

Parallel Incremental 2D-Discretization on Dynamic Datasets

Srinivasan Parthasarathy* and Arun Ramakrishnan

Computer and Information Science

Ohio State University,

Columbus, OH 43235

Contact: srini@cis.ohio-state.edu

Abstract

Most current work in data mining assumes that the database is static, and a database update requires re-discovering all the patterns by scanning the entire old and new database. Such approaches can waste a lot of computational and I/O resources, and result in relatively slow response times, to essentially an interactive process. In this paper we address this issue in the context of 2-dimensional discretization within a multi-attribute database. Discretization, an important problem in data mining, is typically used to partition the range of continuous attribute(s) into intervals which highlight the behavior of a related discrete attribute. It can be used to build decision trees and to determine appropriate aggregations for On-Line Analytical Processing. We first propose a time-optimal solution to the problem. We then parallelize and incrementalize the algorithm so that it can dynamically maintain the required information even in the presence of data updates without re-executing the algorithm on the entire dataset. Experimental results confirm that our approach results in execution time improvements of up to several orders of magnitude on large datasets.

1 Introduction

The field of knowledge discovery and data mining (KDD), spurred by advances in data collection technology, is concerned with the process of deriving interesting and useful patterns from large datasets. The KDD process is a compute and data-intensive process and is inherently interactive and iterative in nature. In fact, interactivity is often the key to facilitating effective data understanding and knowledge discovery. In such an environment response time is crucial because lengthy time delay between responses of two consecutive user requests can disturb the flow of human perception and formation of insight. The task of guaranteeing quick response times is more complicated in dynamic datasets, where there is a constant influx of data. Changes to the data can invalidate existing

patterns or introduce new ones. Simply re-executing algorithms from scratch on a database update can result in an explosion in the computational and I/O resources required. What is needed is a way to incrementally process the data. In this article we present such an approach for a key data-mining task: discretization.

Discretization has typically been thought of as the partitioning of the range of a continuous (base) attribute into intervals, in order to highlight the behavior of a related discrete (goal) attribute. It has been used for classification in the decision tree context, as well as for summarization in situations where one needs to transform a continuous attribute into a discrete one with minimum “loss”. While typically discretization methods have focused on discretizing a single continuous attribute, in past work[12] we have noted that while such methods can provide optimal discretizations along one dimension they are not capable of generating optimal discretizations for the multi-dimensional case where there are several dependent attributes that need to be classified into regions. To gain an intuitive insight into why this is so, consider the classic “xor” example. Let the distribution of a class (1:value of a goal attribute) be characterized by 2 normals with means at opposite corners of the unit square, representing the X-Y (the two base attributes being discretized) plane, say $((0, 0), (1, 1))$. Let the distribution of another class (2) be characterized by 2 normals with means at the other corners of the unit square, i.e., $((1, 0), (0, 1))$. Viewing the joint distribution when projected onto a single dimension blurs the obvious separation that exists. A more general version of the problem in the multi-attribute database case is to identify the two attributes that result in the best discretization. This problem requires each pairwise attributes to be evaluated against an optimization function. In this paper we consider an parallel incremental solution to the the above general problem. Our key contributions are:

- A time-optimal exact solution to the 2D discretization problem.
- A fast parallel incremental implementation for the

*This work supported in part by an Ameritech Faculty Fellowship.

general multi-attribute 2D Discretization problem.

- Empirical evaluations of the algorithm on a large multi-attribute dynamic dataset.

The rest of this paper is organized as follows. In the next section we present our problem statement and related definitions. Then in Section 3 we present the time-optimal exact solution to solve the 2D discretization problem along with a proof of optimality. In Section 4 we present the parallel incremental implementation for the general multi-attribute 2D Discretization problem. We present empirical evidence to validate the claims made in this paper in Section 5. Section 6 surveys some of the literature relevant to the work presented in this paper. Finally in Section 7 we conclude with directions for future work.

2 Problem Definition

Let X and Y be two continuous valued attributes, which we model as random variables. Let G be the goal attribute, over which the two base attributes (X and Y) are to be discretized. The goal attribute could be either continuous or discrete valued. For simplicity, we assume that G is discrete and can take on a constant number of values (C_g), i.e., $G \in G_v = \{g_1, g_2, \dots, g_{C_g}\}$.

We assume that the relations among these three attributes are given as a pdf (probability distribution function) $p(x, y, g)$, i.e., $p(x, y, g) \geq 0$ for all $(x, y, g) \in X \times Y \times G$ and $\sum_{x, y, g} p(x, y, g) = 1$. We shall assume that we are given the joint pdf $p(x, y, g)$. In practice, this pdf is estimated at discrete locations, $X \times Y \times G_v$, where $X = \{x_1, x_2, \dots, x_n\}$ and $Y = \{y_1, y_2, \dots, y_n\}$. The locations are chosen so as to be equi-probable rather than equi-spaced so as to provide finer resolution in denser regions. Essentially, the values of X (and similarly for Y) are chosen so that $\int_{x=x_i}^{x=x_{i+1}} p(x) \approx \frac{1}{n}$.

We will partition the XY plane (the pdf) in terms of ‘‘control points’’, or cut-points. Choosing one control point (cut-point) (an x_i, y_j value) splits the plane into 4 regions, or intervals, as we would have called them in the 1D case. For simplicity we assume that the X-Y plane is bounded by the half infinite interval (non-negative values only). Note that this is not a restrictive assumption and that all the results in this chapter can be extended to the more general case.

Definition 1 A point $o = (x_o, y_o)$, where $x_o, y_o \geq 0$ partitions the (x, y) plane into 4 regions $\mathcal{R}(o) = \{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4\}$ where $\mathcal{R}_1 = (0 \leq x \leq x_o, 0 \leq y \leq y_o)$ $\mathcal{R}_2 = (0 \leq x \leq x_o, y > y_o)$ $\mathcal{R}_3 = (x > x_o, 0 \leq y \leq y_o)$ $\mathcal{R}_4 = (x > x_o, y > y_o)$. Note that $\{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4\}$ are simple rectangular regions.

We define the following notation. The net probability at a discrete location is defined as:

$$p(x, y) = \sum_g p(x, y, g) \quad (1)$$

The total probability weight of a region \mathcal{R} is defined as:

$$w(\mathcal{R}) = \sum_g \sum_{x, y \in \mathcal{R}} p(x, y, g) \quad (2)$$

The total probability weight of goal attribute g in a region \mathcal{R} is defined as:

$$p(g, \mathcal{R}) = w_g(\mathcal{R}) = \sum_{x, y \in \mathcal{R}} p(x, y, g) \quad (3)$$

Define:

$$p(g|\mathcal{R}) = \frac{w_g(\mathcal{R})}{w(\mathcal{R})} \quad (4)$$

Given a point o and a region $\mathcal{R} \in \mathcal{R}(o)$ one can define the average pdf of the region $q(x, y, g)(\mathcal{R})$ as one in which the relative probabilities of the goal outcomes is the same for all points in the region and is equal to the average relative probabilities. We make this precise below in Equation 5

$$q(x, y, g)(\mathcal{R}) = p(x, y) \times p[g|\mathcal{R}], \quad x, y \in \mathcal{R} \quad (5)$$

If the approximation introduced by discretization is to cause as little information loss as possible, then p and q should be as similar as possible. The Kulback-Leibler Distance (KL-Distance) [3], $D(p||q)$, is a metric that can be used to measure the distance between the two pdfs, p and q , which share the same range.

$$D(\mathcal{R}) = D(p||q) = \sum_{g \in G} \sum_{x, y \in \mathcal{R}} p(x, y, g) \log \frac{p(x, y, g)}{q(x, y, g)(\mathcal{R})} \quad (6)$$

We can then state our objective function as:

$$D(o) = \sum_{\mathcal{R} \in \mathcal{R}(o)} D(p(\mathcal{R})||q(\mathcal{R})) \quad (7)$$

We are now in a position to formally state the 2D Discretization problem in terms of the objective function $D(o)$.

Problem Definition 2.1 Given a pdf $p(x, y, g)$, find a point $o = (x, y)$ such that $D(o)$ is minimized.

An alternative intuition to capture the optimal discretization is that it should minimize classification error. This is possible when the concept of error can be meaningfully defined. This is usually possible if G is a discrete attribute and C_g is small. Let \hat{g} be the most likely outcome of the goal attribute in a region, i.e., $\sum_x \sum_y p(x, y, \hat{g}) \geq$

$\sum_x \sum_y p(x, y, g) \forall g \neq \hat{g}$. Abbreviating classification error as CE we have,

$$CE(\mathcal{R}) = \sum_{g \in G, g \neq \hat{g}} \sum_x \sum_y p(x, y, g) = w(\mathcal{R}) - w_{\hat{g}}(\mathcal{R}) \quad (8)$$

$$CE(o) = \sum_{\mathcal{R} \in \mathcal{R}(o)} CE(\mathcal{R}) \quad (9)$$

Now the alternative definition in terms of $CE(o)$ is:

Problem Definition 2.2 Given a pdf $p(x, y, g)$, find a point $o = (x, y)$ such that $CE(o)$ is minimized.

The more general multi-attribute 2D discretization problem can be stated as:

Problem Definition 2.3 Given a database with several base attributes $a_1 \dots a_n$ find the pair of attributes a_X and a_Y for which the objective function (CE or D) is optimal.

This requires one to solve the 2D discretization problem for each pair of attributes and identify the pair of attributes for which the objective function is optimal. Although several useful extensions to the above problem definitions are possible, these are not discussed here due to space constraints (see [10] for details).

3 Algorithms

In this section we describe the basic algorithms to solve the problems defined in the previous section. In the simplest case, the solution to these problems simply boils down to a brute force search of all possible control points and identifying the control point(s) that optimizes a given objective function. This naive approach is presented in Section 3.1. By re-using previously computed results and organizing the search in a particular manner we show how one can solve this problem optimally (complexity-wise and result-wise) in Section 3.2. Note that even in the refined method we essentially conduct a brute force search of all possible control points as defined by the pdf grid. However, in prior work we have shown that going to very fine granularity grids does not buy you much in terms of accuracy for many real applications. In fact we showed that for all practical purposes a pdf grid size of 64X64 ($n=64$) or 128X128 ($n=128$) is ample. As we shall see from the experimental results, even the naive brute force search of such (relatively) small grids can be expensive so our new re-use-based algorithm is extremely effective.

Below, we first present some notation and some basic lemmas that we use throughout this section. We noted in the previous section that a single control point divides the XY plane into four rectangular regions. We define evaluating one of these rectangular regions as follows.

Notation 3.1 Evaluation of a rectangle, \mathcal{R} , comprises calculating $w(\mathcal{R}), D(\mathcal{R}), p(g|\mathcal{R})$ for Problem 2.1 and $w(\mathcal{R}), w_g(\mathcal{R}), \min(CE(\mathcal{R}))$ for Problem 2.2.

The cost of evaluating these rectangles is a function of the total area it covers since each point in the rectangular region affects the values of $w(\mathcal{R}), D(\mathcal{R}), p(g|\mathcal{R}), w_g(\mathcal{R})$, and $\min(CE(\mathcal{R}))$. This can be formally stated as:

Lemma 1 The cost of evaluating a rectangle is proportional to its area and the number of goal classes (C_g). If the number of goal classes is fixed to a constant, the cost of evaluating rectangle, $[(x_i, y_j), (x_k, y_l)]$, is simply $O((x_i - x_k)(y_j - y_l))$.

As previously stated evaluating a cut-point amounts to evaluating the four rectangular regions it induces. This can be formally stated as:

Lemma 2 A cut-point $C(x_i, y_j)$ requires evaluating rectangles $[(x_1, y_1), (x_i, y_j)], [(x_{i+1}, y_{j+1}), (x_n, y_n)], [(x_1, y_{j+1}), (x_i, y_n)], [(x_{i+1}, y_1), (x_n, y_j)]$

We now describe our naive algorithm to solve the 2D-Discretization problem.

3.1 Naive Algorithm

A straight-forward approach to finding the optimal discretization is to consider all possible locations of the control point(s). This leads to the following theorem.

Theorem 1 A 2-dimensional discretization with one control point (Problems 2.1 and 2.2) can be solved in $O(n^4)$ time.

Proof : Since there are n^2 possible control points (Lemma 1), and the cost of evaluating each control point is $O(n^2)$ (Lemma 2), the statement holds. ■

However, by reusing previous computations, we can reduce the cost to $O(n^2)$. What is remarkable about this result is that the cost of evaluating a single control point is asymptotically the same as evaluating all $O(n^2)$ control points. Lemma 3 specifies a limited type of rectangle additions/subtractions that we use to re-use previous computations. These operations are such that the resultant region is also a rectangle and can be computed in $O(1)$ time.

Lemma 3 (Addition and Subtraction of Rectangles)

For all i, j, k, l , and m where $1 \leq i \leq k \leq m \leq n$ and $1 \leq j \leq l \leq n$

$$[(x_i, y_j), (x_k, y_l)] + [(x_{k+1}, y_j), (x_m, y_l)] = [(x_i, y_j), (x_m, y_l)]$$

$$[(x_i, y_j), (x_m, y_l)] - [(x_{k+1}, y_j), (x_m, y_l)] = [(x_i, y_j), (x_k, y_l)]$$

Lemma 4 Let \mathcal{R}_1 and \mathcal{R}_2 be two disjoint regions. Let $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$. Given $D(\mathcal{R}_1), D(\mathcal{R}_2), w(\mathcal{R}_1), w(\mathcal{R}_2), p(g|\mathcal{R}_1)$ and $p(g|\mathcal{R}_2)$, one can compute $D(\mathcal{R}), p(g|\mathcal{R})$, and $w(\mathcal{R})$ in $O(1)$ time.

Proof : $w(\mathcal{R}) = \sum_g \sum_{x,y \in \mathcal{R}} p(x,y,g) = \sum_g (\sum_{x,y \in \mathcal{R}_1} p(x,y,g) + \sum_{x,y \in \mathcal{R}_2} p(x,y,g))$ as long as \mathcal{R}_1 and \mathcal{R}_2 are disjoint. Hence, $w(\mathcal{R}) = w(\mathcal{R}_1) + w(\mathcal{R}_2)$ and can be computed in $O(1)$ time.

Equation 10 is simple to derive and shows that $p(g|\mathcal{R})$ can be computed in $O(1)$ time. Obviously

$$p(g|\mathcal{R}) = \frac{p(g|\mathcal{R}_1) \times w(\mathcal{R}_1) + p(g|\mathcal{R}_2) \times w(\mathcal{R}_2)}{w(\mathcal{R}_1) + w(\mathcal{R}_2)} \quad (10)$$

and this can be computed in $O(1)$ time. We shall use b as short hand for x,y . Since $p(b,g) = p(g|b)p(b)$ by Bayes Law and $q(b,g) = p(b)p(g|R)$ by Equation 5, $D(\mathcal{R})$ can be re-written as $\sum_g \sum_{b \in \mathcal{R}} p(b,g) \log \frac{p(g|b)}{p(g|\mathcal{R})}$. This can be re-written as $D_1(\mathcal{R}) - D_2(\mathcal{R})$ where

$$D_1(\mathcal{R}) = \sum_g \sum_{b \in \mathcal{R}} p(b,g) \log p(g|b)$$

$$\text{and } D_2(\mathcal{R}) = \sum_g \sum_{b \in \mathcal{R}} p(b,g) \log p(g|\mathcal{R})$$

$$= \sum_g \log p(g|\mathcal{R}) \sum_{b \in \mathcal{R}} p(b,g)$$

$$= \sum_g \log p(g|\mathcal{R}) w_g(\mathcal{R}) = \sum_g \log p(g|\mathcal{R}) p(g|\mathcal{R}) w(\mathcal{R})$$

by Equation 4. Now, using Equation 10, $D_2(\mathcal{R})$ can be computed in $O(1)$ time, since C_G is a fixed constant.

Note that $D_1(\mathcal{R}) = D_1(\mathcal{R}_1) + D_1(\mathcal{R}_2)$. We can also re-write $D_1(\mathcal{R}_1)$ as $D(\mathcal{R}_1) + D_2(\mathcal{R}_1)$, and $D_1(\mathcal{R}_2)$ as $D(\mathcal{R}_2) + D_2(\mathcal{R}_2)$ from the above representations of D_1, D_2 and D . Therefore $D_1(\mathcal{R}) = D(\mathcal{R}_1) + D(\mathcal{R}_2) + D_2(\mathcal{R}_1) + D_2(\mathcal{R}_2)$. The same argument developed above that $D_2(\mathcal{R})$ can be computed in $O(1)$ time, can be used for $D_2(\mathcal{R}_1)$ and $D_2(\mathcal{R}_2)$ as well. Therefore, $D_1(\mathcal{R})$ can be computed in $O(1)$ time. Putting it all together, $D(\mathcal{R}) = D(\mathcal{R}_1) + D(\mathcal{R}_2) + \sum_g (w(\mathcal{R}_1)p(g|\mathcal{R}_1) \log p(g|\mathcal{R}_1) + w(\mathcal{R}_2)p(g|\mathcal{R}_2) \log p(g|\mathcal{R}_2) - w(\mathcal{R})p(g|\mathcal{R}) \log p(g|\mathcal{R}))$ which can be computed in $O(1)$ time. ■

Lemma 5 Let \mathcal{R}_1 and $\mathcal{R}_2 \subset \mathcal{R}_1$ be two regions. Let $\mathcal{R} = \mathcal{R}_1 - \mathcal{R}_2$. Given $D(\mathcal{R}_1), D(\mathcal{R}_2), w(\mathcal{R}_1), w(\mathcal{R}_2), p(g|\mathcal{R}_1), p(g|\mathcal{R}_2)$ one can compute $D(\mathcal{R}), p(g|\mathcal{R})$ for all g , and $w(\mathcal{R})$ in $O(1)$ time.

Proof : Similar to Proof of Lemma 4. ■

A similar set of lemmas and proofs for when the objective function is classification error (problem definition 2.2) is not detailed here due to lack of space (see[10] for details).

3.2 Refined Algorithm

We are now in a position to state our refined algorithm. Our refined algorithm is based on the fact that the calculation of the objective function for a control point can gainfully use much of the computation performed to calculate the objective function for a neighboring control point in $O(1)$ time. As there are $O(n^2)$ points in the pdf estimate there are $O(n^2)$ possible control points to evaluate. The steps of the algorithm are:

Step A: Precomputation Step: The precomputation step of the algorithm requires us to pre-evaluate the following rectangles¹ which takes $O(n^2)$ time[10]:

$$\forall i, j : 1 \leq i, j < n : [(x_i, y_1), (x_i, y_j)], [(x_i, y_{j+1}), (x_i, y_n)]$$

Step B: Evaluate the first cut point $C(x_1, y_j)$ in each row. This step takes $O(n^2)$ time since all the points in the grid need to be used to compute the objective function[10].

Step C: Traverse the row and evaluate the following cut points in the order: $C(x_1, y_j), C(x_2, y_j) \dots C(x_n, y_j)$ This step takes $O(n)$ time for each row, a follow through of the lemmas described earlier and the precomputed information. Overall since there are n rows this step also takes $O(n^2)$ time [10].

The overall running time of the 2D-Discretization algorithm, assuming the PDF has been pre-computed is therefore $O(n^2)$. The complete general version of the problem in the multi-attribute database case requires $O(m^2(n^2 + N \log n))$ time where m is the number of attributes and N is the total number of records in the database, and $n \times n$ is the pdf grid size. The $N \log n$ component reflects the cost to build the PDF for a given attribute pair. The n^2 component comes from the above 2D discretization algorithm. The multiplicative m^2 component comes from the number of possible base attribute pairs that have to be evaluated.

4 Parallel Incremental Algorithm

In this section we consider the problem of parallelizing the general version of the 2D-Discretization problem (refer to problem definition 2.3) in the presence of data updates. A naive strategy for parallelization would be to equi-partition the set of attribute pairs amongst the available processors and to simply construct the pdf for each pair of attributes that need to be evaluated and evaluate the objective function using the the refined algorithm presented in the previous section. This approach however may result in poor resource utilization and management and may incur unnecessary communication overheads. To improve on this we use global affine scheduling[11] which takes into account the data that has been processed on each node to determine which tasks to assign to whom. Basically, we identify cliques of tasks that minimize the set

¹technically we are all evaluating columns and sub-columns in a two dimensional matrix.

of resources (data columns) required to process them and assign these cliques to individual processors. While the parallelization is relatively straightforward, handling data updates is not so easy. Each data update can affect the PDF grid in two ways.

First, each time there is an update, the probabilities in each element of all the pairwise pdf grids will change. For instance if one has four elements in a grid, with each element having 1 transaction associated with it, the probabilities of each cell would be 0.25. Now, if we were to add one transaction to any one cell, the revised probabilities would be 0.4, 0.2, 0.2 and 0.2. All cell probabilities would have to be revised! Our solution to this problem is to maintain frequencies rather than cell probabilities. Maintaining frequencies has the advantage that only those cells that are updated will get changed. Then by computing probabilities on the fly (wherever needed² we can still take advantage of the results from the previous section.

Second, as we noted in Section 2 we break up each attribute space into equi-probable regions (depending on the number of desired grid points (control points)). The problem when we have updates is that the range and location of these equi-probable regions and therefore the location of the control points may shift. To address this issue, we compute the KL-distance between the current distribution induced by each base attribute and the ideal one (the equi-probable one), after each update. If the distributions are too far apart (as defined by a user-defined threshold) a re-normalization routine is triggered for the given attribute. Currently, to re-normalize we simply look at the entire column and compute the equi-probable partitions. A statistically equivalent approach for the discretization problem would be to re-normalize by just evaluating the distribution and inducing (approximating) the equi-probable partitions and the number of transactions within each partition. This approach would require applying a kernel filter[4] on the data that has already been processed as well as on the newly acquired (dynamic/streaming) data. This approach would trade off potential loss in accuracy for huge I/O savings. This is likely to be effective for our domain where we are dealing with large datasets.

The incremental aspects of the overall algorithm have also been parallelized. Both the normalization routines as well as the frequency grid updates are executed in parallel.

5 Experimental Evaluation

Our experimental evaluation was carried out on an 8 node dual pentium processor SMP cluster. Each SMP has 2GB of memory and 120GB of disk space. We assume that the dataset is centrally located and that updates happen in a periodic fashion. The data is partitioned vertically in a column by column format. We have implemented the parallel program using the MPI message-passing library (MPICH

²sometimes frequencies are enough.

over GM³). Access to the central dataset is via PVFS (over Myrinet), a parallel file system for Linux clusters. PVFS delegates the actual file operations to the native filesystem on our cluster: XFS-Linux. We use the ROMIO⁴ implementation to interface to PVFS. Two of the nodes were used as I/O servers for PVFS.

To perform our evaluations we used a set of synthetic datasets from a locally developed dataset generator. In our dataset generator one can vary the number of attributes (columns), the number of records (rows), the attribute ranges, and the attribute distributions (gaussian, uniform) and associated distribution parameters. In addition to the number of base attributes all datasets had one goal attribute with two categories C0 and C1. The user is also allowed to specify to the dataset generator if two or more attributes (including the goal attribute) are to be correlated in a specify way. We evaluated datasets within the following parameter ranges: #records [100,000-10,000,000], #attributes [15], attribute distributions [gaussian, uniform] with different parameters. This resulted in databases ranging from 60MB to 5.8GB in size. For each of the datasets we evaluated, we specified that two of the base attributes were correlated with the goal attribute in a manner similar to the XOR dataset described in [12]. This ensured that the best results would always be obtained when discretizing over these two attributes (a validation mechanism for our experiments). The data increments for each database was constructed in a similar way, often however, with different distributions as we wanted to stress-test the algorithm.

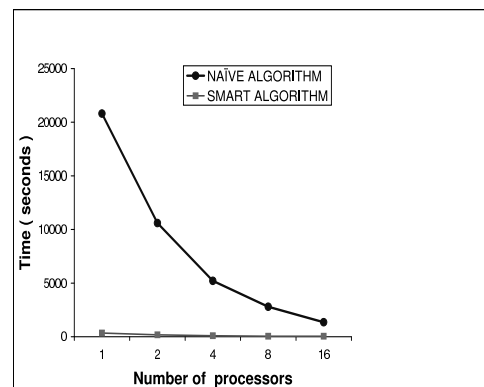


Figure 1. Comparison between Naive and Refined Approaches

5.1 Performance of Naive vs. Refined Approach

The first experiment we conducted was to empirically verify that the refined algorithm does outperform the naive algorithm significantly. In this experiment we compared the execution time of the naive algorithm to the execution time of the refined algorithm on the smallest of our three datasets (with 100,000) records. Note, that the execution

³www.myricom.com

⁴www-unix.mcs.anl.gov/romio

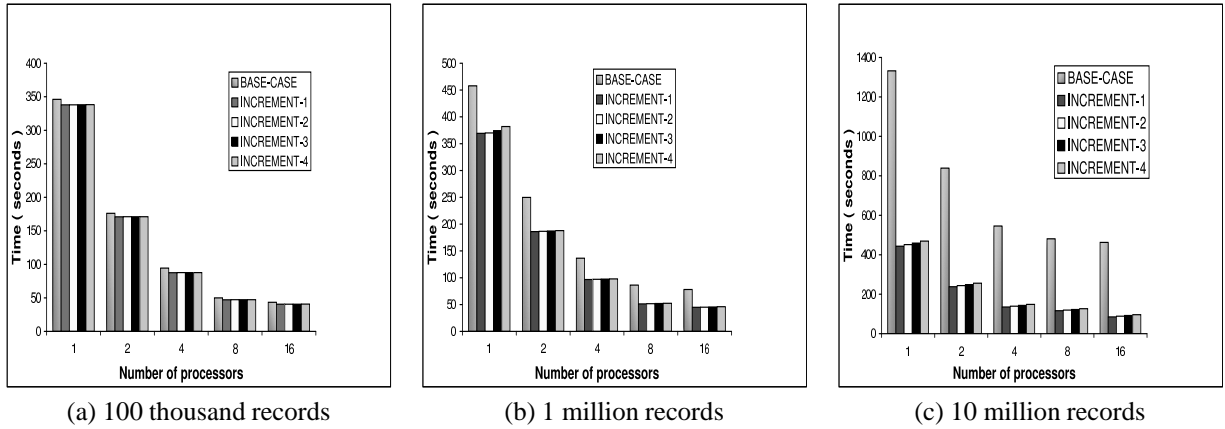


Figure 2. Parallel Incremental 2DD Performance

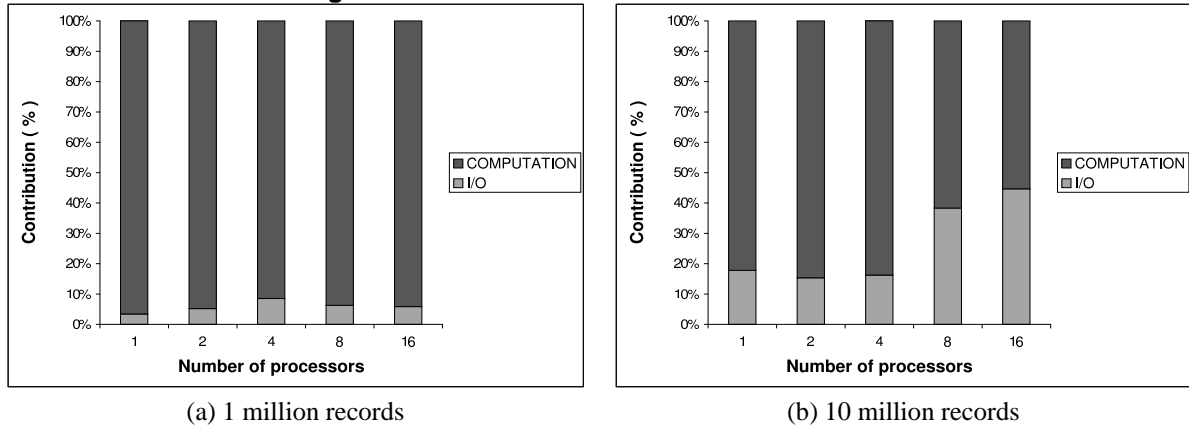


Figure 3. Execution Time Breakup: Computation and I/O

time of the algorithm does not really depend too much on the number of records but on the grid size ($N=128$). As we can see from Figure 1 the refined approach clearly outperforms the naive approach significantly. Although the Naive approach shows linear speedup properties, such properties are also exhibited by the refined approach as we shall see in the ensuing experiments. For the rest of the experiments we shall only consider the performance of the refined approach.

5.2 Parallel Incremental 2DD Performance

In this section we expand on the performance of the refined algorithm. Figure 2. The X-axis in this figure varies the number of processors used and the Y-axis reflects the total execution time of the program and the various incremental components. For each processor column the left-most bar reflects the base case performance of executing the algorithm on the original dataset. The bars to the right reflect the time taken to process additional increments (up to four are shown although the experiments can be run with continuous increments, i.e., streaming data) which account for roughly 10% of the original dataset (each). We observe two important trends. First, the refined algorithm seems to scale pretty well, for the smaller two datasets, up to 8 processors, yielding speedups ranging from 5.5 - 7. The

largest dataset does not scale very well. To see why this is so consider the running time of the algorithm as described in Section 3.2: $O(m^2(n^2 + N \log n))$. With large N the second part of this order equation will dominate (note that the first part is pretty much the same for all the datasets). The second part will result in heavy I/O traffic for larger values of N . In our current setup with only two I/O servers (under PVFS) there is increased access contention for larger processor counts. Also note that the I/O to computation ratio of the algorithm, for the larger datasets, is larger (since computation amount, which depends only on grid size, remains the same). These trends can also be observed from figure 3 which compares the I/O costs for the two larger datasets⁵.

Second, processing the original dataset vs. processing the increments does not seem to make much of a difference for the smaller dataset (improvement factor is a little more than one) but does seem to matter for the larger datasets (improvement factor is between three and four for the largest dataset). Processing an increment differs from processing the entire dataset in only one respect, in terms of generating the pdf grid (an I/O intensive task as seen above). The computational aspects of both problems are

⁵The I/O contribution towards the overall execution time of the algorithm on the dataset with 100,000 transactions is negligible.

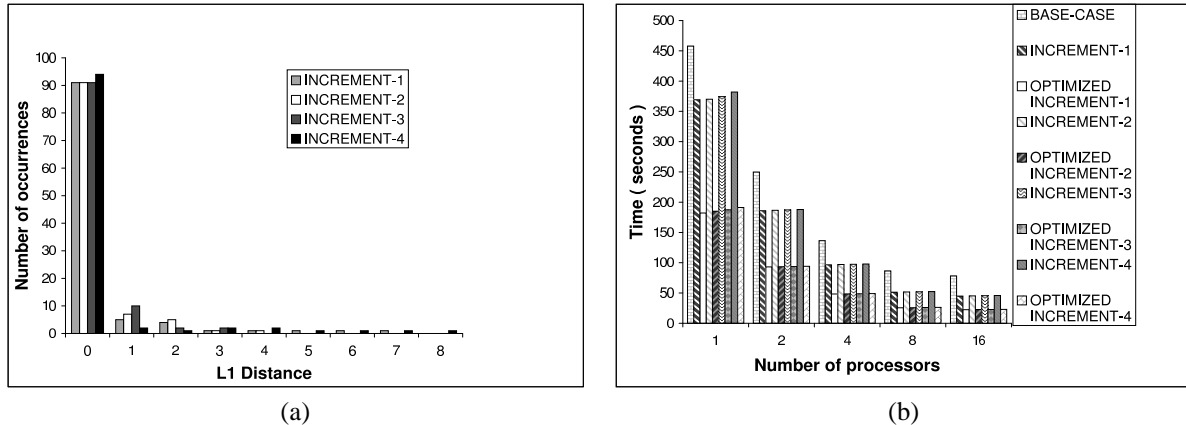


Figure 4. Improving the Incremental Process

the same. Since in the case of the smaller datasets the computation time dominates there is no noticeable difference. However, in the larger datasets since the I/O to computation ratios are higher (see Figure 3) processing the increments is relatively quicker (accounting for the factor improvements). Note also in Figure 2, that when the number of processors is increased the factor of improvement is more noticeable (as the I/O contribution is higher (Figure 3)). Note, that trend 2 alleviates the I/O problem caused by trend 1 and allows the increment processing components to scale much better for the larger datasets.

5.3 Improving the Incremental Process

As mentioned earlier the computational component for both the incremental component and the original component is the same and is dictated by the grid size. However, one observation that can alleviate this problem, is that the increment is a small percentage (typically less than 10% of original) of the original dataset. As a result the increment should not affect the position of the control point by much. To evaluate this aspect, we instrumented our algorithm on the 1-million transaction dataset and computed for each increment the resulting shift (measured by the popular L1 metric) in the optimal control points for each of the 105 pairwise pdfs. The graph of this L1 shift is plotted in Figure 4A. Clearly the majority of the points were shifted by less than 2 units or less. Now, armed with this information, (and we observed similar behavior on many different datasets with different increment distributions) we modified the incremental component to only search within a 2 unit neighborhood of the previous best control point. By doing so we noted that overall accuracy of the method drops very little. But doing so can reduce the computational time of the incremental component drastically (see optimized increments in Figure 4B).

5.4 I/O Performance

Since I/O performance affected some of the above performance results adversely we wanted to evaluate the impact of using PVFS (with two I/O nodes) as it has cur-

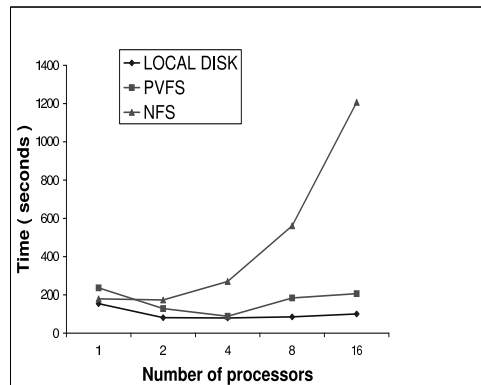


Figure 5. Performance comparison of various I/O access methodologies

rently been deployed on our system and compare its performance against an ideal (replicated dataset, local disk based access). We also include the performance results for an NFS based implementation to serve as a benchmark for comparison. In this experiment we compare the I/O performance of the algorithm executing on the 10 million transaction database under the above 3 different disk access scenarios. The results are documented in Figure 5. Not surprisingly, NFS performs very poorly as we increase the number of processors. PVFS for a single process performs quite poorly due to the increased overheads associated with a parallel file system (locking/synchronization, meta-data access etc.). From two to four processors the performance of PVFS compares very well with the performance of the replicated local disk access (ideal scenario). At 8 processors the PVFS based scheme again suffers somewhat due to contention effects (two I/O servers and also due to the fact that at least two of the nodes have both computation processes and I/O processes that contend for local resources). At 16 processors the performance of both the local disk-based scheme and PVFS falters, relative to the 8-processor performance because of contention effects (recall we have 8 nodes and two processors per node. In the PVFS case two of the nodes are overloaded

due to PVFS related services.). Basically, based on the results for two and four processors we can expect that our PVFS-based implementation can scale if the number of I/O servers are increased.

6 Related Work

Most work on discretization focuses on discretizing a single continuous attribute. These methods can compute optimal discretizations along one dimension, but they cannot generate optimal discretizations for the two dimensional case. Dougherty *et al* [5], present an excellent classification of current methods in discretization along three separate axes, viz., global vs. local, supervised vs. unsupervised and static vs. dynamic. Amongst the discretization methods reviewed in [5] and elsewhere, the following are the most germane to our work.

The simplest discretization method is an unsupervised static method called equal sized discretization. It calculates the maximum and minimum for the attribute being discretized and simply partitions the range observed into (some k) equal sized intervals. Another unsupervised static method is equal frequency discretization. It counts the number of values we have from the attribute that we are trying to discretize and partitions it into intervals containing the same number of examples. ChiMerge is a supervised, incremental, bottom up method described by Kerber [7]. It suggests that intra-interval similarity should be maximized and inter-interval similarity should be minimized. ChiMerge uses the Chi-Squared statistic to determine the independence of the class from the two adjacent intervals. Entropy discretization is an supervised dynamic method described in Fayyad et al [6]. Entropy discretization recursively selects the cut-points, minimizing entropy and uses the minimum-description-length principle to determine the appropriate number of intervals (stopping criteria). An improvement on this approach was presented recently by Subramonian et al [15]. Maass [9] provides an efficient algorithm that minimizes classification error. Catlett [2] describes a supervised dynamic discretization method that recursively selects cut-points maximizing Quinlan's gain [13] until a stopping criteria based on a set of heuristic rules ends the recursion.

Recently, we formulated and discussed the 2D discretization problem and an interactive approach to solve it [12]. In the above work we compared the approach with a state of the art algorithm and found that we obtain similar results at a fraction of the computational cost while generating a discretization that can be described with fewer partitions. To solve the 2D discretization problem efficiently, we considered an approximate solution based on simulated-annealing search. In the current work we provide an exact solution to this problem in a time-optimal manner. Additionally, none of the above solutions considered the problem of discretizing in the presence of dy-

namic updates, or parallelizing the process over a network of workstations, which we do consider in this paper.

7 Conclusions

In this paper we considered the problem of 2D-Discretization on dynamic datasets. We presented a time and result-optimal solution to the basic problem along with a parallel incremental version of the multi-attribute 2D discretization problem. Experimental results confirm that our approach results in execution time improvement of up to several orders of magnitude for even very large datasets (greater than 5GB) versus the naive approach. Our parallel implementation also addresses I/O and computational scalability issues of the parallel incremental approaches.

References

- [1] P.H. Carns, W.B. Ligon III, R.B. Ross and R. Thakur. PVFS: A Parallel File System For Linux Clusters. In *PROC of the 4th Annual Linux Showcase and Conference*, 2000.
- [2] J. Catlett. Changing continuous attributes into ordered discrete attributes. *European Working Session on Learning*, 1991.
- [3] Thomas Cover and Joy Thomas. *Elements of Information Theory*. Wiley and Sons, 1991.
- [4] L. Devroye. A course in density estimation. In *Birkhauser: Boston MA*, 1987.
- [5] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *12th International Conference on Machine Learning*, 1995.
- [6] Usama. M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *14th Joint Conference on AI*, 1993.
- [7] Randy Kerber. Chimerge: Discretizaion of numeric attributes. In *National Conference on AI*, 1991.
- [8] Stephen Lord. Porting XFS to Linux. In *Ottawa Linux Symposium*, July 2000.
- [9] W. Maass. Efficient agnostic pac-learning with simple hypotheses. In *ACM COLT*, 1994.
- [10] S. Parthasarathy and A. Ramakrishnan. Parallel Incremental 2D Discretization. OSU Tech. Rep., 2002.
- [11] S. Parthasarathy and R. Subramonian. Facilitating data mining on a network of workstations. In *Advances in Distributed and Parallel Knowledge Discovery*, , AAAI Press, 2000.
- [12] S. Parthasarathy, R. Subramonian, and R. Venkata. Generalized discretization for summarization and classification. In *PADD 1998*.
- [13] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 5(1):71–100, 1986.
- [14] R.B. Ross Providing Parallel I/O on Linux Clusters. In *Second Annual Linux Storage Management Workshop*, 2000.
- [15] R. Subramonian, R. Venkata, and J. Chen. A visual interactive framework for attribute discretization. In *KDD*, 1997.
- [16] P. Smyth and D. Wolpert. Anytime exploratory data analysis for massive data sets. In *KDD*, 1997.