b' overlap and that they both contain only m-cells. However, this situation cannot happen because for any given measure value m, Algorithm 3 computes non-overlapping intervals. So, the proof is complete. □

We note that, although Theorem 1 shows an important theoretical feature of our approach, its impact in practice is of little relevance. Indeed, as shown in the following example, in the worst case, Algorithm 2.1 outputs blocks that are reduced to one single cell. However, the following example also shows that, with realistic threshold values, our approach can compute relevant blocks, even if in C, the measure values are not displayed in the form of blocks.

Example 4 *Let us consider the 2-dimensional cube C of Figure 4.3 and limit thresholds, for instance $\sigma = 0$ and $\gamma = \nu = 98\%$. In this case, for measure values 5 and 8, Algorithm 2.1 computes blocks that are reduced to one single cell, whereas for the measure value 6, Algorithm 2.1 computes the block $[C5, C6] \times ALL_{PRODUCT}$.*

To see this, let us consider the computation of $\mathcal{L}_1(8)$ for dimension CITY. First, for $\sigma = 0$, every slice $\mathcal{T}(Ci)$, $i = 1, \ldots, 6$, is frequent for 8. Moreover, for all slices, neighbors$(Ci^-, 8)$ or neighbors$(Ci^+, 8)$ are less than 1. Therefore, Algorithm 3 computes the intervals $[Ci, Ci]$, for

$i = 1, \ldots, 6$. *For dimension* $PRODUCT$*, a similar computation is produced and we obtain:*

$$\mathcal{L}_1(8) = \{[Ci, Ci] \mid i = 1, \ldots, 6\} \cup \{[Pj, Pj] \mid j = 1, \ldots, 4\}.$$

Then, when combining these intervals, Algorithm 2.1 outputs each 8*-cell of* C *as a block.*

It can be seen that a similar computation is done for the measure value 5*, whereas, for measure value* 6 *the block* $[C5, C6] \times ALL_{PRODUCT}$ *is returned by Algorithm 2.1.*

Now, we would like to emphasize that if we consider non limit thresholds, our approach computes relevant blocks, even in the case of this example. Indeed, let us consider as in Example 2, $\sigma = 1/12$*,* $\gamma = 80\%$ *and* $\nu = 60\%$*. Then, Algorithm 3 returns the following:*

$\mathcal{L}_1(5) = \{[C1, C1],\ [C3, C4],\ [P1, P3]\}$,
$\mathcal{L}_1(6) = \{[C1, C1],\ [P1, P4]\}$,
$\mathcal{L}_1(8) = \{[C2, C3],\ [P3, P4]\}$.

Applying Algorithm 2.1, we obtain the following blocks:

- *For measure value* 5*:* $[C1, C1] \times [P1, P3]$ *and* $[C3, C4] \times [P1, P3]$.
- *For measure value* 6*:* $[C5, C6] \times [P1, P4]$.
- *For measure value* 8*:* $[C2, C3] \times [P3, P4]$.

We note that, in this case, two blocks overlap and that only two cells (namely $\langle C2, P1, 5\rangle$ *and* $\langle C4, P4, 8\rangle$*) do not belong to any of these blocks.*

Now, the following proposition shows that, when in C, measure values are displayed in the form of blocks, then Algorithm 2.1 actually computes these blocks. To this end, we use the following terminology: a block b with all cells containing the same measure value m is called an m-block. Moreover, we introduce the notion of *block partition* as follows.

Definition 12 - Block Partition. *Let* C *be a* k*-dimensional cube and* $\mathcal{B} = \{b_1, \ldots, b_n\}$ *a set of blocks such that, for every* $i = 1, \ldots, n$*, all cells of* b_i *contain the same measure value* m_i*.* $\mathcal{B}$ *is called a* block partition *of* C *if:*

- *for all distinct* i *and* i' *in* $\{1, \ldots, n\}$*,* $b_i \cap b_{i'} = \emptyset$*,*
- $b_1 \cup \ldots \cup b_n$ *is equal to the set of all cells of* C*.*

The block partition $\mathcal{B} = \{b_1, \ldots, b_n\}$ *of* C *is said to be* maximal *if for any measure value* m*, there does not exist an* m*-block in* C *that contains an* m*-block* b_i *in* $\mathcal{B}$*.*

It is easy to see that, for the cube C of Figure 3.3, the set

$$\mathcal{B} = \{ \begin{array}{l} [C1, C3] \times [P3, P4],\ [C4, C6] \times [P1, P2], \\ [C1, C3] \times [P1, P2],\ [C4, C6] \times [P3, P4] \quad \} \end{array}$$

is a maximal block partition of C in which $[C1, C3] \times [P3, P4]$ and $[C4, C6] \times [P1, P2]$ are two 8-blocks, $[C1, C3] \times [P1, P2]$ is a 5-block and $[C4, C6] \times [P3, P4]$ is a 6-block. Moreover, it can also be seen that, considering limit thresholds (*i.e.,* $\sigma = 1/24$, $\gamma = \nu = 1$), Algorithm 2.1 computes exactly these blocks.

The following proposition generalizes this remark, and thus shows that our method is complete when the cube can be maximally partitioned into m-blocks.

Proposition 3 *Let* C *be a* k*-dimensional cube and let* $\mathcal{B} = \{b_1, \ldots, b_n\}$ *be a maximal block partition of* C*. Assume that, for every measure value* m*, it is the case that for all* m*-blocks* $b_i = \delta_{i,1} \times \ldots \times \delta_{i,k}$ *and* $b_j = \delta_{j,1} \times \ldots \times \delta_{j,k}$ *in* $\mathcal{B}$*,* $\delta_{i,p} \cap \delta_{j,p} = \emptyset$ *for every* $p = 1, \ldots, k$*. Then, for limit thresholds, Algorithm 2.1 returns* $\mathcal{B}$*.*

Proof: We first note that Theorem 1 shows that Algorithm 2.1 computes sub-blocks of blocks in $\mathcal{B}$. So, we have to show that for every m-block $b_i = \delta_{i,1} \times \ldots \times \delta_{i,k}$ in $\mathcal{B}$, Algorithm 3 returns exactly each interval $\delta_{i,p}$ for every $p = 1, \ldots, k$.

Given one of these intervals for dimension p, say $\delta_{i,p} = [\alpha, \beta]$, let us assume that Algorithm 3 returns an interval $\delta'_{i,p} = [\alpha', \beta']$ such that $\delta'_{i,p} \subset \delta_{i,p}$. Hence, at least one of the following two inequalities hold: $rep(\alpha) < rep(\alpha')$ or $rep(\beta') < rep(\beta)$. Assuming that $rep(\alpha) < rep(\alpha')$, let us consider the slice $\mathcal{T}(\alpha')$. According to our hypothesis on the intervals, b_i is the only m-block in $\mathcal{B}$ that intersects $\mathcal{T}(\alpha')$. As a consequence, $\mathcal{T}(\alpha')$ contains at least one m-cell, and thus is frequent for m.

Moreover, since b_i is a block and since $rep(\alpha) < rep(\alpha')$, for every m-cell $c = \langle v_1, \ldots, v_{p-1}, \alpha', v_{p+1}, v_k, m \rangle$ in $\mathcal{T}(\alpha')$, the cell $c^- = \langle v_1, \ldots, v_{p-1}, v, v_{p+1}, v_k, m' \rangle$ where $v = rep_p^{-1}(rep_p(\alpha) - 1)$ is such that $m = m'$ (*i.e.,* c^- is an m-cell). Therefore, in Algorithm 3, the value of n^- for the slice $\mathcal{T}(\alpha')$ is 1, in which case no new interval is considered. Thus, we have $\alpha = \alpha'$. As it can be shown in the same way that $\beta = \beta'$, the proof is complete. □

PRODUCT

P1	5	5	5	8	5	5	
P2	5	5	5	8	5	5	
P3	8	8	8	8	8	8	
P4	5	5	5	5	5	8	
	C1	C2	C3	C4	C5	C6	**CITY**

Figure 6: The Cube for Example 5

The following example shows that, if in the maximal partition $\mathcal{B}$ of C two m-blocks $b = \delta_1 \times \ldots \times \delta_k$ and $b' = \delta'_1 \times \ldots \times \delta'_k$ are such that $\delta_i \cap \delta'_i \neq \emptyset$ for some i, then Algorithm 2.1 does not compute $\mathcal{B}$.

Example 5 *Consider the cube C as shown in Figure 4.3 in which blocks are to be computed according to the limit thresholds. In this case, a maximal partition $\mathcal{B}$ of C is*

$$\begin{aligned}\mathcal{B} = \{ \quad & [C1, C3] \times [P1, P2],\ [C4, C4] \times [P1, P2], \\ & [C5, C6] \times [P1, P2],\ [C1, C6] \times [P3, P3],\ [C1, C6] \times [P4, P4] \quad \}\end{aligned}$$

It is clear that $\mathcal{B}$ does not satisfy the hypothesis of Proposition 3, due, for instance, to the two 5-blocks $[C1, C3] \times [P1, P2]$ and $[C1, C6] \times [P4, P4]$. On the other hand, running Algorithm 2.1 on the cube C of Figure 4.3 does not produce $\mathcal{B}$, since the 5-block $[C1, C3] \times [P4, P4]$ is split into $[C4, C4] \times [P4, P4]$ and $[C5, C6] \times [P4, P4]$ by the algorithm. In fact, Theorem 1 shows that in this case, Algorithm 2.1 outputs a non maximal partition of C that refines $\mathcal{B}$ in the following sense: for every block b computed by Algorithm 2.1, there exists a block b' in $\mathcal{B}$ such that $b \subseteq b'$.

It should be noticed that, in the case of intervals, Theorem 1 and Proposition 3 still hold since pairwise disjoint intervals can be thought of as single values. However, in the case of fuzzy intervals, we conjecture that our completeness results do not hold, because in this case, a member value v may belong to more than one interval. On the other hand, based on Proposition 2, it is conjectured that blocks computed by our method still cover the whole cube C, *i.e.,* it can be shown that:

- for Theorem 1, each cell belongs to *at least* one block in $\mathcal{B}$, and
- for Proposition 3, Algorithm 2.1 outputs *super blocks* of blocks in $\mathcal{B}$.

5 Experiments

In this section, we report on experiments in terms of runtime, number of blocks, and rate of overlapping blocks. Experiments have been performed on synthetic multidimensional data randomly generated. Depending on the experiments, the cubes contain up to 10^7 cells, the number of dimensions ranges from 2 to 9, the number of members per dimension ranges from 2 to 10, and the number of cell values ranges from 5 to 1000.

The first experiments report on the impact of taking into account single values, or intervals, or fuzzy intervals.

Figure 7 shows the number of blocks output by the three methods (single values, intervals and fuzzy intervals) according to the number of dimensions, and figure 8 shows the number of blocks output by the three methods (single values, intervals and fuzzy intervals) according to the number of members per dimension.

It should be noted that we obtain more blocks based on intervals than blocks based on single values. This is due to the fact that taking intervals into account increases the chance for a value to match a block value. However, the number of blocks based on fuzzy intervals is lower than the number of blocks based on the other two methods. This is due to the fact that fuzzy blocks can merge several blocks (which would have overlapped, as shown below).

Figures 9 and 10 show the runtime of the three methods (single values, intervals, fuzzy intervals) according to the size of the cube (number of cells). It can be seen that taking intervals and fuzzy intervals into account leads to slightly higher runtimes, if compared with the case of single measure values. However, all runtimes are still comparable and behave the same way.

Figure 11 shows the rate of overlapping blocks depending on the method (single values, intervals of fuzzy intervals) according to the number of dimensions of the cube. This figure suggests that, in the case of this dataset, using crisp methods leads to the fact that many blocks overlap (100% in the case of this experiment), while taking fuzziness into account reduces the rate of overlapping. This fact could be put in relation with the imprecision/uncertainty trade off, *i.e.,* the more certain, the less precise and conversely.

The following experiments show the impact of taking neighbors into account. Figure 12 shows the behaviour of the runtime according to the number of cell values. It can be seen that taking neighbors into account has no significant effect on the runtime.

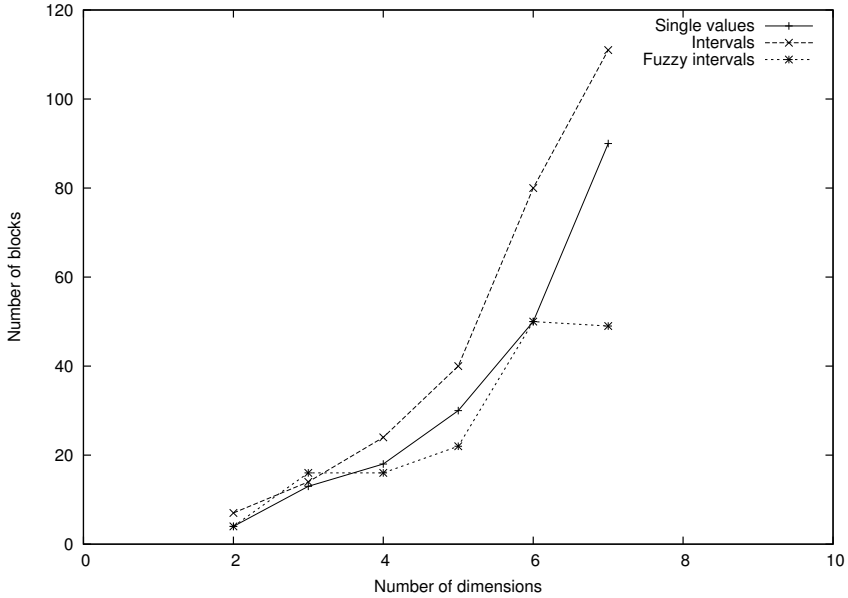

Figure 7: Number of discovered blocks w.r.t. the number of dimensions

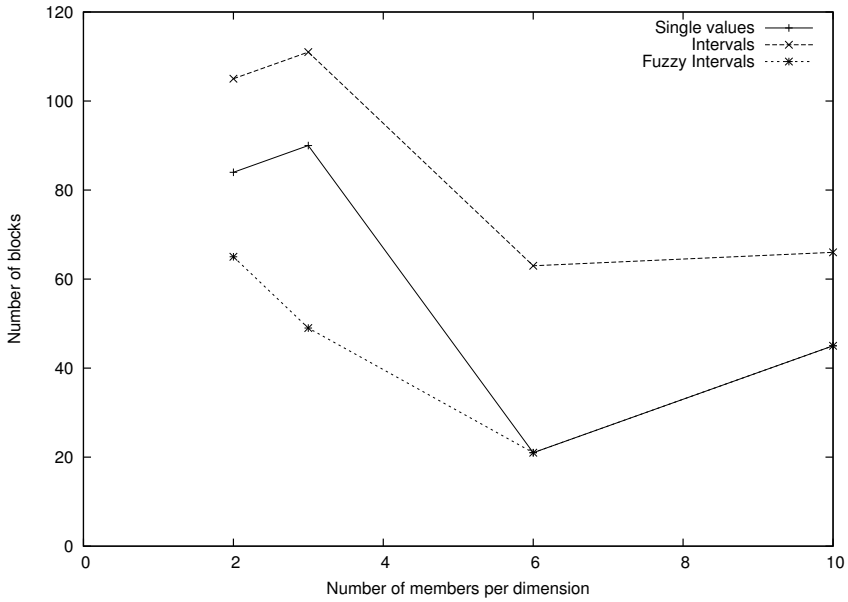

Figure 8: Number of discovered blocks w.r.t. the number of members

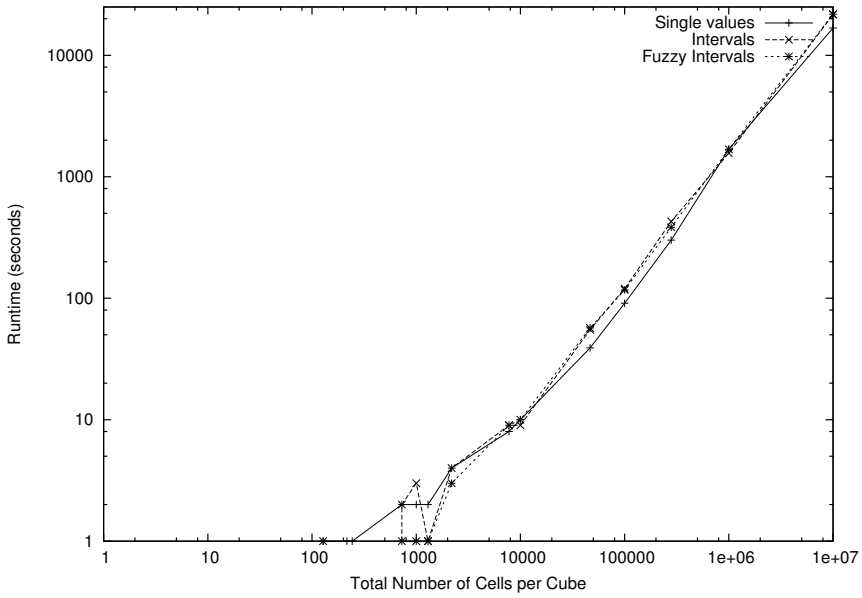

Figure 9: Runtime w.r.t. the size of the cube

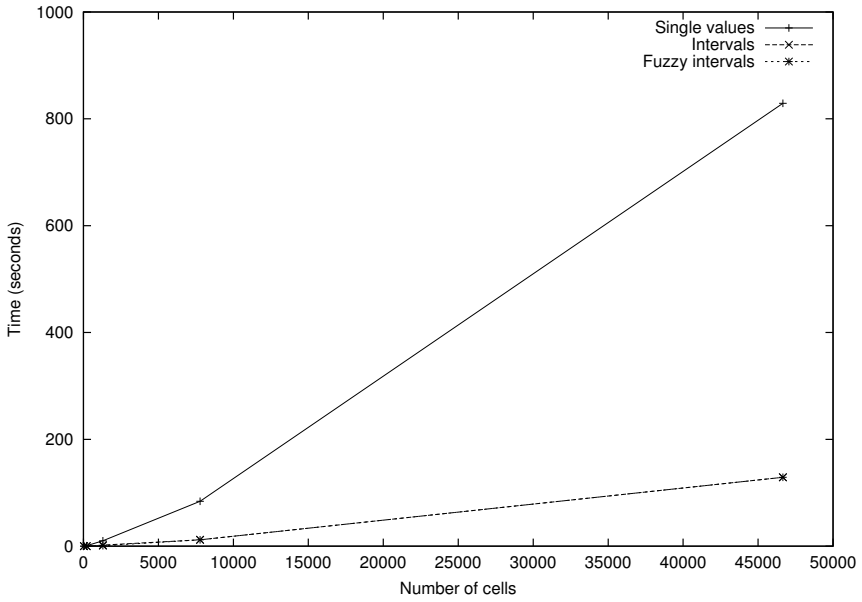

Figure 10: Runtime w.r.t. the size of the cube

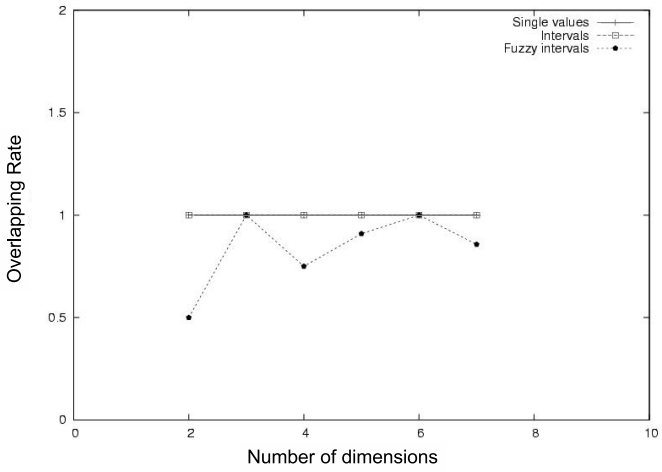

Figure 11: Rate of overlapping blocks w.r.t. the number of dimensions

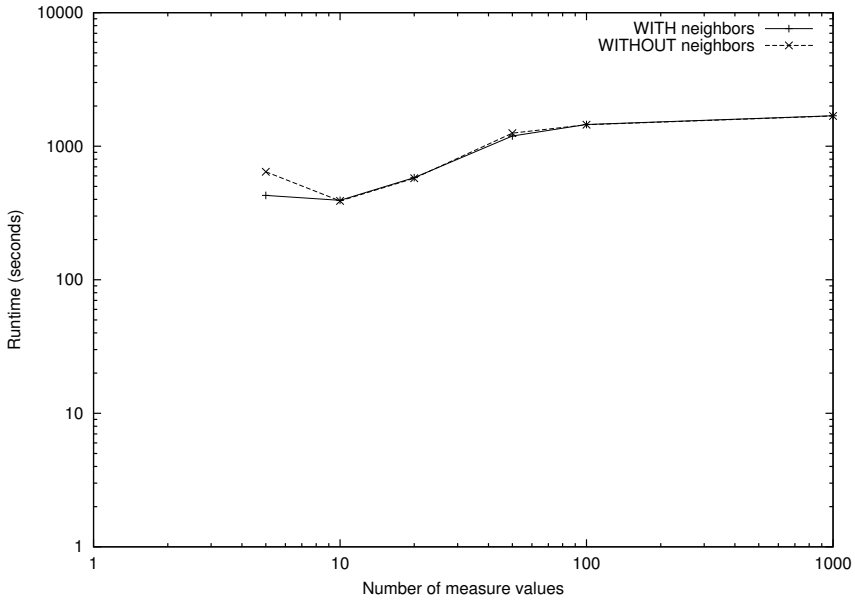

Figure 12: Runtime w.r.t. the number of measure values with and without neighbors (5 dimensions, 9 members per dimension)

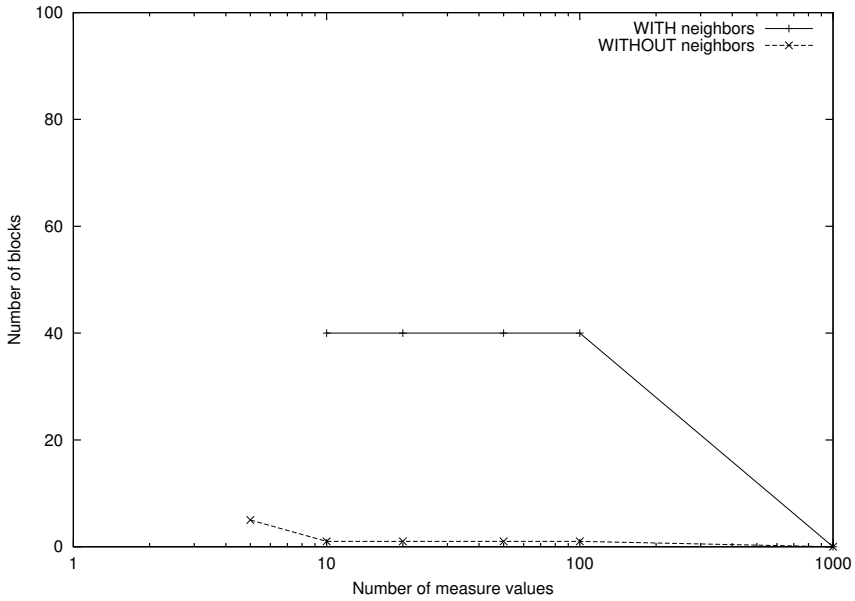

Figure 13: Number of blocks w.r.t. the number of measure values with and without neighbors (5 dimensions, 8 members per dimension, $\nu = 50\%$)

Figure 13 shows the behaviour of the number of blocks according to the number of measure values. It can be seen that taking neighbors into account leads to the discovery of more blocks.

We have also applied our method on the titanic database [8]. In this case, the database is organized according to four dimensions:

1. Dimension called $PASSENGER\_CLASS$ and defined on $dom_{PASSENGER\_CLASS} = \{1st, 2nd, 3rd, crew\}$.
2. Dimension called AGE and defined on $dom_{AGE} = \{adult, child\}$.
3. Dimension called SEX and defined on $dom_{SEX} = \{male, female\}$.
4. Dimension called $SURVIVED$ and defined on $dom_{SURVIVED} = \{yes, no\}$.

Moreover, we have considered a representation of the cube defined from the usual order as implicitly stated in the titanic file [8], that is:

1. $rep_1(1st) < rep_1(2nd) < rep_1(3rd) < rep_1(crew)$,
2. $rep_2(adult) < rep_2(child)$,

3. $rep_3(male) < rep_3(female)$,

4. $rep_4(yes) < rep_4(no)$.

The cube consists of 32 cells the content of which being the number of passengers concerned by the combination of one value per dimension. These numbers ranging from 0 to 670, we have partitioned the interval $[0, 670]$ into the following four intervals: $[0, 0]$, $]0, 20]$, $]20, 192]$, $]192, 670]$.

Considering the actual content of the cube, the first bin corresponds to no passenger, the second bin corresponds to numbers of passengers ranging from 4 to 20 (since there are no values between 0 and 4), the third bin corresponds to numbers of passengers ranging from 35 to 192 (since there are no values between 20 and 35), and the last bin corresponds to numbers of passengers ranging from 387 to 670 (since there are no values between 192 and 387).

We first note that, with a minimum confidence threshold of 50% and a minimum support threshold of $\frac{2}{32}$, taking single measure values into account has lead to the discovery of no block. On the other hand, with the same thresholds, when considering the intervals defined above, the following four blocks have been discovered:

1. $b_1 = ALL_{PASSENGER\_CLASS} \times [child, child] \times ALL_{SEX} \times [no, no]$, a $]192, 670]$-block meaning that the number of those passengers who were male children and who did not survive is among the highest.

2. $b_2 = [1st, 3rd] \times [adult, adult] \times ALL_{SEX} \times ALL_{SURVIVED}$, a $]20, 192]$-block meaning that the number of those adult passengers who were not crewmembers, whatever their sex and whatever their fate (survived or not) was between 35 and 192.

3. $b_3 = [1st, 2nd] \times [child, child] \times ALL_{SEX} \times ALL_{SURVIVED}$, a $]0, 20]$-block meaning that the number of those children passengers belonging to class 1st or 2nd, whatever their sex and their fate (survived or not), was very low but not null.

4. $b_4 = [crew, crew] \times [child, child] \times ALL_{SEX} \times ALL_{SURVIVED}$, a $[0, 0]$-block meaning that there were no children hired as crew members.

6 Related Work

The work on the building of blocks of similar values in a given data cube as presented in this paper can be related to the work on data clustering of high

dimensional data, which is an important area in data mining. For a large multidimensional database where the data space is usually not uniformly occupied, data clustering identifies the sparse and dense areas and thus discovers the overall distribution patterns (or summary) of the dataset. Some examples of work on clustering of high dimensional data include CLARANS [17], BIRCH [22], CLIQUE [19] and CURE [11].

Several subspace clustering methods are introduced to detect clusters residing in different subspaces (*i.e.,* subsets of the original dimensions). In this case, no new dimension is generated. Each resultant cluster is associated with a specific subspace. Some examples of these methods are CLIQUE [19] and ORCLUS [1].

CLIQUE (CLustering In QUEst) adopts a density-based approach to clustering in which a cluster is defined as a region that has higher density of points than its surrounding area. To approximate the density of data points, the data space is partitioned into a finite set of cells. Note that a block in our work is almost similar to the concept *unit* in [19] which is obtained by partitioning every dimension into intervals of equal length. Thus a unit in the subspace is the intersection of an interval from each of the k dimensions of a k-dimensional cube. However, the construction of the blocks in [19] is not determined by the same measure value, but rather by arbitrary chosen partitions of the member values.

Constructing subspaces using various methods can be viewed as related research but the aim is normally directed to tackling the issue of high dimensionality for clustering problems.

Research work on (fuzzy) image segmentation may appear as related works [18]. Although the goals are the same, it is not possible to apply such methods due to problems of scalability and because also of the multidimensional nature of data. For example, clustering-based color image segmentation [20] is normally limited to a 2-dimensional environment with the possibility of an extension to 3 dimensions.

Segmentation methods (*e.g.* clustering) have been proposed in the multidimensional context [19], [10]. In [12], the authors study the generation of fuzzy partitions over numerical dimensions. However, these propositions are not related to our measure-based approach, and thus these propositions are different from our work where the measure value is the central criterion.

On the other hand, the feature selection methods are used to select a subset of dimensions for supervised classification problem [16]. The idea is to produce an *optimal* pool of *good* dimension subsets for searching clusters. Therefore, in this approach, clusters are built up according to criteria related

to dimensions whereas in our approach, blocks are built up according to similarrity criteria on the measure values.

In [15] the authors aim at compressing data cubes. However there is no consideration on cube representations and homogeneous blocks generation.

The work presented in [3] proposes a method to divide cubes into regions and to represent those regions. However, the authors aim at representing the whole cube. They use statistical methods to construct an approximation of the cube, while we aim at discovering relevant areas, which may not cover the whole cube.

In [21], the authors propose the concept of *condensed* data cube. However, the authors aim at considering the cube without loss of information, while we aim at displaying relevant information to the user, which may be a partial representation of data.

7 Conclusion

In this paper, we have presented an efficient method for summarizing and visualizing multidimensional data. In our approach, blocks of homogeneous data are built to summarize the content of a given data cube, based on user specified thresholds. We have used a levelwise approach for the computation of the blocks and we have shown that our approach is tractable, in particular when the set of measure values is partioned into (fuzzy) intervals. Although efficiency results in a non complete method, completeness issues have been considered, and the experimental results obtained on synthetic data sets show that relevant blocks can be obtained efficiently.

In our future work, we plan to run further tests on real data to better assess the effectiveness and the accuracy of our approach. Moreover, we are also investigating the following research issues:

- How to combine the work in [6] and the work presented in this paper, in order to find a representation of the data cube for which large blocks can be computed?
- How standard OLAP operators such as roll-up or drill-down impact the blocks? More precisely, having built the blocks for a given level of details, can we optimize the construction of the blocks on the same data cube on which a roll-up or drill-down operation has been applied?
- The visualization of the blocks computed by our approach is also an issue we want to investigate further, based on our preliminary work reported in [5].

References

[1] C.C. Aggrawal and P.S. Yu. Finding generalized clusters in high dimensional spaces. In *Proc. of the Int. Conf. on Management of Data (SIGMOD'00)*, pages 70–81, 2000.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of ACM SIGMOD*, pages 207–216, 1993.

[3] D. Barbara and M. Sullivan. Quasi-cubes: Exploiting approximation in multidimensional data sets. *SIGMOD Rec.*, 26:12–17, 1997.

[4] Y.W. Choong, D. Laurent, and A. Laurent. Summarizing multidimensional databases using fuzzy rules. In *Int. Conf. IPMU'04*, 2004.

[5] Y.W. Choong, D. Laurent, and A. Laurent. Pixelizing data cubes: a block-based approach. In *Visual Information Expert Workshop (VIEW'06)*, Lecture Notes in Computer Science, 2006, to appear.

[6] Y.W. Choong, D. Laurent, and P. Marcel. Computing appropriate representation for multidimensional data. *DKE Int. Journal*, 45:181–203, 2003.

[7] E.F. Codd, S.B. Codd, and C.T. Salley. Providing olap to user-analysts: An it mandate. In *Technical Report*. Arbor Software Corporation, 1993.

[8] C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. UCI repository of machine learning databases, 1998.

[9] D. Dubois, E. Hülermeier, and H. Prade. A note on quality measures for fuzzy association rules. In *Int. Fuzzy Systems Association World Congress on Fuzzy Sets and Systems*, 2003.

[10] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 323–333, 1998.

[11] S. Guha, R. Rastagi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In *Proc. of the ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'98)*, pages 73–84, 1998.

[12] A. Gyenesei. A fuzzy approach for mining quantitative association rules. Technical Report 336, Turku Center for Computer Science (TUCS), 2000.

[13] W.H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, 1992.

[14] A. Kaufmann. *Introduction to the theory of fuzzy subsets*. Academic Press, 1973.

[15] L. Lakshmanan, J. Pei, and J. Han. Quotient cube: How to summarize the semantics of a data cube. In *Proc. of VLDB*, pages 778–789, 2002.

[16] Hiroshi Motoda and Liu Huan Liu. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, 1998.

[17] R.T. Ng and J. Han. Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

[18] S. Philipp-Foliguet, M. Bernardes Vieira, and M. Sanfourche. Fuzzy segmentation of color images and indexing of fuzzy regions. In *1st European Conference on Color in Graphics, Image and Vision (CGIV'02)*, pages 507–512, 2002.

[19] D. Gunopulos R. Agrawal, J. Gehrke and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. of the Int. Conf. on Management of Data (SIGMOD'98)*, pages 94–105, 1998.

[20] R.H. Turi. Clustering-based colour image segmentation, 2001.

[21] W. Wang, H. Lu, J. Feng, and J. X. Yu. Condensed cube: An effective approach to reducing data cube size. In *Proc. of the Int. Conf. on Data Engeneering (ICDE)*, pages 155–165, 2002.

[22] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proc. of the ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'06)*, pages 104–114, 1996.