The Generalized MDL Approach for Summarization

Laks V.S. Lakshmanan
Univ. of British Columbia
laks@cs.ubc.ca

Raymond T. Ng
Univ. of British Columbia
rng@cs.ubc.ca

Christine Xing Wang
Univ. of British Columbia
cwang@cs.ubc.ca

Xiaodong Zhou
Univ. of British Columbia
xdzhou@cs.ubc.ca

Theodore J. Johnson
AT&T Labs–Research
johnsont@research.att.com

Abstract

There are many applications in OLAP and data analysis where we identify regions of interest. For example, in OLAP, an analysis query involving aggregate sales performance of various products in different locations and seasons could help identify interesting cells, such as cells of a data cube having an aggregate sales higher than a threshold. While a normal answer to such a query merely returns all interesting cells, it may be far more informative to the user if the system returns summaries or descriptions of regions formed from the identified cells. The Minimum Description Length (MDL) principle is a well-known strategy for finding such region descriptions.

In this paper, we propose a generalization of the MDL principle, called GMDL, and show that GMDL leads to fewer regions than MDL, and hence more concise "answers" returned to the user. The key idea is that a region may contain "don't care" cells (up to a global maximum), if these "don't care" cells help to form bigger summary regions, leading to a more concise overall summary. We study the problem of generating minimal region descriptions under the GMDL principle for two different scenarios. In the first, all dimensions of the data space are spatial. In the second scenario, all dimensions are categorical and organized in hierarchies. We propose region finding algorithms for both scenarios and evaluate their run time and compression performance using detailed experimentation. Our results show the effectiveness of the GMDL principle and the proposed algorithms.

Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

1 Introduction

There are many applications in data analysis where we often wish to find out cells in a multi-dimensional data set where some property p of interest holds. For instance, property p may be minimum frequency, or aggregate sales above a certain threshold. A *cell* here refers to a combination of values from all dimensions. We give two examples and elaborate on one. The first example is a two-dimensional space over the dimensions `age` and `salary`, where a cell might correspond to an intersection of intervals over each dimension. Rather than enumerating all the interesting cells individually, the user may prefer a concise summary covering all interesting cells and no uninteresting ones. A summary amounts to a non-redundant covering of interesting cells using maximal axis-parallel hyper-rectangular regions [1]. The *Minimum Description Length* (MDL) principle is one of the most widely used approaches to provide such descriptions, where the description language is restricted to hyper-reactangular regions. (As will be clear shortly, hyper-rectangular regions are the most natural means of summarizing when hierarchies are involved.) Consequently, finding the MDL summary corresponds to finding a minimum rectangular covering.

As a second example, consider a table `sales(storeId, product, date, dollarAmt)`, registering the sales in dollar amount of products in different stores of a chain on various dates. Suppose the dimensions `storeId, product, date` are categorical in nature and have hierarchies defined on them. For example, stores may be grouped into cities, which may then be grouped into states, which in turn into countries and so forth. A cell in this example is any combination of values of the dimension attributes (`storeId, product, date`). A user may be interested in those cells satisfying a predicate p, e.g., those where the total aggregate dollar amount is $\geq$ \$500. Once again, a concise summary description of interesting cells is preferable to their enumeration since the former may be more meaningful and intuitive to the user. Note that a description does not correspond to any spatial regions in this example. Rather, it is a covering of interesting cells via "regions" in the categorical space, which correspond to (tuples of) nodes in the dimension hierarchies. For instance, a region might be (northeast, home electronics, 2002-Qtr-1).

In this paper, we study the general problem of summarizing cells of interest by means of a covering with regions. While the MDL principle has been successfully applied to many problems, we argue that the number of regions returned according to the MDL principle can still be large for the user's purpose. And for many situations, the determination of the property p may not be hard-and-fast. For instance, in the salary and age example above, the user may designate cells with t points or more as definitely "interesting", cells with $u(\ll t)$ points or less as definitely "undesirable", and the rest as "don't care". Similarly, for the data cube example, the interesting cells may be those with sales $\geq$ \$500; undesirable cells may be those with sales $\leq$ \$200; and "don't care" cells are those in between. In this paper, we study summarization based on a relaxation of the MDL principle – by allowing some number of "don't care" cells to be included in the summary. In return, the total number of summary regions is reduced. More specifically, the user can mark cells as blue ("interesting"), red ("undesirable"), and white ("don't care"). The generalized MDL summarization algorithm should then generate region coverings that cover all blue cells, no red cells, and possibly some white cells (up to a user-specified maximum "budget"). Specifically, we make the following contributions.

- We introduce the *Generalized* MDL (GMDL) region finding problem. One instance of the problem concerns the spatial case, where attributes are numeric. The other instance of the problem concerns categorical attributes organized in hierarchies (Section 2.3).

- As in [1], the MDL region finding problem has been studied for the spatial case where all attributes are numeric. The problem is NP-hard even for two dimensions, and heuristic approaches have been studied. The same complexity applies to GMDL region finding in the spatial case. Thus, we develop several heuristic algorithms. One type of algorithms is based on bottom-up merging of regions; the other type is based on top-down splitting of regions. Our experimental results show that, relative to MDL, the GMDL approach can improve compression by 2 orders of magnitude. And one top-down algorithm is the best for low dimensional data, whereas the bottom-up algorithm performs the best for higher dimensional cases (Section 3 and 4).

- As for categorical data, a key analytic result of this paper is to show that MDL region finding on data with tree hierarchies is not NP-hard, and can be solved in time linear in the sizes of the hierarchies. We then develop an efficient algorithm for finding the optimal MDL summary. The algorithm guarantees that no node in any hierarchy is visited more than twice. To our knowledge, this is the first algorithm for MDL region finding for categorical data. This algorithm is then extended to handle GMDL region finding. Experimental results show that, relative to MDL, again the GMDL approach improves compression by at least an order of magnitude (Section 5).

1.1 Related Work

There has been considerable work on using MDL as a principle for generating summaries in such diverse areas as data compression [12], decision tree construction [13, 9], learning of patterns [8], and image segmentation [6], to name just a few. As all these studies apply the MDL principle for generating summaries, the concept of GMDL introduced in this paper may also be applicable if the applications can tolerate the inclusion of "don't care" cells in the summary – in exchange for reduced summary size.

With respect to region finding specifically, the subspace clustering algorithm developed by Agrawal et al. is highly relevant [1]. For each subspace of a given multi-dimensional space, their algorithm identifies cells with high enough density, connects those cells to form clusters, and summarizes each cluster with regions based on the MDL principle. One of our algorithms developed for the spatial case, called Algorithm BP, can be viewed as an extension to their algorithm. BP first identifies MDL regions, and then merges pairs of MDL regions to form GMDL regions as long as the white budget is not exhausted. Furthermore, apart from the BP algorithm, we develop top down algorithms, which have no parallel in [1]. Last but not least, we consider in this paper categorical data organized in hierarchies. To the best of our knowledge, this is the first paper to consider MDL and GMDL region finding for categorical data.

In numerous data warehousing and data cube scenarios, categorical dimensions with hierarchies are customary. In recent years, finding cells in data cubes satisfying certain user-defined property p has received a lot of attention, including the studies by Garcia-Molina et al. [4], Beyer and Ramakrishnan [3], and Ng et al. [10]. All these studies focus on the efficient computation of the cells satisfying p. However, enumerating all the qualified cells may not be the most intuitive to the user. The GMDL algorithms developed here can then be used as a "post-processing" step to provide a more concise summary.

2 Motivating Examples and Problem Statements

2.1 An Example for The Spatial Case

Suppose a company plans to conduct a promotion of a new product. Based on the buying patterns of its current customers, the company wishes to divide customers into the categories of top-notch, mediocre and hopeless. Let us say that the customer segmentation is performed over the dimensions `age` and `salary`, with a point (x, y) in the space representing a person with age x and salary y. Figure 1 shows a particular example, where cells are in 10K intervals in salary and 5-year intervals in age. Let a cell be defined as: (i) blue (i.e., top-notch), if its frequency is $\geq t$ (here, frequency may mean the number of existing customers buying a similar product); (ii) red (i.e., hopeless), if its frequency is $\leq t/2$; and (iii) white, otherwise. In Figure 1, blue cells are marked 'O', red cells marked 'X', and white cells unmarked.

Figure 1 shows an MDL-based covering with rectangles R_1, R_2, R_3 and R_4. Rectangle R_1 corresponds to the region $35 \leq age \leq 50$ and $60K \leq salary \leq 80K$, and so on. Note that the four rectangles cover all the blue cells, and no other cells.

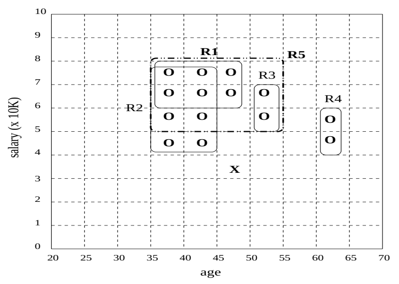


Figure 1: An Example for the Spatial Case

Now let us apply the GMDL principle with a "white budget" w, i.e., no more than a total of w white cells can be included. Suppose the white budget is $w = 2$. Then R_1 and R_3 can be merged into the bigger rectangle R_5, corresponding to the region $35 \leq \texttt{age} \leq 55$ and $50K \leq \texttt{salary} \leq 80K$. Thus, the rectangles R_2, R_4, R_5 form a GMDL-based covering, and the size of this covering is one less than that of the MDL-based covering. To push this further, with a white budget of $w = 10$, the size of the GMDL-based covering is reduced to one, merging $R_1, \ldots, R_4$ altogether.

Note that when $w = 0$, there is no difference between MDL and GMDL. In general, the bigger the value of w, the more *probable* it is for the number of summary regions under GMDL to be smaller. However, it is not necessary that GMDL will always do better (but guaranteed to do no worse), because the distribution of red cells is critical. In our example, if the cells (i) $\texttt{age} \in [45, 50)$ & $\texttt{salary} \in [50K, 60K)$ and (ii) $\texttt{age} \in [55, 60)$ & $\texttt{salary} \in [50K, 60K)$ were both red, the GMDL-based covering would be back to the original 4 rectangles, irrespective of the white budget.

2.2 An Example for The Hierarchical Case

Consider now sales of clothes in various locations. This might be recorded using a table such as `sales(itemId, city, dollarAmt)`, where `itemId` represents the id of the particular clothes item being sold. Here clothes and location are dimensions while dollar amount is the measure. And there are hierarchies on the dimensions, e.g., clothes classified into men's and women's which may be further sub-classified, as shown in Figure 2. For categorical data, a cell corresponds to a tuple of leaf level values in the various dimensions. For example, (`new york, ties`) is a cell, while (`new york, men's`) is not.

In the figure, again a blue cell is marked 'O', red cell marked 'X', and white cell unmarked. Suppose here the company manager wants to summarize all the "hot" items. Because there is no hard-and-fast rule for defining "hot" items, the manager may define a cell as blue if the sales this year is at least 2 times that of last year, a cell as red if the sales this year is below 50otherwise. Suppose there are n white cells in all. Typically, the white budget p will be set to be a fraction of n, that is deemed acceptable to the user for trading off the purity of the summary for a reduced size. Notice that the situation would be different if we were to re-color p best white cells to be blue and set the white budget to 0. The difference is that in the latter case, there would be no choice of white cells and all blue cells have to be covered. In contrast, with the current definition, any p of the n white cells can be chosen.

First let us consider covering blue cells based on the MDL principle. Figure 2 shows an MDL covering. Each region must admit a description as a pair of values (nodes) in the dimension hierarchies. For instance, the region R_9 at the bottom right has the description (`men's, northeast`). Note that regions R_7 and R_8 *cannot* be combined into one even though they are adjacent, since there is no hierarchical descriptor for such a combination.

Next let us turn to the GMDL principle and consider the red cell marked 'X'. Suppose we have a white budget $w = 2$. Regions R_7 and R_8 can be replaced by a new region R_{10} with description (`men's, midwest`). When the white budget w is raised to 7, we can obtain a better covering by replacing regions R_5, R_6, R_7, R_8 with a new region R_{11} with description (`clothes, midwest`). However, note that when w is increased beyond 7, the placement of the red cells prevents any more regions to be merged.

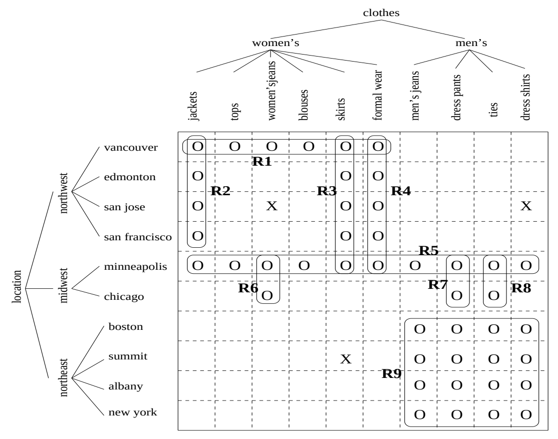


Figure 2: An Example for the Hierarchical Case

2.3 Problem Statements

2.3.1 The Spatial Case

Let there be k totally ordered, finite domains $D_1, \ldots, D_k$. Domain D_i is partitioned into n_i (non-overlapping) intervals (not necessarily of equal length). A cell is the intersection of one interval from each dimension. Thus, there are $n_1 * \ldots * n_k$ cells. Let $\mathcal{S}$ denote the set of all cells. $\mathcal{S}$ is partitioned into three subsets: $\mathcal{B}$, the subset of blue cells, $\mathcal{R}$, the subset of red cells, and $\mathcal{W} = \mathcal{S} - \mathcal{B} - \mathcal{R}$, the subset of white cells. A *region* is an axis-parallel hyper-rectangle of cells without any hole. A *GMDL covering* is a set of regions $\{R_1, \ldots, R_m\}$, where $(R_1 \cup \ldots \cup R_m) \supseteq \mathcal{B}$, and $(R_1 \cup \ldots \cup R_m) \cap \mathcal{R} = \emptyset$. A GMDL covering is *feasible* with respect to the white budget w, if $|(R_1 \cup \ldots \cup R_m) \cap \mathcal{W}| \leq w$. The GMDL region finding problem is to find a feasible GMDL covering of the least cardinality. For the example shown in Figure 1, the GMDL covering consisting of rectangles R_2, R_4 and R_5 is optimal with respect to $w = 2$.

Clearly, if the white budget w is equal to 0, or the subset of white cells is empty, a GMDL covering is also an MDL covering (defined in an obvious way). Given that finding the MDL covering of the least cardinality is NP-hard, even in the 2-dimensional case [11], it follows that finding a feasible GMDL covering of the least cardinality is also NP-hard. Thus, in the next section, we develop heuristic algorithms to find feasible GMDL coverings, with no guarantee of optimality. However, to measure the quality of the coverings found by our heurisitcs, we have the following definitions. We say that a region is *blue-maximal* if it cannot be expanded further to include another blue cell without either including a red cell, or exceeding the white budget. (For the MDL case, a blue-maximal region contains neither red nor white cell.) We say that a GMDL (MDL) covering is blue-maximal if every region in it is blue-maximal. Furthermore, we say that a GMDL (or MDL) covering is *non-redundant* provided that no region in it is contained in the union of the remaining regions in the covering.

2.3.2 The Hierarhical Case

Let there be k categorical dimensions with each dimension associated with a finite hierarchy. In this paper, we restrict our attention to tree hierarchies only. Let the k hierarchies be denoted as $T_1, \ldots, T_k$. A cell is a tuple $(c_1, \ldots, c_k)$, where c_i is a leaf node in the tree hierarchy T_i. A region is a tuple $(x_1, \ldots, x_k)$, where x_i is a leaf node or an internal node in hierarchy T_i. Region $(x_1, \ldots, x_k)$ is said to *cover* cell $(c_1, \ldots, c_k)$, if c_i is a (not necessarily proper) descendant of x_i, for all $1 \leq i \leq k$.

An *MDL covering* is a set of regions that cover all the blue cells, and that each region covers blue cells only. The MDL region finding problem for hierarchical dimensions is to find a MDL covering with the least cardinality. As will be shown later in Section 5, to solve the MDL region finding problem, it is sufficient to find an MDL covering that is non-redundant and blue-maximal (with the notions of non-redundancy and blue-maximality defined as in Section 2.3.1). Finally, the GMDL region finding problem for hierarchical dimensions is to find a feasible GMDL covering with the least cardinality (with all related concepts defined as in Section 2.3.1).

3 Algorithms for the Spatial Case

In this section, we present four heuristic algorithms for solving the GMDL region finding problem for numeric dimensions. The first algorithm is a bottom-up algorithm, whereas the other three algorithms are top-down. A bottom-up algorithm first constructs an initial feasible GMDL covering, and then merges regions in the covering to reduce the number of regions, until the white budget is used up. In contrast, a top-down algorithm first begins with a covering consisting of a *single* region covering all blue cells. If this covering is not a GMDL covering (i.e., some region containing a red cell) or is infeasible (i.e., including too many white cells beyond the white budget), then a region in the covering is split either to remove the red cell, or to reduce the total number of white cells covered.

3.1 Algorithm BP

Because the MDL region finding problem for numeric dimensions is a rather well studied problem, it is natural to "borrow" a good algorithm from there and to extend it to give an algorithm for the GMDL problem. Specifically, an MDL algorithm produces a feasible GMDL covering consuming no white cell. To take advantage of the allowed white budget, two regions from the GMDL covering are selected to be merged, as long as no red cell is included, and the white budget is not used up. A key issue here is how to select the two regions. Let $\mathcal{P}$ be the set of pairs of regions

Algorithm BP

```
1 Build an index $I_A$ for all the red cells.

2 Run the algorithm in [1] to produce an MDL covering. Build
  another index $I_B$ for all the regions in the covering. Initialize
  rem-budget to be the white budget and pair-count to 0.

3 While (rem-budget $> 0$) and (pair-count $<$ MAX-trial) {

    3.1 Pick a random pair of regions $(R_1, R_2)$ that are close to
        each other in the region index $I_B$ (e.g., in the same leaf
        page). Increment pair-count.
    3.2 If (rem-budget $< |R_1 \oplus R_2| - |R_1| - |R_2| + |R_1 \cap R_2|$),
        then go back to step (3.1).
    3.3 Use the red index $I_A$ to test if the merged region $R_1 \oplus R_2$
        contains any red cell. If so, go back to Step (3.1).
    3.4 Remove the regions $R_1, R_2$ from index $I_B$ and add the
        merged region $R_1 \oplus R_2$ to the index. Reset pair-count
        to 0, and reduce rem-budget appropriately. }

4 Return all the regions in $I_B$ as the GMDL covering.
```

Figure 3: Pseudo Code for Algorithm BP

(R_1, R_2) in the covering such that the merged region, denoted as $R_1 \oplus R_2$, does not contain any red cell. A best fit strategy is to find the pair of regions that minimizes the additional consumption of white cells, i.e., $|R_1 \oplus R_2| - |R_1| - |R_2| + |R_1 \cap R_2|$. A first fit strategy is to find the first pair of regions in $\mathcal{P}$, based on a certain enumeration ordering of the elements in $\mathcal{P}$. Experimental results indicate that while a best fit strategy, when compared with the first fit strategy, may produce GMDL coverings of slightly higher quality, the cost for doing so is orders of magnitude higher.

Algorithm BP (to stand for "Bottom-up Pairwise") adopts the first fit strategy. As an optimization, the algorithm uses an index to organize all the regions in the current covering. This helps to identify a pair of regions (R_1, R_2) that are "rather close" to each other, as they are clustered by the indexing structure. In this way, if the regions are merged, the additional consumption of the white budget $|R_1 \oplus R_2| - |R_1| - |R_2| + |R_1 \cap R_2|$ will not be too outrageous. If a best fit strategy were used, the entire index might need to be traversed to identify the best pair. In our implementation, we use lean-trees, which is a variant of X-trees developed by Kriegel et al for high dimensional indexing [2]. Algorithm BP builds another index structure containing all the red cells. This helps to facilitate the testing of whether the merged region $R_1 \oplus R_2$ contains any red cell. Finally, for the algorithm producing MDL regions, BP uses the heuristic algorithm developed by Agrawal et al [1]. Specifically, it first finds blue-maximal regions to cover all blue cells, and then removes redundant blue-maximal regions. Figure 3 shows the pseudo code of Algorithm BP.

A stopping condition of the while-loop in step (3) is (pair-count $<$ MAX-trial). This deals with the situation when the remaining budget is close to 0, but every pair of regions attempted thus far uses too many white cells. In this case, only a maximum number of attempts will be made.

3.2 Algorithm CAS-Interior

A bottom-up algorithm has two potential weaknesses. First, finding MDL regions and merging them to form a GMDL covering may appear to incur unnecessary overhead. Second, a bottom-up algorithm may not perform well if there are many MDL regions to start with. This is particularly problematic if the white budget is large. This motivates us to develop top-down algorithms. The aim is to build a feasible GMDL covering, without resorting to finding MDL regions.

A top-down algorithm starts with a region R containing all blue cells. A region in the current covering is chosen for splitting whenever it covers a red cell or too many white cells. This can be modeled as an R-tree node splitting problem [7]. This is the essence of Algorithm RTS (for "R-Tree Splitting"). Since R-tree splitting is a well-studied problem, we do not elaborate RTS any more except to note that in our implementation, we use Garcia et al.'s optimal polynomial time node splitting algorithm [5]. It is one of the best known splitting strategies for dynamic R-trees.

In analyzing Algorithm RTS, a potential weakness is that RTS may be too "color-blind". This is particularly true on its treatment, or lack thereof, of the red cells. RTS relies passively on the node splitting algorithm to produce split regions, which may contain a smaller number of red cells.

Given that all red cells in a covering must be removed to obtain a GMDL covering, the next algorithm, called CAS (to stand for "Color Aware Splitting"), actively seeks out red cells contained in a region. Furthermore, before a red cell is removed in a split, the red cell is grown to form a larger red region, which may contain other red cells and white cells, but no blue cell.

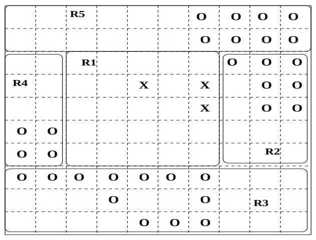


Figure 4: An Example of Color Aware Splitting

As an illustration, consider the region shown in Figure 4. As usual, blue cells are marked 'O', red cells marked 'X', and white cells unmarked. For simplicity, we refer to a cell by its coordinates. For instance, the three red cells in the region are (5,7), (7,6) and (7,7). Algorithm CAS takes a red cell and grows it into a larger region. For our example, the larger region becomes the whole rectangular area R_1, including all three red cells and many white cells. This serves to remove the red cells as quickly as possible, and to reduce the consumption of white cells as a side-benefit.

Algorithm CAS-Interior

```
1 Build indices $I_A, I_B$ for all the red and blue cells respectively.
  Construct a a single region $R$ containing all the the blue cells.
  Initialize the covering $\mathcal{C}$ to contain $R$ only. Initialize curr-
  consumption to be the number of white cells in $R$.
2 While (there exists $R \in \mathcal{C}$ containing a red cell) {
    2.1 Grow the red cell in $R$ to a larger region not containing
        any blue cell using the blue index $I_B$.
    2.2 Split $R$ into (at most) $2d$ regions, where $d$ is the dimen-
        sionality of the space, excluding the entire red region.
    2.3 Remove $R$ from $\mathcal{C}$, but add the split regions into $\mathcal{C}$. Up-
        date curr-consumption. }
3 While (curr-consumption > white budget) {
        Do splitting similar to step (2), this time growing based
        on white cells. }
4 Return all the regions in $\mathcal{C}$ as the GMDL covering.
```

Figure 5: Pseudo Code for Algorithm CAS-Interior

With the red region identified in the interior, the remaining cells are split into 4 non-overlapping regions, $R_2, \ldots, R_5$.

Note that non-overlapping regions represent a tradeoff between quality and efficiency. Region R_2 is not blue-maximal because it does not include the 6 adjacent blue cells in region R_5. This represents a reduction in quality. However, if we were to allow overlapping regions in a top-down algorithm, then calculating the total consumption of white cells in a covering of overlapping regions would require tremendous bookkeeping to avoid double counting of white cells. For example, if R_2 is augmented to become blue-maximal, then when the time comes to have R_2 split, the calculation of the total consumption of white cells in the split regions of R_2 will need to involve R_5 and possibly R_3, which overlap R_2. Furthermore, this bookkeeping would need to be updated for each split. Thus, for efficiency purposes, all the top-down algorithms presented here produce only non-overlapping split regions.

Figure 5 shows the pseudo code of Algorithm CAS-Interior. As usual, there is an index of all the red cells to facilitate the testing of whether a region is free of red cells. This time there is also an index for all the blue cells. The index is used when a red cell (step 2.1) or a white cell (step 3) is grown to be as large as possible, while remaining free of blue cells.

3.3 Algorithm CAS-Corner

So far, we have not yet explained why Algorithm CAS-Interior is qualified with the word "Interior". This is due to the fact that given a region R to be split, if a random red cell or a random white cell is picked to grow into a larger region for the split, chances are the cell is located in the interior of R, rather than along its perimeter. The consequence is that each split may produce $2d$ split regions, where d is the dimensionality of the space. A potential problem is that the number of regions in a covering multiplies rapidly, thus sacrificing the overall quality of the covering.

Algorithm CAS-Corner

```
The same as in CAS-Interior, except for step (4):
4 While (curr-consumption > white budget) {
    4.1 Identify the region $R \in \mathcal{C}$ containing the largest number
        of white cells.
    4.2 First try to locate a corner of $R$ occupied by a white cell.
        If all corners are occupied by blue cells, pick a random
        white cell.
    4.3 Perform steps (2.1), (2.2) and (2.3) to this white cell,
        except that if the white cell or the eventual white region
        is at a corner, (at most) $d$ split regions are produced in
        step (2.2). Reduce curr-consumption accordingly. }
```

Figure 6: Key Step for Algorithm CAS-Corner

Because every red cell has to be removed from a region R in a covering, if a red cell is located in the interior, there is nothing much we can do to avoid creating $2d$ split regions. However, the situation is very different for a white cell. Because there is a white budget allowing some number of white cells to be included, we may want to avoid picking white cells in the interior. As a variant of Algorithm CAS-Interior, Algorithm CAS-corner, shown in Figure 6, first examines the corners of R for white cells. The advantage of this strategy is that if a split occurs at a corner, only d split regions will be produced. If, unfortunately, all corners of R are occupied by blue cells, then the algorithm reverts to picking a random white cell, likely from the interior of the region.

3.4 Properties of the Algorithms

As pointed out before, finding a covering with the least cardinality is NP-hard. Thus, all the algorithms are heuristics. While in the next section we will conduct an empirical evaluation of the four algorithms, the statement below summarizes the correctness of the algorithms and compares the algorithms based on blue-maximality and non-redundancy.

Property 3.1 1. Algorithms RTS, CAS-Interior and CAS-Corner produce a non-redundant feasible GMDL covering.

2. Algorithm BP produces a feasible GMDL covering. It can be made to produce a non-redundant covering by adding a redundancy checking post-processing step.

It is straightforward to verify that all four algorithms are correct, i.e., producing a feasible GMDL covering. For the top-down algorithms RTS, CAS-Interior and CAS-Corner, splitting a region into smaller non-overlapping regions guarantees non-redundancy. As remarked before, however, the non-overlapping split regions may not be blue-maximal.

The situation for the bottom-up Algorithm BP is more complicated. First of all, merging two regions may create redundant regions in the covering. For efficiency reasons, Algorithm BP, as presented in Figure 3, does not check to eliminate redundant regions in the covering. (Checking a region for redundancy is expensive since it involves computing the union of all the remaining regions. Furthermore, removing redundant regions is not Church-Rosser.) Like the top-down algorithms, BP does not necessarily give a blue-maximal covering. However, the reason is different. Merging two blue-maximal regions A, B does not necessarily give a merged region $A \oplus B$ that is blue-maximal. This is because there may be another blue-maximal region C that is adjacent to and can merge with the merged region $A \oplus B$.

4 Experiments

4.1 Experimental Setup

We implemented all four algorithms in C++ and the MDL region finding algorithm developed in [1]. We used the code developed by Scott Leutenegger for optimal node splitting [5], and the lean tree implementation developed by Kriegel et al [2]. We implemented a data set generator to produce synthetic data sets. The generator takes as input various parameters, including the dimensionality of the numeric space, the number of intervals per dimension, the number of blue cells, blue cell density, etc. For most of the results reported below, the default configuration is a 3-dimensional space with 100 intervals per dimension (giving rise to a million cells). There are 50,000 blue cells, which are randomly picked from a clustered region of size equal to 50,000/bd, where bd is the blue density. For example, if the blue density is $bd = 75\%$, the clustered region is of size $50000/0.75 = 66{,}667$ cells. Within this region, 5% are randomly placed red cells. (In other words, while there may be many more red cells, they are irrelevant if they are not in a region with blue cells.) The white budget is expressed as a budget ratio relative to the number of blue cells, essentially measuring the purity of the summary regions.

To measure the effectiveness of the algorithms, we use the relative quantity called *compression gain*, which is defined as the ratio of the number of blue cells to the number of GMDL regions. To measure efficiency, we use the total runtime of an algorithm, including the creation of indices, if required. Because of the random nature of the data sets and some algorithms, each runtime figure reported in the graphs represents the average of 3 runs. The machine we used is a Sparc-10 workstation.

4.2 Runtime and Compression Gain

In the first set of experiments, we ran all four algorithms using data sets of the default configuration. We varied the number of blue cells, while keeping the white budget to be 75% of the number of blue cells. Figure 7(a) shows the compression gain values as the number of blue cells increases from 10,000 to 50,000. Figure 7(b) shows the corresponding execution times.

Algorithm RTS delivers the worst compression gain (around 50 times) and takes relatively long to execute. CAS-Interior is better than RTS in that it takes a lot less time to deliver similar compression gain. On the other hand, Algorithm BP produces superior compression gain (e.g., over 100 times); unfortunately, it takes even more time than RTS to achieve the gain. Clearly, CAS-Corner is the dominant algorithm, as it delivers the best compression gain (around 150 times) in 1-2 seconds of total execution time.

4.3 Varying the White Budget: GMDL vs MDL

The second set of experiments was identical to the first set, except that we varied the size of the white budget, while keeping the total number of blue cells to be 50,000. Figure 7(c) shows the compression gain as the white budget increases from 0 to slightly over 100% of the number of blue cells.

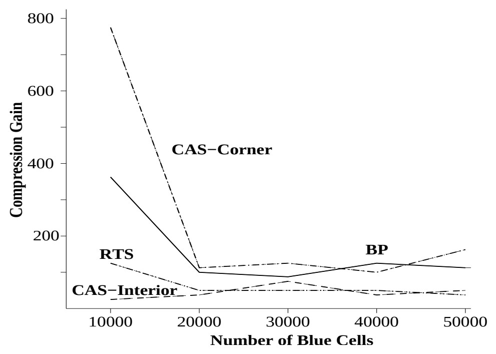
Notice that when the white budget is 0, this corresponds to finding MDL regions. The figure clearly shows the effectiveness of the GMDL principle. Consider CAS-Corner as an example. Even for a white budget of 40% (i.e., 20,000 cells), the compression gain is around 140 times, as compared with 2 times based on the MDL principle. This corresponds to a 70-fold reduction in the size of the covering. Amongst the proposed algorithms, CAS-Corner is the dominant one.

4.4 Changing the Dimensionality

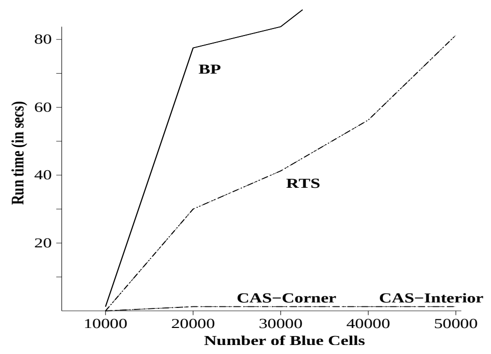
In this set of experiments, we varied the dimensionality of the space from 4 to 10. The total number of cells in the space is made directly proportional to the dimensionality. Specifically, there are 4 intervals per dimension (e.g., about 1 million cells in a 10-dimensional space). The percentages of blue and red cells remain constant. For lack of space, we suppress all those graphs and show instead the single graph in Figure 7(d). The graph shows the compression gain normalized with the execution time. In a sense, this measures compression gain per unit of effort. Given that Algorithms RTS and CAS-Interior are dominated by the others, the graphs in the figure only show BP and CAS-Corner.

Consistent with the graphs discussed earlier, CAS-Corner is the one to use when the dimensionality d is low (i.e., $d < 6$). Recall that in CAS-Corner each split creates d split regions. As d becomes larger and larger, the number of regions in the covering multiplies rapidly. This increases execution time, and lowers the compression gain significantly.

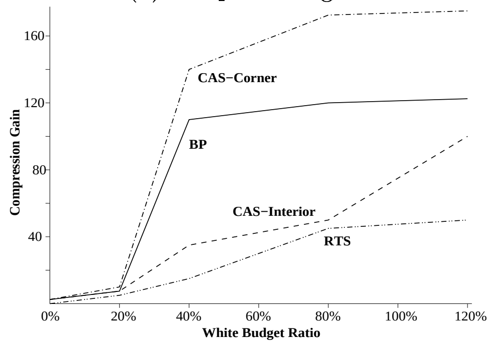
In contrast, the bottom-up Algorithm BP scales up better with respect to dimensionality. While the compression gain does not degrade as dimensionality in-



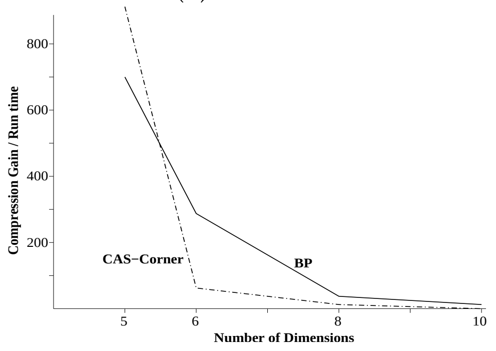
(a) compression gain



(b) run time



(c) changing the white budget



(d) changing the dimensionality

Figure 7: Comparing the Four Algorithms for the Spatial Case

creases, the execution time increases. However, this is due largely to the corresponding increase in the number of blue cells. Algorithm BP is the recommended choice for dimensionality $d \geq 6$.

5 Algorithms for the Hierarchical Case

So far we have studied the GMDL region finding problem for the spatial case. In this section we turn our attention to the hierarchical case – that is, categorical data organized in hierarchies. Since, to our knowledge, this is the first paper to study the MDL region finding problem for categorical data with hierarchies, we devote the first part of this section to the MDL problem. Then we consider the GMDL version at the end.

5.1 Uniqueness of the Optimal MDL Covering

A key result below is that, unlike in the spatial case, finding an optimal MDL covering for categorical data with tree hierarchies can be solved in polynomial time. But before delving into the details, it is important to first analyze carefully the differences between the spatial case and the hierarchical case. To this end, two key points should be noted regarding MDL (GMDL) coverings in general. First, an MDL (GMDL) covering of the least cardinality must be non-redundant. For if it was not, we could remove any redundant regions and obtain a smaller covering. Second, an MDL (GMDL) covering of the least cardinality need *not* be blue-maximal. It is straightforward to expand it into one, however. Figure 8(a) illustrates this situation for the spatial case. Here the lettered cells are blue and red cells marked 'X' as usual (which should be ignored for MDL coverings). Then the covering $\{\{a, b\}, \{c, d\}\}$ is a covering of the least cardinality. However, it is not blue-maximal; the corresponding blue-maximal covering of the least cardinality is $\{\{a, b\}, \{b, c, d\}\}$.

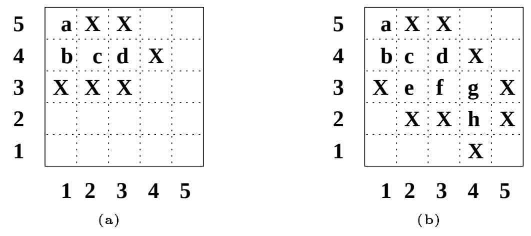


Figure 8: Blue-maximality vs. Non-redundancy for the Spatial Case

A harder issue is whether there is more than one blue-maximal non-redundant MDL covering, and if so, whether there are such coverings with different cardinalities. Figure 8(b) shows that the answers to both questions is in the affirmative for the spatial case. Again, the lettered cells are all the blue cells. There are two non-redundant blue-maximal coverings: (i) $\{\{a, b\}, \{b, c, d\}, \{e, f, g\}, \{g, h\}\}$; and (ii) $\{\{a, b\}, \{c, d, e, f\}, \{g, h\}\}$. The latter covering is strictly smaller in size and is the optimal one. Thus, for the spatial case, generating an arbitrary blue-maximal non-redundant MDL covering does not guarantee optimality. This is a manifestation of NP-hardness in the spatial case.

To return to the categorical case with tree hierarchies, we will show, however, that an optimal MDL covering can indeed be obtained in polynomial time. The first observation is the following.

Property 5.1 Let R be a region $(x_1, \ldots, x_k)$. For $1 \leq i \leq k$, let R_i be the projection of the region R on hierarchy T_i, i.e., the set of leaves in T_i in the subtree rooted at x_i. For any pair of regions R, S, it is necessary that for $1 \leq i \leq k$, either $R_i \subseteq S_i$, or $S_i \subseteq R_i$, or $R_i \cap S_i = \emptyset$.

In other words, the only possible type of overlap between projections is containment. This is because if there is an overlap without containment, then there must exist a value $v \in (R_i \cap S_i)$ such that v has at least two parents in the hierarchy. To illustrate this point, let us go back to Figure 8(b) for the spatial case. Consider the projection on the y-axis. The region $\{a, b\}$ when projected on the y-axis gives {4,5}. And the region $\{c, d, e, f\}$ when projected on the y-axis gives {3,4}. Thus, for the spatial case, regions in the optimal covering can have projections that overlap, which complicates matters. For the hierarchical case, the theorem below shows that because of the above property, there is a unique non-redundant, blue-maximal MDL covering. The proof, which is far from trivial, is omitted for brevity. Due to the theorem, we have the following corollary.

Theorem 5.1 Consider k categorical dimensions, each of which is organized in a tree hierarchy $T_i (1 \leq i \leq k)$. There is a unique non-redundant blue-maximal MDL covering.

Corollary 5.1 The unique non-redundant, blue-maximal MDL covering can be constructed on a per hierarchy basis.

5.2 Algorithm MDL-Tree

Based on the above corollary, Algorithm MDL-Tree, shown in Figure 9, first attempts to create a blue-maximal covering on a per dimension/hierarchy basis. For each node d, a list, denoted as $list[d]$, is used to contain all the regions in the current covering involving d. Specifically, when the same region is contained in the lists of *all* the children of d, then the same region can be "generalized" and be added to $list[d]$. This is the key step for achieving blue-maximality. In Figure 9, step (2a) initializes the lists for all the leaves of a hierarchy. Then step (2b) generates the blue-maximal regions in a bottom-up fashion.

Algorithm MDL-Tree

```
1 Initialize $\mathcal{C}$ to contain all the blue cells of the form $(c_1, \ldots, c_k)$.
2 Repeat the following steps on $T_i$ for all $1 \leq i \leq k$:
    (a) For each non-leaf node $d$ in hierarchy $T_j$, initialize $list[d]$ to be empty. For each leaf node $d$ in $T_j$, initialize $list[d]$ to contain all elements in the set:
        $\{(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k) \mid (x_1, \ldots, x_{j-1}, d, x_{j+1}, \ldots, x_k) \in \mathcal{C}\}$.
    (b) Repeat the following step in a post-order traversal of the nodes in $T_j$ (i.e., from the leaves to the root):
            If for all children $d_i$ of $d$, $(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k)$ is in $list[d_i]$, add $(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k)$ to $list[d]$, and add $(x_1, \ldots, x_{j-1}, d, x_{j+1}, \ldots, x_k)$ to $\mathcal{C}$.
3 /* Redundancy check */ Perform this step in a pre-order traversal of the nodes $d$ in $T_k$. In one pass, delete a region $R$ from $list[d]$ if $R$ is contained in $list[e]$ for some ancestor $e$ of $d$, or if $R$ is a descendant of region $S$ for some $S$ also in $list[d]$. At the end, the MDL regions are the ones that remain in $list[d]$ for any node $d$.
```

Figure 9: Pseudo Code for Algorithm MDL-Tree

While at the end of step (2) all blue-maximal regions are created, the covering is not yet non-redundant. Step (3) is designed to remove all redundant regions in a top-down fashion. Starting from the root of the last hierarchy, each list $list[d]$ is checked to remove redundant regions within it. One source of redundancy arises from regions contained in an ancestor's list. To facilitate this operation, we can use a structure to accumulate the regions from all the ancestors of a node d in a pre-order traversal of the nodes. Figure 10 shows a 2-dimensional example to illustrate the algorithm. Nodes in hierarchy T_1 are lettered, while those in T_2 are numbered. As usual, blue cells are marked 'O', red cells marked 'X', and white cells unmarked. The figure illustrates 3 main states of the algorithms.

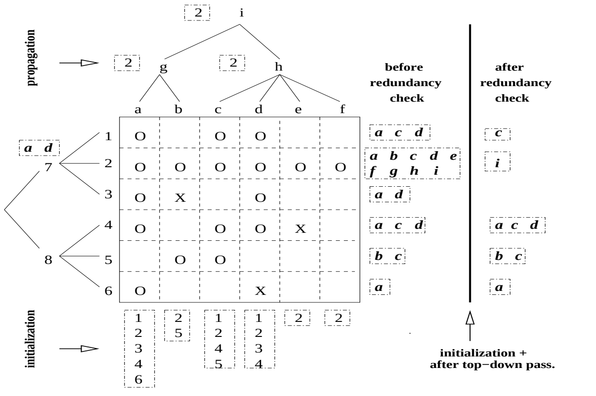


Figure 10: An Example: Before and After Redundancy Check

- First, consider the first iteration of step (2), operating on hierarchy T_1. Step (2a) initializes the lists of the leaves according to the blue cells. For instance, $list[a]$ indicates that $(1, a)$, $(2, a)$, $(3, a)$,

$(4, a)$ and $(6, a)$ are blue, and so on. (The a itself is not recorded in $list[a]$.) Similarly, $list[b]$ contains $2, 5$. Now in step (2b), because 2 appears in both $list[a], list[b]$, 2 is added to $list[g]$, g being the parent of a, b. Similarly, 2 is added to $list[h]$. Finally, 2 is added to $list[i]$, representing the blue-maximal MDL region $(2, i)$.

- Next consider the second iteration of step (2), this time operating on hierarchy T_2. The initialization in step (2a) is now based not only on the blue cells, but also on the results of the previous iteration. A good example is $list[2]$. Not only does it contain a, b, c, d, e, f because of the blue cells, but it also contains g, h, i, because 2 is contained in $list[g], list[h]$ and $list[i]$ from the previous iteration. Then in step (2b), a, d are added to $list[7]$.

  Note that after the first iteration of step (2), no redundancy check is carried out (e.g., removing 2 from $list[a], \ldots, list[h]$ because 2 also appearing in $list[i]$). This is because if redundancy check were to be carried out at this point, the execution of step (2) for subsequent hierarchies would be more complicated. For instance, with the present setup, because a, d appear in $list[2]$ (and for that matter $list[1], list[3]$), a, d are added to $list[7]$ in step (2b). In this manner, step (2b) is completely confined to the current hierarchy. If $list[2]$ contained i only, then the first hierarchy would need to be examined for a, d to be added.

- Finally, after performing step (2) to both hierarchies, redundancy checking occurs in step (3). $List[7]$ still records the blue-maximal regions $(7, a), (7, d)$. Consequently, a, d are removed from $list[1], list[2]$ and $list[3]$. Now for $list[2]$, after a, d are removed, all others except i are removed as well, because i is an ancestor of all of them. Thus, $list[2]$ contains a single i at the end of redundancy checking. Also note that at the end, apart from $(7, a), (7, d)$ and $(2, i)$, all the other MDL blue-maximal regions are merely blue cells, like $(1, c)$, etc.

Lemma 5.1 MDL-Tree visits a node in $T_1, \ldots, T_{k-1}$ once, and a node in T_k twice.

The above lemma is due to the fact that step (2), conducted in a bottom-up fashion, visits a node in each hierarchy once. For the redundancy check in step (3), only the last hierarchy needs to be traversed one more time. Together with Theorem 5.1, this shows that MDL region finding on tree hierarchies can be done in time linear in the sizes of the hierarchies.

It is easy to see that the worst case lower bound for MDL region finding is to visit each node at least once. Thus, the MDL-Tree algorithm presented here is rather close to the ideal situation. Because the last hierarchy is visited twice, a simple optimization is to pick the smallest hierarchy to be the last one, for Corollary 5.1 states that the order of the dimensions used in step (2) is immaterial.

The algorithm assumes we can store any given hierarchy in memory. This is a valid assumption for most data warehousing applications, for which typically the size of the dimension tables is orders of magnitude smaller than the fact table.

5.3 Algorithm GMDL-Tree

Recall that Theorem 5.1 states that there is a unique non-redundant, blue-maximal MDL covering for hierarchical dimensions. However, it is easy to see that when it comes to the optimal GMDL covering for a given white budget w, there may be multiple solutions, depending on the specific white cells chosen. Given Lemma 5.1, our approach to developing an effective heuristic algorithm for the GMDL region finding problem for hierarchical dimensions is to rely on the MDL-Tree algorithm. The proposed GMDL-Tree algorithm first selects white cells, up to the budget w, and then runs MDL on the blue cells as well as the chosen white cells. The white cells are chosen to maximize the reduction of regions in the MDL covering.

Before we show the skeleton of the algorithm, we give an example. Recall from Figure 10 the initial lists for the leaves a, b, c, d, e and f. Based on these lists, we create two tables, one for g and one for h (i.e., the parents of the leaves).

candidate	$(1, h)$	$(2, h)$	$(3, h)$	$(4, h)$	$(5, h)$
occurrence	2	4	1	2	1
max-gain	1	3	0	1	0
cost	2	0	3	X	3

The above table is for h (the more interesting one). The first row is obtained by taking a union of $list[c], \ldots, list[f]$. There is, then, one column for each element/"candidate" in the union. The second row simply counts the occurrences of each candidate in the two lists. For example, $(1, h)$ occurring twice corresponds to the blue cells $(1, c)$ and $(1, d)$. To measure the cost and benefit of each candidate as a choice for adding white cells, we use a pair of values (max-gain, cost). If sufficient white cells are added, the max-gain value calculates the reduction in the number of regions in the GMDL covering. For example, considering candidate $(1, h)$, without the white cells, both $(1, c)$ and $(1, d)$ may be blue-maximal. But if enough white cells are added, the two regions will be covered by $(1, h)$, representing a reduction of one region in the covering. Notice that this is a maximum estimate because the blue cells may be covered by larger regions. For instance, as it turns out, $(1, d)$ is covered by the larger region $(7, d)$. The third row of the table shows the max-gain value for each candidate. In effect, the max-gain value is equal to the number of occurrences minus one. The cost, computed in the fourth row, indicates the number of white cells required to deliver the max-gain. Because h has 4 children in c, d, e, f, the cost is simply 4 minus the number of occurrences. Here there are two interesting cases. First, for candidate $(4, h)$, the X indicates that there is at least a red cell in the region, and is therefore ruled out. Second, for candidate $(2, h)$, the cost is 0, indicating that the entire $(2, h)$ region is already blue, and no more white cell can be chosen. The remaining candidates $(1, h)$, $(3, h)$ and $(5, h)$ move on to the next round for sorting. But before we explain the sorting, we obtain the following tables for the other parents: g for the first hierarchy, and $7, 8$ for the second hierarchy (cf: the lists in Figure 10).

candidate	$(1,g)$	$(2,g)$	$(3,g)$	$(4,g)$	$(5,g)$	$(6,g)$
occurrence	1	2	1	1	1	1
max-gain	0	1	0	0	0	0
cost	1	0	X	1	1	1

candidate	$(7,a)$	$(7,b)$	$(7,c)$	$(7,d)$	$(7,e)$	$(7,f)$
occurrence	3	1	2	3	1	1
max-gain	2	0	1	2	0	0
cost	0	X	1	0	2	2

candidate	$(8,a)$	$(8,b)$	$(8,c)$	$(8,d)$
occurrence	2	1	2	1
max-gain	1	0	1	0
cost	1	2	1	X

Finally, the sorting step is to sort all remaining candidates (i.e., those with a positive cost) in descending order of max-gain, then in ascending order of cost. This corresponds to picking first the regions with the highest max-gain and the lowest cost of white cells. For our example, the sorted candidate list is: $(7, c)$, $(8, a)$ $(8, c)$, $(1, h)$, $(1, g)$, etc. For a given white budget w, the white cells are allocated based on the sorted candidate list, essentially a greedy strategy.

Figure 11 shows a skeleton of the algorithm. While the above example illustrates all the key aspects of the algorithm, step (3) requires further elaboration. As the ranked list is being traversed, let say that candidates $R_1, \ldots, R_w$ have been picked, and the next candidate R is being considered. If R is to be selected, the "true" cost value of R must be within the remaining budget. Note that because in step (1) the cost value of R is computed in isolation, R may overlap with some of the picked candidates $R_1, \ldots, R_w$. To avoid over-counting white cells, the cost value of R may be reduced to reflect the number of extra white cells required to select R. If the reduced cost is still too high for the remaining budget, the next candidate in the ranked list is considered.

Notice that in searching for candidates, step (1) restricts its search space to parents of leaves. Clearly, the search space can be extended to grandparents, great grandparents and so on. While the benefit is that the white cell budget may be spent more aggressively, the tradeoff is that the selection process in step (3) will become more complicated to deal with double counting. Furthermore, a grandparent is more likely than a

```
Algorithm GMDL-Tree

1 Repeat the following steps on $T_i$ for all $1 \leq i \leq k$:
    (a) For each leaf node $d$ in $T_j$, initialize $list[d]$ to contain
        all the blue cells involving $d$.
    (b) For each parent $e$ of a leaf node in $T_j$, do:
        i. Let $d_1, \ldots, d_u$ be all the children of $e$. Create a
           candidate set $S = list[d_1] \cup \ldots \cup list[d_u]$.
        ii. Create a table having one column for each candidate in the
            candidate set $S$. For each candidate, set the occurrence value
            to be the total number of occurrences of the candidate in
            $list[d_1], \ldots, list[d_u]$. For each candidate, set the
            max-gain value to be the number of occurrences - 1. Finally,
            for each candidate, set the cost value to be equal to 'X' if
            the candidate contains a red cell, or to ($u-$ occurrence) otherwise.
2 Collect all those candidates from the above step, omitting those whose
  cost value is either X or 0. Sort all the candidates first in descending
  order of the max-gain value, then in ascending order of the cost value.
3 Select the first $w$ white cells based on the sorted list.
4 Run MDL with the original set of blue cells and the set of the chosen
  white cells (treating them as blue).
```

Figure 11: Pseudo Code for Algorithm GMDL-Tree

parent to encounter a red cell in its leaves. Thus, we restrict the search space to parents only. But to compensate, we allow candidates with the max-gain value 0 to be selected in step (3) if there is enough white budget. This in turn may permit larger regions to be formed. For instance, when $(8, b)$ is picked, $(8, a)$ has already been selected, and the two of them allow the bigger region $(8, g)$ to be formed.

5.4 Experiments

To evaluate the effectiveness of the MDL-Tree and the GMDL-Tree algorithms, we developed a hierarchy generator. The generator produces hierarchies of different fanouts to mimic the hierarchies used in data warehousing benchmarks. For the results shown in Figure 12, there are 3 tree hierarchies, each of the form 1 x 4 x 4 x 6 x 6 (i.e., 4 great grandparents, each having 4 children, etc.). There are 1 million blue cells randomly picked from a clustered region of about 110 leaves per hierarchy (i.e., a blue density of about 75%).

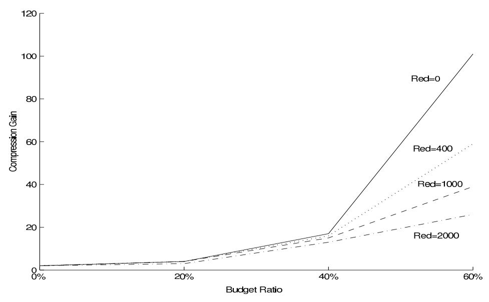


Figure 12: Gain vs White Budget: Hierarchical case

Figure 12 shows the compression gain as the white budget varies from 0% to 60%. Again, when the white budget is 0, this corresponds to MDL summarization. The gain is between 1 and 2 (i.e., the number of MDL regions is at least half the number of blue cells). The key reason for the low gain is that under the MDL principle, if the fanout is 6, say, it only takes 1 out of 6 children to be non-blue to prevent any generalization from happening. In other words, to generalize to a parent, all the 6 leaves must be blue. Because we generate blue cells randomly, this situation does not occur frequently. (It does occur occasionally, as there are some big MDL regions involving grandparent nodes.) In general, the smaller the fanout, the higher the compression gain for MDL.

Compression gain improves dramatically once the white budget is raised. The four curves in the figure show the compression gain relative to the white budget, when the number of red cells are 0, 400, 1000 and 2000. When there is no red cell, the compression gain is over 100 times. As expected, as the number of red cells increases, the compression gain is reduced.

The following table shows the effects of changing dimensionality. Here the number of blue and red cells are kept constant at 1 million and 1000; the white budget is 100% of the number of blue cells.

	3-d	4-d	5-d
compression gain	39.2	26.4	5.9
run time (secs.)	159	514	1573

The results above shows that the GMDL approach is still superior to the MDL approach. The improvement is easily an order of magnitude (e.g., 39 times in the 3-dimensional case). The key reason why the compression gain in the 5-dimensional case shown above is only single digit is that, to provide for a fair comparison, the number of blue cells is kept at 1,000,000. To do so, blue cells are generated from a clustered region of only 17 leaves per hierarchy (i.e., $17^5 > 1,000,000$), as opposed to 110 leaves per hierarchy in the 3-dimensional case. Thus, with this setup, there is relatively little chance for generalization per hierarchy in the 5-dimensional case.

6 Summary and Future Work

In this paper, we study the GMDL region finding problem for numeric data, and both the MDL and the GMDL region finding problem for data with hierarchies. The GMDL approach in both cases is very effective in reducing the number of summaries beyond those found by the MDL approach. The reduction in many cases can be 2 orders of magnitude. In terms of algorithms, we recommend the CAS-Corner algorithm and the BP algorithm for the numeric/spatial case. For lower dimensionality, the top-down splitting CAS-Corner algorithm is the best, whereas for higher dimensionality the bottom-up merging BP algorithm is the choice. As for data with hierarchies, we show that MDL region finding can be solved in time linear in the sizes of the hierarchies, and develop a near optimal MDL-Tree algorithm for it. This algorithm is then extended to form the GMDL-Tree algorithm for GMDL region finding. Experimental results show that the GMDL-Tree algorithm is effective in delivering significant compression.

In ongoing work, we attempt to summarize data with both numeric and hierarchical dimensions. One obvious way to do so is to use the algorithms reported here to first deal with the numeric dimensions, then with the hierarchical dimensions, or vice versa. We hypothesize that region finding in this manner may not be as effective as region finding in a more interleaved fashion.

References

[1] Rakesh Agrawal et al. Automatic subspace clustering of high dimensional data for data mining applications. *ACM SIGMOD*, May 1998, pp. 94-105.

[2] S. Berchtold, D. Kiem and H. P. Kriegel. The X-tree: an Index Structure for High-dimensional Data. *VLDB 1996*, pp. 28–39.

[3] Kevin S. Beyer, Raghu Ramakrishnan. Bottom-Up Computation of Sparse and Iceberg CUBEs. *ACM SIGMOD Conference 1999*, pp. 359-370.

[4] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, Jeffrey D. Ullman. Computing Iceberg Queries Efficiently. *VLDB 1998*, pp. 299-310.

[5] Yvan J. Garcia, Mario A. Lopez and Scott Leutenegger. On Optimal Node Splitting for R-trees. *VLDB 1998*, pp. 334–344.

[6] Haisong Gu, Yoshiaki Shirai, Minoru Asada. MDL-Based Segmentation and Motion Modeling in a Long Image Sequence of Scene with Multiple Independently Moving Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(1), pp. 58-64, 1996.

[7] A. Guttman. R-trees: a Dynamic Index Structure for Spatial Searching. *ACM SIGMOD 1984*, pp. 47–57.

[8] Pekka Kilpelinen, Heikki Mannila, Esko Ukkonen. MDL learning of unions of simple pattern languages from positive examples. *EuroCOLT 1995*, pp. 252-260.

[9] Manish Mehta, Jorma Rissanen, and Rakesh Agrawal. MDL-based decision tree pruning. *KDD 1995*, pp. 216-221.

[10] Raymond T. Ng, Alan Wagner and Yu Yin. Iceberg-cube computation with PC Clusters. *ACM SIGMOD 2001*, pp. 25–36.

[11] R. A. Reckhow and J. Culberson. Covering Simple Orthogonal Polygon with a Minimum Number of Orthogonally Convex Polygons. *ACM Annual Computational Geometry Conference*, pp. 268–277, 1987.

[12] Eric S. Ristad and Robert G. Thomas. Context models in the MDL framework. *Data Compression Conference 1995*, pp. 62-71.

[13] J. Ross Quinlan, Ronald L. Rivest. Inferring Decision Trees Using the Minimum Description Length Principle. *Information and Computation*, 80(3), pp. 227-248, 1989.