MDL Summarization with Holes

Shaofeng Bu
Univ. of British Columbia
sfbu@cs.ubc.ca

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

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

Abstract

Summarization of query results is an important problem for many OLAP applications. The Minimum Description Length principle has been applied in various studies to provide summaries. In this paper, we consider a new approach of applying the MDL principle. We study the problem of finding summaries of the form $S \ominus H$ for k-d cubes with tree hierarchies. The S part generalizes the query results, while the H part describes all the exceptions to the generalizations. The optimization problem is to minimize the combined cardinalities of S and H. We first characterize the problem by showing that solving the 1-d problem can be done in time linear to the size of hierarchy, but solving the 2-d problem is NP-hard. We then develop three different heuristics, based on a greedy approach, a dynamic programming approach and a quadratic programming approach. We conduct a comprehensive experimental evaluation. Both the dynamic programming algorithm and the greedy algorithm can be used for different circumstances. Both produce summaries that are significantly shorter than those generated by state-of-the-art alternatives.

1 Introduction and Motivation

It is well known that complex aggregate queries involving millions of records are one of the hallmarks of OLAP-style data analysis applications running on data warehouses. Phenomenal strides have been made over the past decade in the development of efficient algorithms for computing the data cubes [4, 17], in materialization of the cubes [5], in approximation [16], in decomposition [9], and in compression [15]. An equally important topic is the summarization of the query results. Instead of returning individual tuples satisfying a specified set of querying conditions, the tuples are summarized into "rollup regions" using non-leaf nodes in the hierarchies associated with the dimensions [14, 8]. The Minimum Description Length Principle (MDL) [11] is often used to do so.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

**Proceedings of the 31st VLDB Conference,
Trondheim, Norway, 2005**

Figure 1 shows a 2-dimensional data cube over the dimensions `location` and `clothes`. The base table contains the volume of sales of every type of clothing item in every location. The figure also shows the hierarchies. Suppose a user asks the query "which locations grossed a sales volume of over 100,000 for any clothing item?" This is an aggregate selection query. The cells that satisfy this query is marked ◯, and we call them the blue cells. There are 36 blue cells in this example. The MDL approach exploits rollups and uses such regions as (`SanFrancisco`,`men's`) to summarize the 4 blue cells it covers. There are 5 other MDL regions(i.e.,(`Summit`,`men's`),(`northwest`,`formal wear`),(`northeast`,`men's jeans`),(`northeast`, `dress pants`), (`northeast`,`dress shirts`)) and 15 single blue cells left. The MDL summary reduces the original length from 36 to 21. Thus, the user gets a more succinct and intuitive answer.

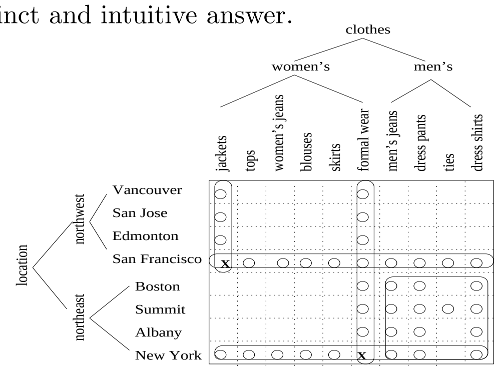


Figure 1: Motivation Example

Note that an MDL region consists of blue cells only. One approach that we have studied in [8] to get additional summarization is the Generalized MDL approach. The idea is that by allowing non-blue cells to be covered in the regions, the description length can be further reduced. However, non-blue cells represent impurities. GMDL *controls* the situation in two ways. First, there are red cells that cannot be covered by any region. For our example, any cell with sales below 80,000 may be coloured red. Second, there are white cells, which are not that different from blue cells, and can be covered in the regions, but only up to a maximum white budget. For example, any cell with sales between 80,000 and 100,000 may be coloured white. In Figure 1, red cells are marked X, and white cells are unmarked. With a white budget of 3 cells, GMDL can use the region (`northeast,men's`). Then the GMDL summary offers a length of 18, instead of 21.

In this paper, we ask whether there is a more effective mechanism to control impurities. We explore whether a better control mechanism is to explicitly identify the non-blue cells. Specifically, we consider summaries of the form: $S \ominus H$, where S is the set of regions covering all the blue cells, and H explicitly identifies the non-blue cells, also called *holes*, that are covered in S. We call this an MDLH summary and its length is the sum of the cardinalities of S and H. In stark contrast to GMDL, there is no need to color non-blue cells as red or white, and there is no white budget to set. There is also no extra post-processing step to identify which impure white cells are included in a GMDL summary. MDLH can pick any non-blue cell but its inclusion in H is counted towards the length. We believe that the MDLH framework is easier for the user to relate to. The open question is whether it gives shorter summaries. For the example in Figure 1, the optimal MDLH summary includes:

- 5 regions: (`location, formal wear`), (`San Francisco, clothes`), (`northwest, jackets`), (`New York, clothes`), and (`northeast, men's`), all of which are marked in the figure; and
- 5 holes: (`New York, ties`), (`New York, formal wear`), (`San Francisco, jackets`), (`Boston, ties`), and (`Albany, ties`).

The total length of the MDLH summary is 10. Note that in some applications where approximate query answering is acceptable, we envisage that the user may be satisfied seeing the S part (i.e., only 5 regions in our example), and the H part is shown only in "drill-down" mode.

In this paper, *we study the problem of finding optimal MDLH summaries* for aggregate selection queries on k-d data cubes, where the dimensions have tree hierarchies defined on them. We make the following contributions.

- We characterize the hardness of finding optimal MDLH summaries. The first positive, but unsurprising, result is that for a 1-d data cube, the optimal MDLH summary can be found in time linear in the size of the hierarchy. However, the situation quickly deteriorates. We show that even for 2-d and 2-level hierarchies, finding the optimal MDLH summary is NP-hard (Section 2).
- We develop three heuristic algorithms based on greedy, dynamic programming, and quadratic programming approaches. The algorithmic development is non-trivial and we illustrate how they work with examples (Section 3).
- We ran a comprehensive set of experiments on the TPC-H benchmark data set to measure: (i) the added value of considering MDLH for summarization; and (ii) the effectiveness of the proposed heuristics, with respect to summary sizes and runtimes. We show that MDLH is a very promising approach (Section 4).
- To further reduce the length of an MDLH summary, we apply the MDLH algorithms virtually unchanged on the H part. That is, we now describe H as $S_H \ominus H_H$. The final summary is, thus, of the form: $S \ominus S_H + H_H$ (Section 5).

1.1 Related Work

In a series of papers [12, 13, 14], Sarawagi et al. study the problem of automating data explorations around specific tuples of interest to the user. In [14], they consider the problem of finding maximal consistent generalizations of "interesting" tuples. The summary permits exceptions just like MDLH. Their framework requires the specification of an upper bound on the number of exceptions, and an error function that charges for the inclusion of "wrong" tuples and the exclusion of other "right" tuples in the generalization. In their generalization step, they exploit Apriori style pruning, since a tuple can be generalized along a set of dimensions only if it can be generalized along all subsets of dimensions. This property does not hold when we want to optimize description length. Indeed, we can get global optimality without the sub-regions satisfying local optimality.

In [10], Mendelzon and Pu consider MDL descriptions for subsets of structured sets. They show that for k-d hierarchies, the problem is NP-hard. One similarity between our framework and theirs is that exceptions are allowed. However, their framework is more expressive (hence, an NP-hardness result is less surprising) in that they allow Cartesian products to be formed. For example, in a 2-d cube, $(a + b) \times w$ is a valid expression to cover two cells (a, w) and (b, w), where a, b are leaves in the first hierarchy, and w is a leaf in the second. Thus, their framework is not purely hierarchical because $(a+b)$ may not correspond to any non-leaf node in the hierarchy. Furthermore, algorithmic development is not the focus of their work.

In [8], we study GMDL summarization. A major part of that study concerns the spatial case when there is no hierarchy to restrict the formation of regions. For the hierarchical case, the MDL-Tree and GMDL-Tree algorithms are proposed. These two algorithms are included in our experimental evaluation in Section 4, which shows that MDLH produces significantly shorter summaries. Description length aside, the MDLH region finding problem studied here is novel. Technically, characterizing the MDLH problem is challenging; and algorithmically, the MDLH heuristics are very different from the existing algorithms. Last but not least, the MDLH approach offers a summarization framework easier for the user to understand.

2 Characterizing the Problem

2.1 Problem Definition

Let there be k categorical dimensions with each dimension consisting of a finite number of members associated with a hierarchy. In this paper, we restrict our attention to tree hierarchies only. For the most part, we assume that all the leaf nodes appear in the same level. This assumption is satisfied with many real life examples we encounter, and is not technically important; it just serves to simplify the discussion whenever possible. However, in Section 5, we explicitly consider situations not meeting this assumption.

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 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 $1 \leq i \leq k$.

Given a collection D of blue cells (i.e., cells of interest), an *MDL summary or covering* is a set of regions that cover all the blue cells, and that each region covers blue cells only. An *MDLH summary or covering* ("H" for "Holes") is of the form: $S \ominus H$, where:

- S is a set of *regions* covering all the blue cells;
- H is a set of *non-blue cells* (i.e., the holes); and
- a cell c is in H iff there exists a region x in S such that x covers c and c is non-blue.

Thus, an MDLH covering with an empty H part is an MDL covering. The *description length* of an MDLH summary is the sum of cardinalities of S and H, i.e., $|S| + |H|$. The *MDLH region finding problem* is to find an MDLH covering $S \ominus H$ of the smallest description length. The decision problem is whether there is an MDLH covering of length $\leq K, K > 0$.

2.2 Warming Up: Solving the 1-d Problem

Let us introduce the notion of *benefit*. For a given set D of blue cells to be covered, let $S \ominus H$ be an MDLH summary of D. Then the benefit of the summary is:

$$Ben(S \ominus H) = |D| - (|S| + |H|).$$

In other words, given a fixed set D of blue cells, minimizing the length of $S \ominus H$ is equivalent to maximizing the benefit of $S \ominus H$. We can specialize the above definition to determine the benefit of a single region. Given a k-d region $x = (x_1, \ldots, x_k)$, let $Cell(x)$ denote the set of all cells that it covers. Moreover, let $BlueCell(x)$ and $NonBlueCell(x)$ denote the sets of blue cells and non-blue cells in $Cell(x)$ respectively. Then the benefit of x is:

$$Ben(x) \;\;=\;\; |BlueCell(x)| - (|NonBlueCell(x)| + 1) \quad (1)$$

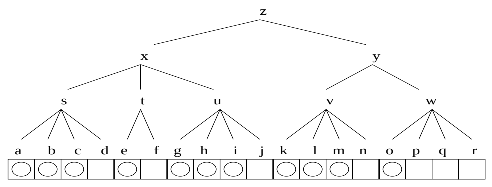


Figure 2: An Example 1-d Problem

Let us use the 1-d example shown in Figure 2 to illustrate. It is a 4-level hierarchy counting both the root and the leaves. Consider internal node u. $Ben(u)$ = 3 - (1+1) = 1. This means that between the two options of $\{g, h, i\}$, and $(\{u\} \ominus \{j\})$, the latter has a net reduction in length of 1.

Notice that Equation (1) extends to the boundary case when x itself consists of a single cell. If x is a single blue cell, then $Ben(x) = 0$; otherwise, $Ben(x) = -2$. The following lemma is handy for evaluating $Ben(x)$ inductively and is used in the ensuing analysis extensively. The lemma is straightforward.

Lemma 2.1 Let x be a k-d region $x = (x_1, \ldots, x_k)$, such that there is an x_i which is an internal node in hierarchy T_i. Let $Chd(x_i)$ denote the child nodes of x_i. Then: $Ben(x) = |Chd(x_i)| - 1 + \Sigma_{x_{ij} \in Chd(x_i)} Ben(x')$, where $x' = (x_1, \ldots, x_{i-1}, x_{ij}, x_{i+1}, \ldots, x_k)$. ▌

To illustrate the lemma, let us consider internal node x in Figure 2. A direct application of Equation (1) gives: $Ben(x) = 7 - (3 + 1) = 3$. However, applying the above lemma indicates that $Ben(x) = 3 - 1 + Ben(s) + Ben(t) + Ben(u)$. A further examination reveals that $Ben(s) = 1$, $Ben(t) = -1$, and $Ben(u) = 1$. Thus, $Ben(x) = 3 - 1 + 1 - 1 + 1 = 3$. Notice that even though $Ben(t)$ is negative, the parent of t has a positive benefit. This suggests that in the worst case, every node in the hierarchy may need to be examined. In terms of extracting the optimal covering, node x is chosen (i.e., $\{x\} \ominus \{d, f, j\}$), because there is a net gain in benefits from its children. In contrast, node y is not selected and the optimal descriptions of its children are used instead. That is, for child v, the covering is $\{v\} \ominus \{n\}$, and for child w, the covering is $\{o\}$. Thus, the final optimal covering is $\{x, v, o\} \ominus \{d, f, j, n\}$.

The 1-d situation is not of practical OLAP value; the discussion here only serves to introduce key aspects of the MDLH problem. We omit the details of an algorithm for finding optimal MDLH covering for the 1-d case; see [3] for details.

Lemma 2.2 The 1-d MDLH region finding problem can be solved in $O(|h|)$ time, where h is the number of nodes in the hierarchy. ▌

2.3 Understanding the Hardness of the 2-d Problem

The tractability of the MDLH region finding problem deteriorates rapidly from 1-d to 2-d. Let us consider the following example. Figure 3 shows a 7×7 2-d, 2-level cube. There are 28 blue cells. In this simple example, the optimal MDL covering merely enumerates all the 28 blue cells.

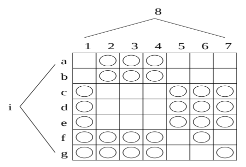


Figure 3: An Example 2-d Problem

Applying the earlier definition of computing benefits for regions, the following table shows the benefits for all the rows and columns.

rows or columns	benefits
(f,8),(g,8)	2
(c,8),(d,8),(e,8)	0
(a,8),(b,8)	-2
(i,1)	2
(i,2),(i,3),(i,4),(i,6),(i,7)	0
(i,5)	-2

If we are to apply a greedy strategy based on the tables above, we would pick $(f,8)$, $(g,8)$ and $(i,1)$. The rest would be the individual cells. The part of the covering corresponding to $(f,8)$ would be $\{(f,8)\}\ominus \{(f,5),(f,7)\}$, for a length of 3 and a benefit of 2. Similarly, the length of the covering corresponding to $(g,8)$ and $(i,1)$ is also 3. Thus, the greedy approach would generate an MDLH covering of length 24, yielding a benefit of 4 over listing all the blue cells.

Obviously, treating columns and rows separately causes double counting. For example, $(i,1)$ intersects with both $(f,8)$ and $(g,8)$. The benefits of the intersected cells $(f,1)$ and $(g,1)$ are doubly counted. Then the question is: if we were more careful with counting benefits, could we fix the greedy approach to give the optimal solution? For example, if we have picked $(f,8)$, we may need to re-calculate the benefit of $(i,1)$.

Unfortunately, re-calculation of benefits does not solve the problem. Let us examine the optimal solution of the example: $\{(c,8),(d,8),(e,8),(i,2),(i,3),(i,4)$ $(f,1),(g,1),(f,6),(g,7)\}\ominus \{(c,2),(c,3),(c,4),(d,2),(d,3),$ $(d,4),(e,2),(e,3),(e,4)\}$. That is, the optimal MDLH covering is based on picking the rows of c,d,e and the columns of $2,3,4$. The odd thing is that none of $(i,1)$, $(f,8)$ and $(g,8)$ is part of the optimal covering. In contrast, even though rows $(c,8),(d,8),(e,8)$ and columns $(i,2),(i,3),(i,4)$ do not have positive benefits, we can get more benefits by selecting *all of them as a group.*

In the 1-d problem, the computation of the optimal MDLH covering is local in the sense that whether an internal node is selected is based on the benefits of its parent, itself and its children. In the 2-d problem, rows and columns interact in complex ways, such as the rows of c,d,e and the columns of $2,3,4$ discussed above. Thus, the tractability worsens in a hurry.

2.4 Proving NP-hardness

In the remainder of this section, we show that the MDLH region finding problem is NP-hard even for two dimensions/hierarchies and each hierarchy only has two levels. We show a proof outline in two steps. First, we show that it is NP-hard to find a maximum induced subgraph in a complete edge-weighted (CEW) bipartite graph, which has 2-value weights on edges. Then we show how the CEW problem can be reduced to the MDLH region finding problem.

A complete edge-weighted (CEW) bipartite graph with 2-value weights is of the form $G_e = (V,E), V = V_1 \cup V_2$. For any $v_i \in V_1, v_j \in V_2$, the edge $e = (v_i, v_j)$ is in E. Furthermore, the weights on edges must be one of two possible values, i.e., $wt(e) \in \{w_1, w_2\}$, and at least one of w_1, w_2 is negative. See Figure 4(a) for an example.

For any subgraph $G'_e = (V', E')$ in G_e induced by $V' \subseteq V$, the weight of G'_e is defined as $wt(G'_e) = \sum_{e\in E'} wt(e)$. The *maximum induced subgraph problem* in CEW bipartite graph is to find a subgraph $G'_e = (V', E')$ induced by $V' \subseteq V$ such that $(wt(G'_e) + |V'|)$ is maximized. One can regard the graph also as node weighted with each node having unit weight, so we just compute the weight of the whole graph (or subgraph) including both node weights and edge weights.

The proof of the following lemma is similar to the one used in [6] for showing that the decision problem of maximum edge-weighted biclique in a bipartite graph is NP-complete. See [3] for details.

Lemma 2.3 The following decision problem associated with an induced subgraph in a CEW bipartite graph is NP-complete: *Does G_e contain an induced subgraph $G'_e = (V', E')$ such that $|V'| + wt(G'_e) \geq K$?* ∎

To show that the decision problem associated with the MDLH region finding problem is NP-complete, we exhibit a reduction to the maximum induced graph problem in CEW bipartite graph. While we suppress the technical details of the proof, we show an example of how to transform a given CEW bipartite graph into a corresponding 2-d MDLH region finding problem.

Figure 4 shows a CEW bipartite graph $G = (V,E)$. Note that $|V| = 9$ and $wt(G) = -4$. Suppose that $K = 9$, i.e., we want to find an induced subgraph $G' = (V', E')$ such that $|V'| + wt(G') \geq 9$.

We can construct a 2-d, 2-level MDLH region finding problem as in Figure 4. The cells marked by '◯' are blue cells, and correspond to the edges with weight -1 in graph G. The unmarked data cells are non-blue cells and correspond to the edges with weight 1 in graph G. For the given induced subgraph decision

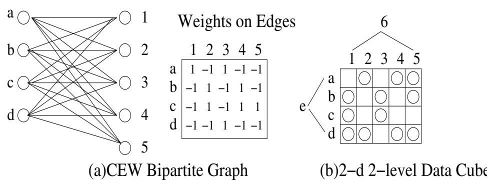


Figure 4: Example of Reduction

problem with $K = 9$, the corresponding MDLH decision problem is with $K' = -(|V| + wt(G)) + K = -(9-4)+9 = 4$, i.e, we want to find whether there is an MDLH covering with a benefit equal to 4 with respect to enumerating all the individual cells.

Consider $\{(a,6),(d,6),(e,1),(e,3),(b,5)\} \ominus \{(a,1),(a,3),(d,3)\}$. This covering is of length 8, representing a net gain/benefit of 4. Essentially, the covering takes the a,d rows and the $1,3$ columns. This covering in turn gives the induced subgraph $G' = (V',E')$ such that $V' = \{b,c,2,4,5\}$, induced by the complement of the chosen rows/columns. As shown in the weight matrix in Figure 4(a), the weight of G' is $wt(G') = 5 - 1 = 4$. In other words, the quantity $|V'| + wt(G')$ is equal to $5 + 4 = 9$.

Theorem 2.1 The following decision problem associated with the MDLH region finding problem is NP-complete. *Given a 2-d, 2-level data cube with blue cells, and an integer M, is there an MDLH covering with description length $\leq M$?* ∎

A proof of the above theorem can be found in [3]. The theorem says that even for a 2-d, 2-level data cube, the MDLH region finding problem is already NP-hard.

3 Heuristic Algorithms

In this section, we develop heuristic algorithms for the MDLH region finding problem. A greedy approach is a very common strategy for heuristically solving NP-hard problems; so we begin with a greedy algorithm. Then we develop two more heuristics based on a dynamic programming approach and a quadratic programming approach. Experimental results will be presented in the Section 4 comparing these heuristics.

3.1 MDLH-Greedy: a Greedy Approach

Recall the greedy approach discussed in Section 2.3. The idea is to order regions in descending order of benefits. Each time, the region with the most benefit is selected till all the remaining regions have negative benefits. A skeleton of this heuristic is shown in Figure 5.

Figure 6 gives a 2-d problem that we use as a running example. In step 2, all the parent regions are generated, i.e., regions with one parent node from one hierarchy but all the other nodes are leaves in the remaining hierarchies (e.g., $(e,1)$, $(a,10)$, but not $(e,10)$ and $(12,a)$). The table below sorts the regions in descending order of benefits and ascending order of the number of holes.

```
Algorithm MDLH-Greedy
Input: a k-d data cube and the set of blue cells D
Output: an MDLH description for D
  1. /* H and H' are sets of holes. $S \ominus H$ is a description for D.
     Each time we select a region x, S' contains the blue cells in
     S but not in x. H' contains holes in H and x. $S' \ominus H'$ is
     the new description by selecting x.*/
     Initialize $S = D$ and $H = S' = H' = \emptyset$.
  2. Compute the benefit of each parent region and sort those regions
     with non-negative benefits to form a sorted list in descending
     order of benefits but in ascending order of the number of holes.
  3. Obtain the next best unprocessed region x from the sorted list.
     Set $S' = S - BlueCell(x)$, and $H' = H + NonBlueCell(x)$.
  4. If $(|S'| + |H'| < |S| + |H|)$ then { $S = S'$ and $H = H'$ }.
  5. If there is any unprocessed region in the sorted list, then go
     to step 3.
  6. Mark each cell in H as blue. Apply MDL-Tree algorithm on
     all the blue cells and the description generated is stored in S.
  7. Return $S \ominus H$ as the MDLH description for D.
```

Figure 5: A Skeleton Algorithm for MDLH-Greedy

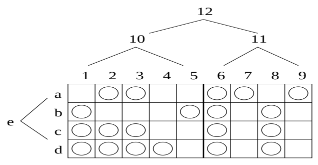


Figure 6: An Example 2-d MDLH Problem

regions	benefits	holes
(e,6)	3	-
(d,10)	2	(d,5)
(e,1)	1	(a,1)
(e,2)	1	(b,2)
(e,3)	1	(b,3)
(e,8)	1	(a,8)
(a,11)	1	(a,8)

Within the first two iterations of the loop between steps 3 and 5, the regions $(e,6)$ and $(d,10)$ are selected. At this point, the state is: $S = \{(e,6),(d,10),(a,2),(a,3),(b,1),(b,5),(c,1),(c,2),(c,3),(a,7),(a,9),(b,8),(c,8),(d,8)\}$. And, $H = \{(d,5)\}$, as $(d,5)$ was added in step 3 when $(d,10)$ was selected.

Breaking ties, let us say that the next best region is $(e,1)$. Then the addition of $(e,1)$ will cause $(b,1)$ and $(c,1)$ to be deleted from the above S. Note that $(d,1)$ has already been deleted from S when $(d,10)$ was added. In essence, the benefit of $(e,1)$ is now updated to 2 - 2 = 0. This is an important point because in step 4, when a selection has been accepted, there is no *explicit* re-computation of benefits.[1] Instead, step 3 implements the benefit adjustment only when the region has been selected. To return to $(e,1)$, this change brings no benefit and is not accepted in step 4.

The situation is similar for $(e,2)$ and $(e,3)$. Eventually, $(e,8)$ and $(a,11)$ are selected. The final output is: $\{(e,6),(d,10),(a,11),(e,8),(a,2),(a,3),(b,1),(b,5),$

[1]It is straightforward to adapt the algorithm so it does these updates. In our preliminary experimentation, we implemented a variant of the greedy heuristic with an explicit benefit update step. We found that this variant offered little improvement over the version of MDLH-Greedy presented here, but its runtime was significantly higher.

$(c,1),(c,2),(c,3)\} \ominus \{(d,5),(a,8)\}$. The length is 13.

Note that the outcome is different when $(a,11)$ is processed before $(e,8)$. The output of MDLH-Greedy in general depends on the tie-breaking policy. Also note that we restrict candidate regions to parents of leaves. Clearly, we can also extend the candidates to grandparents (e.g., $(a,12)$), and parents with parents (e.g. $(e,10)$), and so on. The upside is that potentially higher benefits can be obtained. The downside is that we have more regions to process. Furthermore, the need for $(a,12)$ may already be met by the child regions of $(a,10)$ and $(a,11)$. The only problem is that if both $(a,10)$ and $(a,11)$ are included in S, then it would be beneficial to generalize to $(a,12)$. Step 6 is to cover this situation by applying the MDL-Tree algorithm presented in [8] that gives optimal MDL regions (i.e., regions covering only blue cells) with tree hierarchies.

In terms of complexity, let n be the total number of cells. Then there are at most kn parent regions. The sorting in step 2 requires $O(kn \log kn)$ time. The loop between step 3 and 5 takes $O(kn)$ time in total. Finally, as given in [8], the complexity for generating optimal MDL regions is $O(kt)$, where t is the maximum number of nodes in one of the k hierarchies. As n is expected to be much larger than t, the overall complexity of MDLH-Greedy is $O(kn \log kn)$.

3.2 MDLH-Dynamic: a Dynamic Programming Approach

Dynamic programming is also a widely used method to approximately solve NP-hard problems [1]. The core of the dynamic programming approach is that for each region, we consider the optimal solutions of the child sub-problems, and piece together these solutions to form a candidate solution for the original region. Specifically, given a region $x = (x_1, \ldots, x_k)$, if x_i is a non-leaf node in hierarchy T_i, then one candidate solution for x is:

$$MDLH(x,i) = \cup_{x_{ij} \in Chd(x_i)} MDLH((x_1, \ldots, x_{i-1}, x_{ij}, x_{i+1}, \ldots, x_k)) \quad (2)$$

This is called the "local" solution from hierarchy T_i. In general, there may be as many as k local solutions. Furthermore, there is also a "global" solution, which is simply taking x and enumerating all the holes covered by x:

$$MDLH(x, global) = \{x\} \ominus NonBlueCell(x) \quad (3)$$

Thus, among the candidate solutions: $MDLH(x, global)$, $MDLH(x,1), \ldots, MDLH(x,k)$, the one with the minimum length is chosen. A skeleton of this algorithm is shown in Figure 7.

Let us continue with the 2-d problem in Figure 6. We build two matrices: Len for measuring the length of the partial description, and Sel for recording whether a global solution is chosen or which of the local solutions is picked. In the Len matrix, the entries

```
Algorithm MDLH-Dynamic
Input: a k-d data cube and the set of blue cells D
Output: an MDLH description for D
  1. Allocate space for a k-d matrix $Len$ of size $t_1 \times \ldots \times t_k$, where
     $t_i$ is the number of nodes in hierarchy $T_i$. Label the nodes
     in each hierarchy $T_i$ by post-order. Allocate space for a k-d
     matrix $Sel$ in the same way.
  2. For any data cell $x = (x_1, x_2, \ldots, x_k)$, if $x$ is a blue cell, then
     set $Len(x_1, x_2, \ldots, x_k) = 1$, otherwise $Len(x_1, x_2, \ldots, x_k) =$
     0. The corresponding entry in $Sel$ is left empty.
  3. For any data region $x = (x_1, x_2, \ldots, x_k)$:
     (a) For any $1 \le i \le k$, if $x_i$ is a non-leaf, apply Equation
         2 to construct a local solution. The length of the
         local solution can be obtained by summing up the corresponding
         entries in the $Len$ matrix.
     (b) Construct a global solution based on Equation 3.
     (c) Pick the minimum length description from the
         above candidates and enter the minimum length as
         $Len(x_1, \ldots, x_k)$.
     (d) If the global solution gives the minimum length, then
         set $Sel(x_1, \ldots, x_k)$ as $g$. Otherwise, if the minimum
         length comes from the local solution corresponding to
         hierarchy $T_i$, set $Sel(x_1, \ldots, x_k)$ as $t_i$.
```

Figure 7: A Skeleton Algorithm for MDLH-Dynamic

correspond to the length of the solution that is picked at every stage, while the Sel matrix keeps track of the solution picked. An entry 'g' means a global solution is picked while 't_i' means a local solution, corresponding to hierarchy T_i $(i = 1, 2)$ is picked.

Len	1	2	3	4	5	10	6	7	8	9	11	12
a	0	1	1	0	0	2	1	1	0	1	2	4
b	1	0	0	0	1	2	1	0	1	0	2	4
c	1	1	1	0	0	3	1	0	1	0	2	5
d	1	1	1	1	0	2	1	0	1	0	2	4
e	2	2	2	1	1	8	1	1	2	1	5	13

Sel	1	2	3	4	5	10	6	7	8	9	11	12
a						t_2					g	t_2
b						t_2					t_2	t_2
c						t_2					t_2	t_2
d						g					t_2	t_2
e	g	g	g	t_1	t_1	t_2	g	t_1	g	t_1	t_2	t_2

We enumerate the nodes in each hierarchy by post-order. The first hierarchy enumerates the nodes in the order a, b, c, d, e, and the second in the order of 1, 2, 3, 4, 5, 10, etc. In step 2 of the algorithm, all data cells are populated with 1 or 0 depending on whether they are blue or not. This takes care of the entries for a, b, c, d and $1, \ldots, 5, 6, \ldots, 9$.

Next we consider the parent regions. For region $(a,10)$, in step 3, the local solution of $\{(a,2),(a,3)\}$ is compared with the global solution of $\{(a,10)\} \ominus \{(a,1),(a,4),(a,5)\}$. There is only one local solution because node a is a leaf. In this case, the local solution dominates. Thus, the entry for $(a,10)$ in the Len matrix is 2, and the entry in the Sel matrix is t_2, indicating that the optimal solution comes from the second hierarchy (i.e., the children of node 10).

The situation is similar for $(b,10)$ and $(c,10)$, but $(d,10)$ is different. In this case, the local solution is $\{(d,1),\ldots,(d,4)\}$, whereas the global solution is $\{(d,10)\} \ominus \{(d,5)\}$. The latter dominates, giving rise to the entry g in the Sel matrix for $(d,10)$.

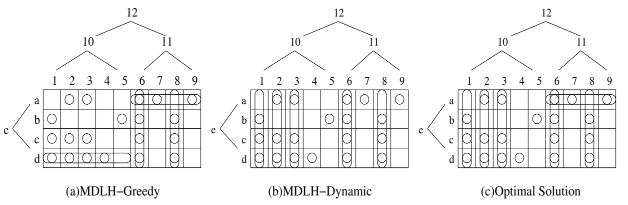


Figure 8: Solutions based on the Three Approaches

Notice that when a local solution is picked, a row (or a column) sum gives the corresponding entry in the *Len* matrix (e.g., $Len(c, 10) = Len(c, 1) + Len(c, 2) + Len(c, 3)$). However, when a global solution is selected, the corresponding length is less than the row sum (e.g., $Len(d, 10)$). The situation is similar for $(e, 1), (e, 2), (e, 3)$, all with the global solutions dominating.

Lastly, let us consider $(e, 10)$. There are two local solutions: one by row and one by column. The former gives a length of 8 (i.e., row sum in the *Len* submatrix) and the latter gives a length of 9 (i.e., column sum). There is also the global solution with length 10. Thus, for the *Sel* matrix, the entry for $(e, 10)$ is t_2, indicating that the solution comes from the children of node 10. Similarly, other entries are filled out.

The solution can be reconstructed from the *Sel* matrix. To see this, let us begin with $(e, 12)$ whose entry is t_2, meaning that node 12 should be expanded to its children nodes 10 and 11. For both $(e, 10)$ and $(e, 11)$, their entries in *Sel* indicate expansion on the second hierarchy again. Thus, for $(e, 10)$, the expansion points to $(e, 1), \ldots, (e, 5)$. The entry g for $Sel(e, 1)$ indicates that Equation 3 applies, i.e., $MDLH((e, 1), global) = \{(e, 1)\} \ominus \{(a, 1)\}$. Constructed in similar fashion, the final solution is: $\{(e, 1), (e, 2), (e, 3), (d, 4), (b, 5), (e, 6), (a, 7), (e, 8), (a, 9)\} \ominus \{(a, 1), (b, 2), (b, 3), (a, 8)\}$. The length is 13 which happens to be equal to the length of the solution obtained from greedy, although the solutions are different as shown in Figure 8. With a length of 12, the optimal solution differs from the dynamic programming solution for $(e, 11)$. Specifically, for $(e, 11)$, the optimal solution is: $\{(e, 6), (e, 8), (a, 11)\} \ominus \{(a, 8)\}$. Notice that the optimal description calls for the combination of a row $(a, 11)$ and columns $(e, 6), (e, 8)$, which is not possible under Equation 2 and Equation 3. This is the reason why the dynamic programming approach may not be optimal.

Algorithm MDLH-Dynamic describes how the *Sel* matrix is constructed, but does not indicate how the solution can be extracted from the matrix. As shown from the above example, the extraction is standard; for space reasons, we omit the details.

In terms of complexity, let t be the maximum number of nodes in one of the k hierarchies. Then the matrices has t^k entries. For step 3, there may be k local solutions, each of which corresponds to f children, where f is the maximum fanout. Hence, the complexity of MDLH-Dynamic is $O(k\ f\ t^k)$.

3.3 MDLH-Quad: a Quadratic Programming Approach

Recall from the previous example that the dynamic programming solution for region $(e, 11)$ is not optimal because it is not capable of picking a combination of rows and columns (i.e., $(e, 6), (e, 8)$ and $(a, 11)$). For this particular example, quadratic programming gets the optimal solution; this motivates the development of MDLH-Quad, a quadratic programming heuristic.

A *quadratic programming* problem is of the form [2]:

$$\begin{aligned} Minimize \qquad & f(y) = \tfrac{1}{2} y^T Q y + c y \\ s.t. \quad & Ay \geq b, By = e, lb \leq y \leq ub \end{aligned}$$

where y is a vector of real variables, and Q, A, B are matrices. It has been proved that a quadratic function $f(y)$ is convex *iff* Q is positive semi-definite [2]. In the ensuing discussion, we describe how to set up a quadratic programming problem to solve the MDLH region finding problem. To make the discussion easier to follow, we use the 2-d problem to illustrate.

To solve a given 2-d problem, let y denote a vector of variables, one for each row (u_i) and one for each column (v_j), such that $0 \leq u_i, v_j \leq 1$. Let R_i denotes the i-th row and C_j denotes the j-th column. Then the term $u_i * Ben(R_i)$ gives the benefit of picking the i-th row when u_i is 1. When $u_i = 0$, there is no benefit contribution from this row. The situation is similar for the columns. One complication is that if both the i-th row and the j-th column are selected, we have double-counted the cell (i, j). In particular, if (i, j) is a blue cell, we have double-counted its positive contribution. If (i, j) is a non-blue cell, we have double-counted its negative contribution. Thus, the benefit function can be formalized as follows:

$$\begin{aligned} TotBen \quad = \quad & \sum u_i \times Ben(R_i) + \sum v_j \times Ben(C_j) \\ & + \sum_{non-blue\ \ d_{ij}} u_i \times v_j - \sum_{blue\ \ d_{ij}} u_i \times v_j (4) \end{aligned}$$

Then, the 2-d MDLH region finding problem can be defined as:

$$Minimize \quad f = -TotBen \quad s.t. \quad 0 \leq u_i, v_j \leq 1$$

The negative sign is to reflect that the intention of maximizing total benefits. Note that u_i, v_j's are not integer variables, as that would make computation hard. Instead, they are treated as real variables. If the optimal quadratic programming solution is not integral, then rounding is used to give an approximate solution.

Figure 9 shows a simple 2-d example. The benefits of rows and columns are shown below:

regions	variables	benefits
$(a, 5)$	u_1	-1
$(b, 5)$	u_2	-1
$(c, 5)$	u_3	1
$(d, 1)$	v_1	2
$(d, 2)$	v_2	-2
$(d, 3)$	v_3	0
$(d, 4)$	v_4	-2

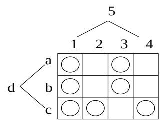


Figure 9: Example for Quadratic Programming

```
Algorithm MDLH-Quad
Input: a k-d data cube and the set of blue cells D
Output: an MDLH description for D
  1. Follow steps 1 and 2 of Algorithm MDLH-Dynamic.
  2. For any data region $x = (x_1, x_2, \ldots, x_k)$:
     (a) If there are exactly two non-leaf nodes $x_i, x_j$ ($1 \le i, j \le k$),
         i. Set up the 2-d quadratic programming objective function as in Equation 4.
         ii. Solve the quadratic programming problem.
         iii. The solution is stored externally. Set $Sel(x_1, \ldots, x_k)$ to point to this solution.
     (b) Otherwise, just follow step 3 of Algorithm MDLH-Dynamic.
```

Figure 10: A Skeleton Algorithm for MDLH-Quad

Then the objective function is $f = -TotBen$, where $TotBen$ is: $(-u_1 - u_2 + u_3) + (2v_1 - 2v_2 - 2v_4) - u_1v_1 + u_1v_2 - u_1v_3 + u_1v_4 - u_2v_1 + u_2v_2 - u_2v_3 + u_2v_4 - u_3v_1 - u_3v_2 + u_3v_3 - u_3v_4$.

The optimal solution here for the quadratic programming problem is: $u_3 = v_1 = v_3 = 1$, with the remaining variables set to 0. This happens to be the true optimal optimal solution: $\{(c, 5), (d, 1), (d, 3)\} \ominus \{(c, 3)\}$. For this example, both MDLH-Greedy and MDLH-Dynamic compute sub-optimal solutions.

There are two key considerations in applying a quadratic programming approach. First, the complexity is high [7]. Let there be f_1 rows and f_2 columns. There is a polynomial time algorithm using the ellipsoid method and in the worst case performs $O((f_1+f_2)^6)$ arithmetic operations. Second, the polynomial time algorithm only guarantees an optimal solution if the objective function is convex. For our MDLH region finding problem, however, it is not convex. Thus, a quadratic programming approach is only a heuristic, and given its expensive nature, it should be applied cautiously. The compromise we strike is to apply the quadratic programming approach to regions $(x_1, x_2, \ldots, x_k)$ only if there are exactly two non-leaf nodes x_i, x_j. For regions where there are more two non-leaf nodes, we resort back to dynamic programming. This is summarized in Figure 10 which gives a skeleton of the algorithm called MDLH-Quad.

The algorithm is identical to MDLH-Dynamic (i.e., Figure 7), except in two areas. First, if there are exactly two non-leaf nodes for the region, then the MDLH problem is solved as a 2-d quadratic programming problem. In our implementation, we call the Matlab function *quadprog* to find a solution. Second, because this solution can include a combination of rows and columns, it is not sufficient to store the solution in the Sel matrix as in Figure 7. Instead, a bit vector is used to record which rows and columns are selected. The Sel entry then points to the bit vector.

This concludes our discussion on heuristic algorithms for solving the MDLH region finding problem. In the next section, we compare these three heuristics empirically, with a specific focus on the tradeoffs between length and computational efficiency.

4 Experimental Evaluation

4.1 Experimental setup

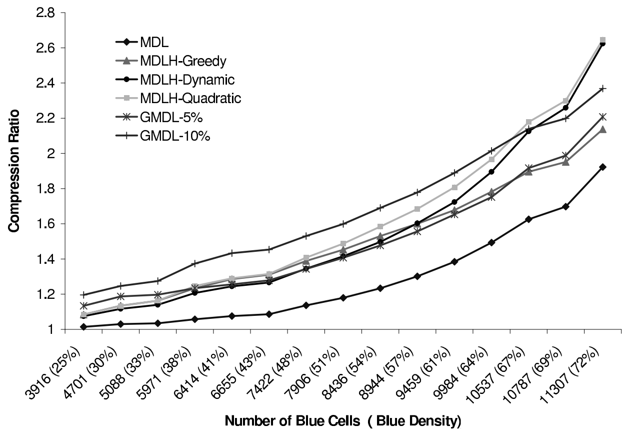
Dataset: Our experiments are based on the TPC-H benchmark. We use the TPC-H database generator to populate the datasets. There are eight tables and the `lineitem` table contains 6 million records. From those tables, we build a default data cube consisting of 3 dimensions: `customers`, `suppliers` and `parts`. Each dimension is a 3-level hierarchy with the fanout of 1-5-5. On the `customers` dimension, there are 5 `geographic regions`; and each `geographic region` contains 5 `countries`. The same hierarchy is used for `suppliers`. On the `parts` dimension, there are 5 `manufacturers`, and each `manufacturer` produces 5 `brands`.

Given the fanout of 1-5-5, the total number of data cells is $25 \times 25 \times 25 = 15,625$ data cells. Each data cell is of the form: $(customerCountry, supplierCountry, partBrand)$. Each data cell records the `quantity` of `partBrand` that customers from `customerCountry` bought from `supplierCountry`. By changing the threshold on quantity, we generate sets of blue cells of different cardinalities.

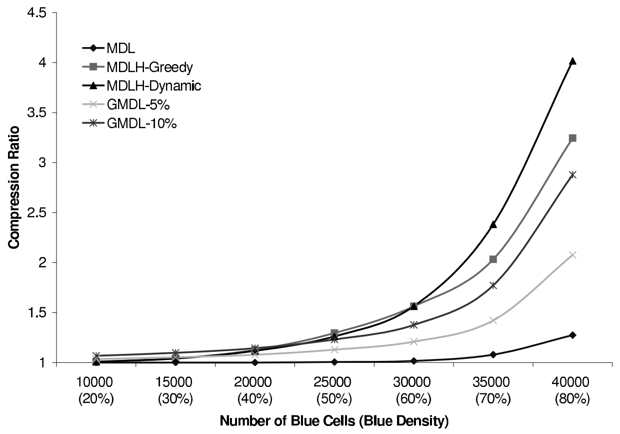
Algorithms: We implemented the three heuristic algorithms MDLH-Greedy, MDLH-Dynamic and MDLH-Quad, all in C++. To compare with other alternatives, we use MDL-Tree and GMDL-Tree developed in [8]. The former produces MDL summaries containing regions consisting of blue cells only; the latter generates GMDL summaries which can contain up to a specified number of white cells. Recall that under the GMDL framework, apart from the blue cells, there are also white cells and red cells, which must be excluded in any region.

Metrics: One obvious way to measure the effectiveness of each algorithm is the length of the summary it produces. For MDLH, the length is $|S| + |H|$. For MDL and GMDL, the length is the total number of regions (or cells). In the case of GMDL, the length does not include the number of white cells. Thus, to some extent, the GMDL framework has an unfair advantage. We use the relative quantity called *compression gain ratio*, which is the ratio of the number of blue cells to the length of the MDL, MDLH and GMDL descriptions. All the algorithms here produce a compression gain ratio ≥ 1. Another way to measure the effectiveness of the algorithms is the average size of each true region (i.e., a region consisting of two or more cells).

The key question we would like to answer is whether



(a) Three 3-level hierarchies



(b) One 4-level and Two 3-level Hierarchies

Figure 11: Compression Ratio: MDLH Against Other Alternatives

MDLH produces any *additional* compression gain on top of MDL. The comparison with GMDL is less straightforward because the GMDL summaries contains non-blue cells, the total number of which is not included in the length.

The gain in compression must be evaluated simultaneously with the time taken to produce the summaries. We use absolute runtime figures based on a Sun Fire 880, 900MHz UltraSPARC-III machine.

4.2 TPC-H: 3-d, 3-level Cube

Figure 11(a) compares the compression gain ratios of the various algorithms. The x-axis shows the total number of blue cells, obtained by varying the threshold of the `quantity` value. The number varies from 3916 to 11,307, corresponding to 25% and 72% respectively (as there are 15,625 cells in total). Hereafter, we refer to this percentage as the *blue density.* The y-axis shows compression gain.

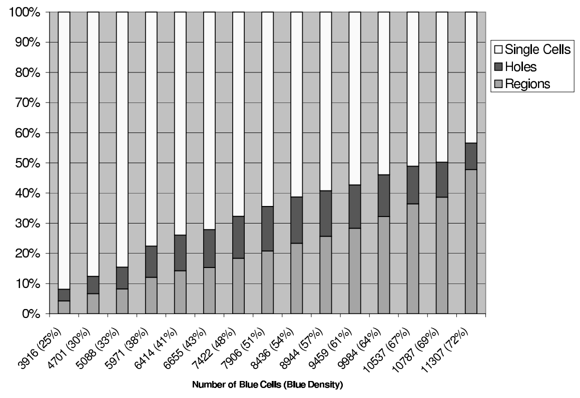
As expected, the worst performer is MDL. The gaps between MDL and the MDLH algorithms are distinct. Furthermore, the gaps widen as the number of blue cells increases. For GMDL, two curves are included. One has a white budget of 5% of the number of blue cells, and the other has 10%. Clearly, the higher the white budget, the higher the compression gain. Yet, the more impure the description becomes, as it contains more and more unidentified non-blue cells. In comparing GMDL with MDLH, we make two observations:

- Just from Figure 11(a), initially GMDL(10%) performs better than MDLH-Quad and MDLH-Dynamic. But as the number of blue cells increases, the latter two eventually outpace GMDL(10%).
- For the GMDL algorithms, the white budget is "free" as the budget is not included in the length of the description. One way to level the playing field is to include the white budget in the length, as if the white cells were identified explicitly as holes like in MDLH. Take the case with 10,537 blue cells as an example. The compression gain of GMDL(10%) is around 2, meaning that the length of the summary it produces is around 10537/2 = 5269. If we add back the number of white cells, the total length of the GMDL summary can be viewed as 5269 + (10% * 10537) = 6323. This corresponds to a compression gain of about 1.67, definitely lower than those produced by the MDLH algorithms including MDLH-Greedy.

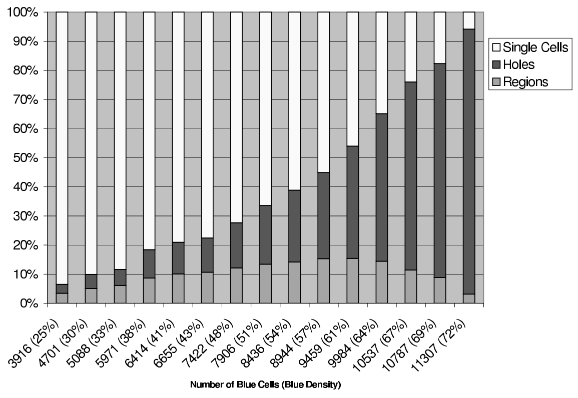
The results for GMDL are based on red cells corresponding to 5% of the number of blue cells. In other words, if there are 10,537 blue cells, there are about 515 red cells. In practice, the number of red cells can be higher, in which case the GMDL performance worsens.

Among the MDLH algorithms, the best performer is MDLH-Quad, with MDLH-Dynamic a close second. One way to explain the superiority of these two algorithms over MDLH-Greedy is to breakdown the summary $S \ominus H$. Figure 12 shows the percentages of true regions (i.e., regions consisting of two or more cells), holes and single blue cells. Figure 12(a) shows that as the number of blue cells increases, MDLH-Greedy finds more and more true regions, with fewer and fewer single blue cells. It is interesting to see that the percentage of holes remains rather steady. In contrast, for MDLH-Dynamic in Figure 12(b), as the number of blue cells increases, the percentage of holes grows at the expense of single blue cells. Furthermore, the percentage of true regions grows and then *shrinks.* From the point of view of compression gain, this could be *good news.* The reason is that a reduction in true regions may indicate that a grandparent region has replaced several parent regions in the summary.

To verify this phenomenon, we measure the average



(a) MDLH-Greedy



(b) MDLH-Dynamic

Figure 12: Percentage Composition of $S \ominus H$

size of a true region, expressed in the number of data cells it covers. The following table shows the snapshot when there are 10,537 blue cells (67% density).

MDL	GMDL (10%)	MDLH-Greedy	MDLH-Dynamic	MDLH-Quad
4	3.1	3.8	16.5	9.8

Recall that the fanout of the TPC-H hierarchies is 1-5-5. Thus, an average size exceeding 5 indicates that there are definitely some grandparent regions included in the summary. On this note, both MDLH-Dynamic and MDLH-Quad fare very well. To relate back to Figure 12(b), MDLH-Dynamic is getting to the grandparent regions by including a fair number of holes. In other words, MDLH-Dynamic is putting the holes to good use. Figure 12 does not show the profile for MDLH-Quad as it is very similar to that of MDLH-Dynamic.

For some applications where approximate query answering is acceptable, it may be sufficient in many situations for the user to see just the S part (with the H part only shown in "drill-down" mode). Take the case with 10,537 blue cells as an example. From Figure 11(a), the compression gain is about 2, meaning that the total length of MDLH summary is about 5,269. Then from Figure 12(b), the number of true regions is only about 11% of the summary, corresponding to about 550 regions. There are about double the number of single cells. Thus, the user may be satisfied with seeing just the 550 true regions and the 1,100 single cells; the remaining list of holes can be shown only if needed. This is a significant improvement over showing the user all the 10,537 blue cells.

4.3 Increasing the Sizes of Hierarchies

The TPC-H cube is small because the hierarchies are small and short. Thus, in this experiment, we increase the size of the two 3-level hierarchies T_1, T_2 by increasing the fanout. Instead of 1-5-5, we augment the fanout to 1-5-10; for example, there are now 10 `countries` under each `geographic region`. Furthermore, we increase the number of levels of the third hierarchy T_3 from 3 to 4; for example, there are now 20 *models* under each `brand` in the `manufacturer` hierarchy. The fanout of this hierarchy is 1-5-5-20. In all, the 3-d cube grows from 15,625 cells to 1.25 million cells. Instead of randomly generating `quantity` values to fill the cells, we directly generate the blue cells. Specifically, we fix a clustered region of $10 \times 10 \times 500 = 5 \times 10^4$ cells (i.e., 10 leaves in T_1 and T_2 and 500 leaves in T_3). Within this clustered region, blue cells are randomly generated. The number varies from 10,000 to 40,000. Thus, the blue density within the clustered region varies from 20% to 80%.

Figure 11(b) compares the compression gain ratios of the various algorithms. The x-axis and y-axis are almost identical to those in Figure 11(a), except that the range of the y-axis is larger. Compared with the smaller cube in (a), the cube here has a larger fanout. In general, a larger fanout allows the parent region to have a higher potential benefit. This gives more opportunity for MDLH to excel. Consequently, the gaps between MDL, MDLH-Dynamic and MDLH-Greedy widen significantly. Similarly, the gaps between GMDL(10%), MDLH-Dynamic and MDLH-Greedy widen as well. The figure no longer includes MDLH-Quad because applying it to larger cube takes much longer.

4.4 Scalability with Dimensions

So far, we have only experimented with 3 dimensions. In this experiment, we further increase the size of the cube by adding two more dimensions. Following the previous experiment, we use four 3-level hierarchies $T_1, \ldots, T_4$ with the fanout 1-5-10 and one 4-level hierarchy T_5 with the fanout 1-5-5-20. Blue cells are randomly picked from a clustered region of about 1 million cells (i.e., 10 leaves in T_1 to T_4 and 100 leaves in T_5). The number of blue cells varies from 200,000 to 800,000, a significant increase in the number from

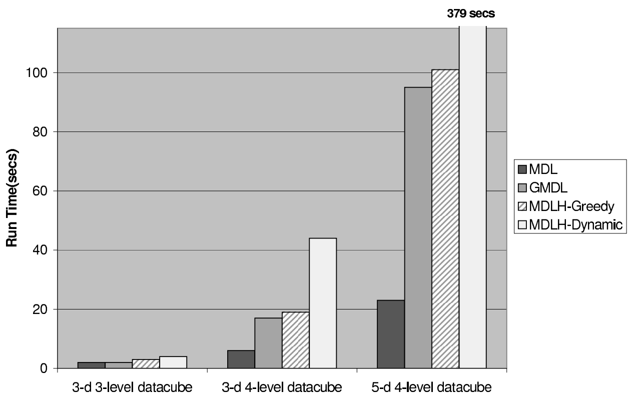


Figure 13: Runtimes on the Three Cubes

the previous cube.

Again, we measure the compression gain ratio of the various algorithm. The graph turns out to be very similar to Figure 11(b). Thus, to save space, we do not include this graph. Instead, we show in Figure 13, the absolute runtimes of the algorithms on the various cubes used so far. The blue density is kept at 70%. In fact, for the first two cubes, the runtime is rather insensitive to the blue density. On first glance, MDL takes the least time; GMDL and MDLH-Greedy take roughly double the time MDL takes; and MDLH-Dynamic may take considerably longer.

However, let us take a closer look at Figure 13 to evaluate scalability. Recall that the numbers of blue cells for the 3 cubes at 70% blue density level correspond to about 11,000, 35,000 and 700,000 respectively. Thus, while the runtime of MDLH-Dynamic appears to grow significantly from the second cube to the third cube, the proportional growth in runtime relative to the number of blue cells is in fact sub-linear. The same is true for MDLH-Greedy.

4.5 Summary: Recommendations

Among the three heuristics, MDLH-Quad, as expected, generates the shortest summary. However, it is not scalable and is not recommended for use. The only reason why we include the algorithm in this paper is to use it as a yardstick of quality. From this standpoint, we observe that MDLH-Dynamic comes very close in quality to MDLH-Quad. In terms of runtime, MDLH-Dynamic is scalable with respect to the number of blue cells. Even for 700,000 blue cells, MDLH-Dynamic offers acceptable performance. But if a user wants a shorter response time, MDLH-Greedy is recommended. As shown in Figure 11(b) and Figure 13, it can keep a fine balance between compression gain and runtime.

As compared with MDL, it is clear that MDLH-Dynamic and MDLH-Greedy bring additional compression. As discussed earlier, these two algorithms offer significant advantages over the GMDL method. As shown in Figure 11(b), the larger the fanout, the more opportunities there are for MDLH to operate. Thus, even for hierarchies with medium fanout, the MDLH algorithms are the recommended methods.

5 Extension: Summarizing the Holes

5.1 Going One Step Further

Notice from Figure 12(b) that as the blue density becomes reasonably high, a large part of the MDLH description is made up of holes. Recall that in an MDLH summary $S \ominus H$, H is a set of non-blue cells. Just as we try to summarize a set D of blue cells with an MDLH description, we can imagine summarizing H with an MDLH description. The motivation is the same, namely to reduce the total length of the description. More formally, we have described D as $S \ominus H$, and we now try to describe H as $S_H \ominus H_H$. To put the two parts back together, we can then describe D as $S \ominus S_H + H_H$. Notice that the '+' sign is to be interpreted as a set union.

Figure 14 gives an example, which is a slight modification of the example in Figure 6. Node 10 in Figure 6 is removed, creating an unbalanced hierarchy. For the set of blue cells given in Figure 14, the MDLH description is $\{(a,12),(d,12),(b,6),(c,8)\}\ominus \{(a,6),(a,8),(a,9),(d,6),(d,7),(d,8)\}$. The length is 10. However, if we apply MDLH to summarize the 6 holes, they can in turn be represented as $\{(a,11),(d,11)\} \ominus \{(a,7),(d,9)\}$. Thus, the total length is reduced by 2.

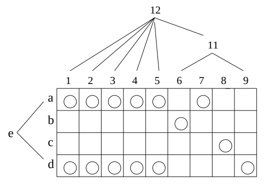


Figure 14: Example of Nested Holes

Figure 15 gives a skeleton of how this can be done. It amounts to simply instructing the various MDLH algorithms to summarize a given set of non-blue cells. No change is needed for any of the algorithms. In theory, we can repeat the same exercise on H_H to try to summarize some of its cells into regions. We do not recommend doing so because we feel that multiple-level of nested negation can be hard for the user to relate to.

5.2 Experimental Evaluation

Recall that when we first introduce the MDLH region finding problem, we make the simplifying assumption that all the leaf nodes appear in the same level. This assumption of a balanced hierarchy is not necessarily true for many datasets. In fact, as shown in the previous example, unbalanced hierarchies can be amenable to hole summarization.

```
Algorithm SumHoles
Input: a k-d data cube and the set of blue cells D
Output: an MDLH description for D

1. Apply MDLH-Dynamic or MDLH-Greedy on $D$ and get the output $S \ominus H$.
2. Feed $H$ as input back into MDLH-Dynamic or MDLH-Greedy to summarize the holes. The output is $S_H \ominus H_H$.
3. Return $S \ominus S_H + H_H$ as the description for $D$.
```

Figure 15: A Skeleton Algo. for Hole Summarization

Here we experiment with unbalanced hierarchies, modified from the TPC-H setup. For every parent node x, we change its first leaf child c_1 to be an internal node. For the first two hierarchies T_1, T_2, which have the original fanout 1-5-10, we add 5 leaf children to c_1. Thus, there are 70 leaves in each of T_1 and T_2. For T_3, which has the original fanout 1-5-5-20, we add 10 leaf children to c_1, creating a total of 725 leaves. Therefore this cube contains $70 \times 70 \times 725 \approx 3.5M$ data cells. Blue cells are randomly picked from a clustered region of about 140,000 cells (i.e., 14 leaves in T_1, T_2 and 725 leaves in T_3). The following table shows the size of the MDLH descriptions generated by MDLH-Dynamic when the blue density is 70% corresponding to about 100,000 blue cells.

	$\lvert S\rvert$	$\lvert H\rvert$	$\lvert S_H\rvert$	$\lvert H_H\rvert$	Total
Before	100	25365			25465
After	100		7227	8006	15327

Before summarization is applied to the holes, the compression gain is about 100000/25465 = 4. After the extra step, the compression gain improves to 6.5. This shows that summarizing holes can lead to considerable additional gain in compression.

6 Conclusions

In this paper, we study the problem of finding optimal MDLH summaries of the form $S \ominus H$ for k-d cubes with tree hierarchies. By a slight twist, the summaries can be further generalized to the form of $S \ominus S_H + H_H$. We show that finding the optimal solution is NP-hard, even for 2-d, 2-level cubes. We propose three heuristics algorithms to find MDLH regions. Experimental results show that MDLH-Dynamic gives very high quality summaries, almost as good as the ones produced by MDLH-Quad. MDLH-Dynamic is scalable to higher dimensional and larger problems. However, if a lighter weight computation is needed for quick exploration, MDLH-Greedy can run considerably faster than MDLH-Dynamic, while delivering compact summaries.

The results here confirms the philosophy of summarization with holes. Compared with MDL, MDLH gives much shorter summaries. Compared with GMDL, MDLH does not need a white budget and explicitly identifies all the impure, non-blue cells in the summary. This makes the MDLH framework simpler for the user to use. Furthermore, experimental results confirm that MDLH summaries, even with the number of holes included, can be significantly shorter than GMDL summaries.

References

[1] G. Ausiello et al. Complexity and Approximations: Combinatorial Optimization Problems and Their Approximability Properties. Springer 1999.

[2] J. Boot. Quadratic Programming: Algorithms, Anomalies, Applications. North-Holland Publishing Company, 1964.

[3] S. Bu. The Summarization of Hierarchical Data with Exceptions. Master Thesis, UBC, 2004.

[4] J. Gray et al. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. Journal of Data Mining and Knowledge Discovery, 1, 1, pp 29-53, 1997.

[5] V. Harinarayan et al. Implementing Data Cubes Efficiently. SIGMOD 96, pp 205-216.

[6] D.Hochbaum. Approximating Clique and Biclique Problems. Journal of Algorithms, 29, 1, pp174-220, 1998.

[7] M. Kozlov et al. Polynomial solvability of convex quadratic programming. Soviet Mathematics Doklady. 20, pp 1108–1111,1979.

[8] L. Lakshmanan et al. The Generalized MDL Approach for Summarization, VLDB 02, pp 766-777.

[9] L. Lakshmanan, J. Pei and J. Han. Quotient Cube: How to Summarize the Semantics of a Data Cube. VLDB 02, pp 778-789.

[10] A. Mendelzon and K.Pu. Concise descriptions of subsets of structured sets. ACM PODS 03, pp 123-133.

[11] J. Rissanen. Modelling by shortest data description. Automatica, volumne 14, pp 465–471, 1978.

[12] S. Sarawagi. Explaining differences in multidimensional aggregates. VLDB 99, pp 42-53.

[13] S. Sarawagi. User-adaptive exploration of multidimensional data. VLDB 00, pp 307-316.

[14] G. Sathe and S. Sarawagi. Intelligent Rollups in Multidimensional OLAP Data. VLDB 01, pp 531-540.

[15] Y. Sismanis et al. Dwarf: Shrinking the petaCube. SIGMOD 02, pp 464-475.

[16] J. S. Vitter and M. Wang. Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets. SIGMOD 99, pp 193-204.

[17] Y. Zhao et al. An Array-Based Algorithm for Simultaneous Multidimensional Aggregates. SIGMOD 97, pp 159-170.