

Fast and Effective Histogram Construction*

Felix Halim

Panagiotis Karras

Roland H. C. Yap

School of Computing
National University of Singapore
{halim, karras, ryap}@comp.nus.edu.sg

ABSTRACT

Histogram construction or sequence segmentation is a basic task with applications in database systems, information retrieval, and knowledge management. Its aim is to approximate a sequence by line segments. Unfortunately, the quadratic algorithm that derives an optimal histogram for Euclidean error lacks the desired scalability. Therefore, sophisticated approximation algorithms have been recently proposed, while several simple heuristics are used in practice. Still, these solutions fail to resolve the efficiency-quality tradeoff in a satisfactory manner. In this paper we take a fresh view on the problem. We propose conceptually clear and scalable algorithms that efficiently derive high-quality histograms. We experimentally demonstrate that existing approximation schemes fail to deliver the desired efficiency and conventional heuristics do not fare well on the side of quality. On the other hand, our schemes match or exceed the quality of the former and the efficiency of the latter.

Categories and Subject Descriptors

F.2 [Analysis of Algorithms and Complexity]: Miscellaneous; H.3 [Information Storage and Retrieval]: Miscellaneous

General Terms

Algorithms, Experimentation, Performance

1. INTRODUCTION

A *histogram* or *segmentation* aims to approximate a sequence of values by piecewise-constant line segments that effectively capture the basic features of the underlying data. This form of approximation finds applications in database systems, as in the approximation of intermediate join results by a query optimizer [13, 30] and approximate query processing [33, 3]; the same form of approximation is used in

*Work supported by Singapore MOE's AcRF grants T12-0701-P01 and T1 251RES0807.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CIKM'09, November 2–6, 2009, Hong Kong, China.

Copyright 2009 ACM 978-1-60558-512-3/09/11 ...\$10.00.

knowledge management applications (as in decision-support systems [1, 17, 39]), in bio-informatics [37], and in information retrieval applications as time-series indexing and similarity search [4]. An overview of the area from a database perspective is provided in [14, 15].

In all cases, a segmentation algorithm is employed in order to divide a given data sequence into a given budget of consecutive *buckets* or *segments* [12]. All values in a segment are approximated by a single representative. Both these representative values, and the bucket *boundaries* themselves, are chosen so as to achieve a low value for an error metric in the overall histogram approximation. A useful metric is the *Euclidean* error. Approximate answers to both *point* and *range queries* can be estimated with a histogram [11].

A histogram optimal in terms of Euclidean error, called *V-Optimal*, is the *optimal choice* when approximating attribute value frequencies in a database for the estimation of the result size of simple equality join and selection queries [16]. This optimal segmentation is derived by an $O(n^2B)$ dynamic-programming algorithm that recursively examines all possible solutions [2, 20]. Still, this algorithm is not scalable enough to be used in practice [18]; database systems employ heuristics for histogram construction.

Recent research has revisited the histogram construction problem from the point of view of approximation algorithms. Guha et al. proposed a suite of approximation and streaming algorithms for histogram construction problems [10]. Of the algorithms proposed in [10], the AHistL- Δ proves to be the best for offline approximate histogram construction. In a nutshell, AHistL- Δ builds on the idea of approximating the error function itself, while pruning the computations of the V-Optimal algorithm. Likewise, Terzi and Tsaparas recently proposed DnS, an offline approximation scheme for sequence segmentation [38]. DnS divides the problem into subproblems, solves each of them optimally, and then utilizes V-Optimal to construct a global solution by merging the segments created in the partial solutions.

However, despite their theoretical elegance, the approximation algorithms proposed in previous research do not always resolve the tradeoff between time complexity and histogram quality in a satisfactory manner. The running time of these algorithms can approach that of the quadratic-time V-Optimal solution. Still, the quality of segmentation they achieve can substantially deviate from the optimal. Furthermore, previous research has not examined how the different approximation algorithms of [10] and [38] compare to each other in terms of efficiency and effectiveness.

In this paper, we propose a middle ground between the

theoretical elegance of approximation algorithms on the one hand, and the simplicity, efficiency, and practicality of heuristics on the other. We develop segmentation algorithms that are both fast and effective, even though they do not provide approximation guarantees with respect to the optimal solution. Still, our solutions have the scalability and the quality that allows them to be employed in practice, instead of the currently used heuristics, when dealing with the segmentation of very large data sets. We conduct the first, to our knowledge, experimental study of state-of-the-art optimal, approximation, and heuristic algorithms for histogram construction. This study demonstrates that our algorithms vastly outperform the guarantee-providing approximation schemes in terms of running time, while achieving comparable or superior approximation accuracy.

2. BACKGROUND AND RELATED WORK

In this section we briefly present previous approaches to histogram construction or segmentation, with a Euclidean-error guarantee in mind. Given an n -size data vector $\mathbf{D} = \langle d_0, d_1, \dots, d_{n-1} \rangle$, the problem is to devise an approximate representation $\hat{\mathbf{D}}$ of \mathbf{D} using at most B space units, so that the *Euclidean error* of the approximation is minimized. This Euclidean error is expressed as

$$\mathcal{L}_2(\hat{\mathbf{D}}, \mathbf{D}) = \left(\frac{1}{n} \sum_i |\hat{d}_i - d_i|^2 \right)^{\frac{1}{2}} \quad (1)$$

\hat{d}_i denotes the approximate estimated value for d_i . An algorithm that aims to minimize \mathcal{L}_2 in practice works on the sum-of-squared-errors (SSE) $\sum_i |\hat{d}_i - d_i|^2$. Previous studies [20, 5, 23, 9, 36, 24, 25, 26, 21, 22] have generalized their results into wider classes of *maximum*, *distributive*, *Minkowski-distance*, and *relative-error* metrics. Still, the Euclidean error \mathcal{L}_2 remains an important error metric for several applications, such as database query optimization [16], context recognition [12], and time series mining [4].

Depending on the application domain, the same approximate representation of \mathbf{D} is called a *histogram*, a *segmentation* [38], a *partitioning*, or a *piecewise-constant approximation* [4]. Such a representation divides \mathbf{D} into $B \ll n$ disjoint intervals $[b_i, e_i]$, where b_i and e_i are *indices* of the data items and $1 \leq i \leq B$; these intervals are called *buckets* or *segments*. Each segment is attributed a single representative value v_i , which approximate all values d_j under its scope, $j \in [b_i, e_i]$. A single bucket is described by the triplet $s_i = \{b_i, e_i, v_i\}$. $2B - 1$ numbers suffice for a representation of B buckets, given that $\forall i, 1 < i \leq B, b_i = e_{i-1} + 1$, and the two endpoints are fixed.

For a given target error metric, the representative value v_i of a bucket that minimizes the resulting approximation error is straightforwardly defined as a function of the data values in the bucket. Thus, for the *average absolute error* the best v_i is the *median* of the values in $[b_i, e_i]$ [38]; for the *maximum absolute error* it is the mean of the maximum and minimum value in $[b_i, e_i]$ [27]; an analysis of respective relative-error cases is offered in [11]. For the Euclidean error \mathcal{L}_2 that concerns us, the optimal value of v_i is the *mean* of values in $[b_i, e_i]$ [20]. The boundary positions e_1, \dots, e_{B-1} of a partitioning that achieves a *minimum* approximation error depend on the target error metric too. The task of a *segmentation algorithm* is to define boundary positions that achieve low approximation error for the error metric at hand.

2.1 V-Optimal Histogram Construction

The $O(n^2B)$ dynamic-programming (DP) algorithm that constructs an optimal segmentation under the \mathcal{L}_2 metric is a special case of Bellman’s general line segmentation algorithm [2]. It was presented by Jagadish et al. [20] and optimized in terms of space-efficiency by Guha [8]. Its basic underlying observation is that the optimal b -segment histogram of a data vector \mathbf{D} can be recursively derived given the optimal $(b-1)$ -segmentations of all prefix vectors of \mathbf{D} . Thus, the minimal sum-of-squared-errors (SSE) $E(i, b)$ of a b -bucket histogram of the prefix vector $\langle d_0, d_1, \dots, d_i \rangle$ is recursively expressed as:

$$E(i, b) = \min_{b \leq j < i} \{E(j, b-1) + \mathcal{E}(j+1, i)\} \quad (2)$$

where $\mathcal{E}(j+1, i)$ is the minimal SSE for the segment $\langle d_{j+1}, \dots, d_i \rangle$. This error is easily computed in $O(1)$ based on a few pre-computed quantities (sums of squares and squares of sums) for each prefix [20]. Thus, this algorithm requires a $O(nB)$ tabulation of minimized error values $E(i, b)$ along with the selected optimal last-bucket boundary positions j that correspond to those optimal error values. As noted by Guha, the space complexity is reducible to $O(n)$ by discarding the full $O(nB)$ table; instead, only the two *running* columns of this table are stored. The *middle bucket* of each solution is kept track of; after the optimal error is established, the problem is divided in two half subproblems and the same algorithm is recursively re-run on them, until all boundary positions are set [8]. The runtime is significantly improved by a simple pruning step [20]; for given i and b , the loop over (decreasing) j that searches for the min value in Equation 2 is broken when $\mathcal{E}(j+1, i)$ (non-decreasing as j decreases) exceeds the running minimum value of $E(i, b)$.

Unfortunately, the quadratic time complexity of V-Optimal renders it inapplicable in most real-world applications. Thus, several works have proposed approximation schemes for histogram construction [6, 7, 10, 38].

2.2 The DnS Algorithm

Terzi and Tsaparas correctly observed that a quadratic algorithm is not an adequately fast solution for practical sequence segmentation problems [38]. As an alternative to the V-Optimal algorithm, they suggested a sub-quadratic constant-factor approximation algorithm.

The basic idea behind this *divide and segment* (DnS) algorithm is to divide the overall segmentation problem into smaller subproblems, solve those subproblems optimally, and then combine their solutions. The V-Optimal algorithm serves as a building block of DnS. The problem sequence is arbitrarily partitioned into smaller subsequences. Each of those is optimally segmented using V-Optimal. Then, the derived segments are treated as the input elements themselves, and a segmentation (i.e., local merging) of them into B larger buckets is performed using the V-Optimal algorithm again. A thorough analysis of this algorithm demonstrates an approximation factor 3 in relation to the optimal \mathcal{L}_2 error, and a worst-case complexity of $O(n^{4/3}B^{5/3})$, assuming that the original sequence is partitioned into $\chi = \left(\frac{n}{B}\right)^{2/3}$ equal-length segments in the first step.

2.3 The AHistL- Δ Algorithm

Guha et al. have provided a collection of approximation algorithms for the histogram construction problem. Out of

them, AHistL- Δ is their algorithm of choice for offline histogram construction [10].

The basic observation underlying the AHistL- Δ algorithm is that the $E(j, b - 1)$ function in Equation 2 is a non-decreasing function of j , while its counterpart function $\mathcal{E}(j + 1, i)$ is a non-increasing function of j . Thus, instead of computing the entire tables of $E(j, b)$ over all values of j , this non-decreasing function is approximated by a staircase histogram representation - that is, a histogram in which the representative value for a segment is the highest (i.e., the rightmost) value in it. In effect, only a few representative values of $E(j, b - 1)$, i.e., the end values of the staircase intervals, are used. Moreover, the segments of this staircase histogram themselves are selected so that the value of the $E(j, b)$ function at the right-hand end of a segment is at most $(1 + \delta)$ times the value at the left-hand end, where $\delta = \frac{\epsilon}{2B}$. The recursive formulation of Equation 2 remains unchanged, with the difference that the variable j now only ranges over the endpoints of intervals in this staircase histogram representation of $E(j, b)$.

On top of this observation, the AHistL- Δ algorithm adds the further insight that any $E(j, b)$ value that exceeds the final SSE of the histogram under construction (or even an approximate version of that final error) *cannot* play a role in the eventual solution. Since $E(j, b)$ form partial contributions to the *aggregate* SSE, only values smaller than the final error Δ contribute to the solution. Thus, if we knew the error Δ of the V-Optimal histogram in advance, then we could eschew the computation of any $E(j, b)$ value that exceeds it. Since Δ is not known in advance, we can still work with estimates of Δ in a binary-search fashion.

The AHistL- Δ algorithm combines the above two insights. In effect, the problem is decomposed in two parts. The *inner* part returns a histogram of error less than $(1 + \epsilon)\Delta$, under the assumption that the given estimate Δ is correct, i.e., there exists a histogram of error Δ ; the *outer* part searches for such a well-chosen value of Δ . The result is an $O(n + B^3(\log n + \epsilon^{-2}) \log n)$ -time algorithm that computes an $(1 + \epsilon)$ -approximate B -bucket histogram.

2.4 Existing Heuristics

Past research has also proposed several heuristics for histogram construction. Some of these heuristics are relatively brute-force segmentations; this category includes methods such as the end-biased [16], equi-width [28], and equi-depth [32, 31] heuristics.

A more elaborate heuristic is the MaxDiff [35] method. According to this method, the $B - 1$ points of highest difference between two consecutive values in the original data set are selected as the boundary points of a B -bucket histogram. Its time complexity is $O(n \log B)$, i.e. the cost of inserting n items into a priority queue of B elements. Poosala and Ioannidis conducted an experimental study of several heuristics employed in database applications and concluded that the MaxDiff-histogram was “probably the histogram of choice”. Matias et al. used this method as the conventional approach to histogram construction for selectivity estimation in database systems, in comparison to their alternative proposal of wavelet-based histograms [30].

Jagadish et al. [20] have suggested a one-dimensional variant of the multidimensional MHIST heuristic proposed by Poosala and Ioannidis [34]. This is a greedy heuristic that repeatedly selects and splits the bucket with the high-

est SSE, making B splits. The same algorithm is mentioned by Terzi and Tsaparas by the name Top-Down [38]; a similar algorithm has been suggested in the context of multidimensional anonymization by LeFevre et al. [29]. Its worst-case time complexity is $O(B(n + \log B))$, i.e., the cost to create a heap of B items while updating affected splitting costs at each step. In the pilot experimental study of [20], MaxDiff and MHIST turn out to be the heuristics of choice; it is observed that the former performs better on more spiked data, while the latter is more competitive on smoother data.

3. MOTIVATION

We observe that a gap exists in the research on segmentation. While several heuristics have been known for a long time, and more sophisticated approximation schemes have been recently proposed, it is unclear whether any them resolves the tradeoff between efficiency and accuracy in a satisfactory manner. Besides, there has been no face-to-face comparison between them.

In particular, it is not clear whether the recently proposed approximation schemes can deliver the time-efficiency that will render them an attractive choice versus the near-linear simple heuristics. The V-Optimal DP algorithm fails to respond to this challenge; approximation schemes of time super-linear in n and/or B may still be problematic in terms of time-efficiency. Thus, there is still a need for an approach that would provide both near-optimal quality and near-linear time complexity. In this paper, we provide algorithms that fill this gap.

4. FAST AND EFFECTIVE HISTOGRAM CONSTRUCTION

Our aim is to develop segmentation algorithms that can achieve both good quality (i.e., low approximation error) and time efficiency (i.e., near-linear time complexity). We start out by presenting a simple greedy algorithm.

4.1 A Basic Greedy Algorithm

The rationale of our algorithm is to begin with an original, *ad hoc* segmentation \mathcal{S}_0 and then greedily modify its boundary positions. \mathcal{S}_0 can be randomly created, or it can be that of a simple heuristic, for example an equi-width histogram [28]. The greedy boundary modification operations matter for the final result more than the choice of \mathcal{S}_0 . These operations iteratively move one boundary from one position to another. The objective of these moves is reduce the total \mathcal{L}_2 error. For that purpose, we maintain a min-heap H^+ of all *running* boundary positions b_i , $1 \leq i \leq B - 1$ (b_0 and b_B are fixed) along with the potential *error increase* $\Delta^+ E_i$ that can be incurred if b_i is removed. At each iteration, the boundary b_i^* of *minimum* $\Delta^+ E_i$ is discarded.

Furthermore, we also maintain a max-heap H^- of all *running* segments s_j , $1 \leq j \leq B$, where $s_j = [b_{j-1}, b_j]$, along with their potential *error decrease* $\Delta^- E_j$ that can be effected if s_j is *optimally* split into two subsegments. At each iteration, the removed boundary is repositioned so as to *split* the segment of *maximum* $\Delta^- E_j$. Thus, at each iteration, the *locally optimal* boundary to remove is lifted, the *locally optimal* segment to split is partitioned in two, and heaps are accordingly updated. This step is repeated until no further move that decreases the total error can be made. Figure 1 shows a pseudo-code for this GDY algorithm.

Algorithm GDY(B)

Input: space bound B , n -data vector $\mathbf{D} = [d_0, \dots, d_{n-1}]$
Output: histogram \mathbf{H} of B segments

1. $S_0 = \text{random } B - 1 \text{ segment boundaries on } \mathbf{D};$ // initialize
2. $i = 0;$ Populate H^+ and H^- with S_i ;
3. **while** (H^+ is not empty)
4. $i = i + 1;$ $S_i = S_{i-1};$
5. $G = \text{take boundary with minimum } \Delta^+ E_i \text{ from } H^+;$
6. Remove G from S_i and update H^+ and H^- ;
7. $P = \text{take segment with maximum } \Delta^- E_j \text{ from } H^-;$
8. **if** ($\Delta^+ E_i - \Delta^- E_j >= 0$)
9. Undo steps 6 and 7; // G is discarded
10. **else**
11. Split segment P , add new boundary to S_i ;
12. Update partitions' costs in H^+ and H^- ;
13. **return** S_i ;

Figure 1: GDY algorithm

The time complexity of GDY is $O(M(\frac{n}{B} + \log B))$, where M stands for the number of moves, $\frac{n}{B}$ for the (average) cost of calculating the optimal split position for a newly created segment after each move, and $\log B$ for the overhead of selecting the best-choice boundary to remove and segment to split using the heaps, and updating them at each step.

4.2 The Greedy-DP Algorithm

We now strive to add some extra sophistication in the operation of this simple greedy heuristic. Our enhanced approach starts out from the observation that a dynamic-programming (DP) algorithm that aims to discover a good segmentation does not need to examine every possible boundary position per se; it can constrain itself to a limited set of candidate boundary positions that are deemed to be likely to participate in the optimal solution. Our objective is to identify a set of such candidate boundary positions, which we call *samples*.

Our approach is inspired from the methodologies of Terzi and Tsaparas [38] and Guha et al. [10], but also contains a departure from them. These authors also aim to eschew part of the DP computation without altogether discarding the DP itself. Still, their approaches are burdened by the need to provide provable approximation guarantees. This meticulousness hampers efficiency. We aim to preserve the quality advantage of such an approach, but avoid the efficiency overhead.

In particular, AHistL- Δ tries to carefully discard from the DP recursion those terms (i.e., candidate boundary positions) whose error contribution can be *approximated* by that of their peers [10]. Likewise, DnS attempts to acquire a set of samples (i.e., candidate boundary positions) by running the DP recursion itself in a small scale, within each of the χ subsequences it creates [38], ending up with χB samples. Thus, DP is applied in a two-level fashion, both at a micro-scale, within each of the χ subsequences, and at a macro-scale, among the derived boundaries themselves. While this approach allows for an error guarantee, it unnecessarily burdens what is anyway a suboptimal algorithm with super-linear time complexity. Likewise, in its effort to provide a *bounded approximation* of every error quantity that enters the problem, AHistL- Δ ends up with an $O(B^3(\log n + \epsilon^{-2})\log n)$ time complexity factor that risks exceeding even the runtime of V-Optimal itself in practice.

In a departure from the above approaches, we opt for a *lighter* version of eschewing part of the DP computation and selecting samples (i.e., candidate boundary positions). We simply run the fast GDY algorithm itself multiple times altering the initial input partitioning in each run, and collect

the boundary positions it ends up with, eliminating duplicates. *All* boundary positions collected in this fashion are themselves participants in a B -segmentation of the input sequence that GDY could not improve further; thus, they are reasonable candidates for an *optimal* B -segmentation. We emphasize that this approach to sample collection contrasts to the one followed in [38]; the samples in [38] do not *themselves* participate in a global B -segmentation, but only in local B -segmentations of subsequences. Thus, even though that methodology allows for the computation of an elegant approximation guarantee, most of the samples it works with are unlikely to participate in an optimal B -segmentation. A similar observation holds for AHistL- Δ . This algorithm endeavors to discard computations related to those boundary positions that have produced error upper-bounded by that of one of their neighbors in an examined subproblem; however, it does not aim to work on boundary positions that are more likely to *participate in the optimal solution* per se.

In case B is less than \sqrt{n} , we run I iterations of GDY until we collect up to $O(\sqrt{n})$ samples. Then, we run the V-Optimal DP segmentation algorithm on this set of samples, in order to select a subset of B out of them that define a minimum-error histogram. Although this is a quadratic algorithm, it behaves as linear in n , taking advantage of the fact that the input size is bounded by \sqrt{n} and $O(\sqrt{n}\sqrt{n}) = O(n)$. Thus, dynamic programming is used to provide a high-quality histogram without sacrificing the linear time complexity of GDY. Figure 2 presents a pseudo-code for this GDY_DP algorithm.

Algorithm GDY_DP(B, I)

Input: bound B , number of runs I , n -data vector $[d_0, \dots, d_{n-1}]$
Output: histogram partitioning \mathbf{H} of B segments

1. $S = \text{empty set of sample boundaries}$
2. **loop** (I times)
3. $\mathcal{P} = \text{GDY}(B);$ // run GDY with random initialization
4. $S = S \cup \mathcal{P};$
5. **return** $\text{DP}(S, B);$

Figure 2: GDY_DP algorithm

4.3 A Batch-Processing Variant

When B is larger than \sqrt{n} , then GDY_DP collects $O(B)$ samples and loses its linear-in- n character, but still runs faster than both DnS and AHistL- Δ . Still, to deal with larger values of B , we propose a batch version of GDY_DP. The idea is to process only \sqrt{n} samples at a time. That is, we divide the sorted sequence of collected samples into subsequences of at most \sqrt{n} consecutive samples; we run separate DP segmentations on each those; and we augment the results into a global B -segmentation. Our approach is reminiscent of the suggested piecewise application of a summarization scheme in [26]. However, it combines the batched or piecewise processing approach with a sample selection mechanism, as in [38]. Thus, it contains the size of the problem in two ways: (i) it examines only selected samples instead of the whole data; and (ii) it processes the data in batches.

In more detail, we use an *auxiliary* segmentation \mathcal{A} , which is provided by a single run of GDY. For each group \mathcal{G} of \sqrt{n} consecutive samples, we find two *dividers* l_a and r_a , among the boundaries in \mathcal{A} , that enclose \mathcal{G} . We then run V-Optimal on the set \mathcal{G} , so as to produce as many boundaries between l_a and r_a as the segmentation \mathcal{A} has allocated in this interval. Thus, the number of boundaries in the derived segmentation is kept appropriate, but their positions are bettered within each \sqrt{n} -boundary subinterval. The intuition is that we

improve on the boundary positions derived by GDY in \mathcal{A} , but we do so in a more sophisticated manner than just greedily moving one boundary at a time. Thus, the quality of the segmentation is improved.

In each group \mathcal{G} we are dealing with \sqrt{n} samples, while there are $O(I \times B)$ samples in total to deal with, where I the constant pre-selected number of GDY runs that produces them. Thus, there are $O\left(\frac{IB}{\sqrt{n}}\right)$ groups of \sqrt{n} samples each, hence the algorithm performs as many separate DP runs. Each run chooses a fraction of the \sqrt{n} samples in group \mathcal{G} , performing an $O\left(\sqrt{n}^3\right) = O(n\sqrt{n})$ DP segmentation over them. Thus, the overall worst-case time complexity of the algorithm is $O(InB) = O(nB)$. Figure 3 depicts a pseudo-code for this batch-DP algorithm GDY_BDP.

Algorithm GD_Batched_DP(B, I)

Input: space bound B , number of GDY runs I ,
 n -data vector $[d_0, \dots, d_{n-1}]$
Output: histogram partitioning \mathbf{H} of B segments

1. $\mathcal{S} = \text{collect samples}$ by running randomized GDY(B) I times;
2. $\mathcal{A} = \text{GDY}(B)$; // an *auxiliary* solution to be improved
3. $\mathcal{S} = \mathcal{S} \cup \mathcal{A}$;
4. $l_s = 0$; // left group divider in \mathcal{S}
5. $l_a = 0$; // left group divider in \mathcal{A}
6. **while** ($l_s + 1 < |\mathcal{S}|$)
7. $r_s = \min\{|\mathcal{S}| - 1, l_s + \sqrt{n}\}$; // right divider in \mathcal{A}
8. $r_a = \text{first boundary in } \mathcal{A} \text{ after } r_s$; // right divider in \mathcal{S}
9. $r_s = \text{matching boundary of } r_a \text{ in } \mathcal{S}$;
10. $\bar{B} = r_a - l_a$; $\mathcal{G} = \mathcal{S}[l_s..r_s]$;
11. $\text{optimalSubPars} = \text{DP}(\mathcal{G}, \bar{B})$
12. Replace $\mathcal{A}[l_a..r_a]$ with optimalSubPars
13. $l_s = r_s$; $l_a = r_a$; // update left index for next batch
14. **return** \mathcal{A} ; // the improved aux solution

Figure 3: GDY_BDP algorithm

The rationale behind GDY_BDP is that, when the number of boundaries is large, selecting some breakpoints for them based on a simple GDY solution and then performing GDY_DP within the \sqrt{n} -sample intervals defined by these breakpoints does not hamper the overall quality too much. On the contrary, it confers near-optimal quality and allows for near-linear time efficiency. As we shall see, the quality achieved with GDY_BDP is always very close to that of GDY_DP, while GDY_BDP is much faster on large B . GDY_BDP also avoids a major loophole in the methodology of DnS. Namely, DnS *forces* each of the *arbitrarily selected* subsequences it works on to produce B samples, and then chooses from the total pool of samples. Thus, it does not pay attention to the actual form of the data. Cases where some data regions require much denser segmentation than other regions are not satisfactorily covered by DnS, but they are covered by GDY_BDP. To our knowledge, no other histogram construction algorithm scales well in both the input size n and the number of segments B , while producing, as we will see, near-optimal quality in terms of error.

Method	Time	Ref
V-Optimal	$O(n^2 B)$	[20]
DnS	$O(n^{4/3} B^{5/3})$	[38]
AHistL- Δ	$O(n + B^3 (\log n + \epsilon^{-2}) \log n)$	[10]
MaxDiff	$O(n \log B)$	[35]
MHIST	$O(B(n + \log B))$	[20]
GDY	$O\left(M \left(\frac{n}{B} + \log B\right)\right)$	this
GDY_DP	$O(nB) \ (n < \sqrt{B})$	this
GDY_BDP	$O(nB)$	this

Table 1: Complexity comparison

5. EXPERIMENTAL EVALUATION

This section presents our extensive experimental comparison of known approximation and heuristic algorithms and the solutions we have proposed. In particular, we have compared the following algorithms:

5.1 The Algorithms

- **V-Optimal** The optimal Euclidean-error histogram construction algorithm of Jagadish et al. [20]. This algorithm is a specialization of the line-segmentation technique proposed by Bellman [2] (see Section 2.1). We add the denotation 2 in the algorithm’s name to indicate the fact that we utilize the simple pruning suggested in [20].
- **DnS** The algorithm for sequence segmentation suggested by Terzi and Tsaparas [38] (see Section 2.2). This algorithm receives no parameters apart from the number of subsequences it uses, for which the authors of [38] suggest an optimal value, $\chi = \left(\frac{n}{B}\right)^{2/3}$, which we employ. We utilize the pruning technique with this algorithm too, hence the denotation 2 in its name.
- **AHistL- Δ** The algorithm suggested by Guha et al. as the best choice for fast offline approximate histogram construction [10] (see Section 2.3). We have used two versions of AHistL- Δ , one for $\epsilon = 0.01$, and one for $\epsilon = 10$, in order to witness how AHistL- Δ resolves the quality-efficiency tradeoff when putting a premium on quality (former case) or efficiency (latter case). In the figures, we indicate the employed value of ϵ as a *percentage* value next to the legend name. Thus, AHistL1 denotes the variant of AHistL- Δ for which $\epsilon = 0.01$.
- **MaxDiff** The relatively elaborate early histogram heuristic which was seen as “probably the histogram of choice” by Poosala et al. [35] (see Section 2.4).
- **MHIST** The one-dimensional adaptation suggested by Jagadish et al. [20] for the multidimensional heuristic proposed by Poosala and Ioannidis [34], and also mentioned by Terzi and Tsaparas by the name Top-Down [38] (see Section 2.4).
- **GDY** Our simple greedy algorithm that iteratively moves a boundary from one position to another until it reaches a local optimum (see Section 4.1).
- **GDY_DP** Our enhanced algorithm that combines a greedy collection of sample boundaries and a DP-based selection of the optimal subset of B out of those samples (see Section 4.2). In the figures, we indicate the number of iterations of GDY that generates the samples for GDY_DP in its legend name; thus, for example, GDY_10DP denotes a variant of GDY_DP that operates with samples generated by 10 runs of GDY.
- **GDY_BDP** The variant of our enhanced greedy algorithm that performs the DP-based selection of optimal boundary-subsets in a *batched* manner, in order to maintain the complexity of the DP operation linear in n (see Section 4.3). As for GDY_DP, the number of runs of GDY that generates the employed samples is indicated in the legend name.

Name	Size	Provenance
Balloon	2001 x 2	http://lib.stat.cmu.edu/datasets/
Darwin	1400 x 1	http://www.stat.duke.edu/~mw/ts_data_sets.html
Exrates	2567 x 12	http://www.stat.duke.edu/data-sets/mw/ts_data/all_exrates.html
Phone	1708 x 8	http://www.teco.edu/tea/datasets/phone1.xls
Shuttle	1000 x 6	http://www-aig.jpl.nasa.gov/public/mls/time-series/
Winding	2500 x 7	http://www.esat.kuleuven.ac.be/~tokka/daisydata.html
DJIA16K	16384 x 1	http://lib.stat.cmu.edu/datasets/djdc0093 (filtered)
Synthetic	100001 x 10	http://kdd.ics.uci.edu/databases/synthetic/synthetic.html

Table 2: Used data sets

All algorithms were implemented with gcc 4.3.0, and experiments were run on a 2 Quad CPU Intel Core 2.4GHz machine with 4GB of main memory running a 64Bit version of Fedora 9. Table 1 summarizes the complexity requirements of these algorithms.

5.2 The Data Sets

Our quality assessment uses several real-world time series data sets. Some of these data are also used in [38]. Table 2 presents the original provenance¹ of the data. We have created *aggregated* versions of these data sets, i.e. concatenated the time series in them to create a united, longer sequence. In the following figures, we denote these aggregated versions by the appellation “-a” in the captioned data set names.

5.3 Quality Comparison

We first direct our attention to a comparison of quality, as measured by the Euclidean error achieved as a function of the available space budget B . In the following figures, the dotted line presents the position of 10% off the optimal error (always achieved by V-Optimal), while the dash-dotted line is the position of 20% off the optimal.

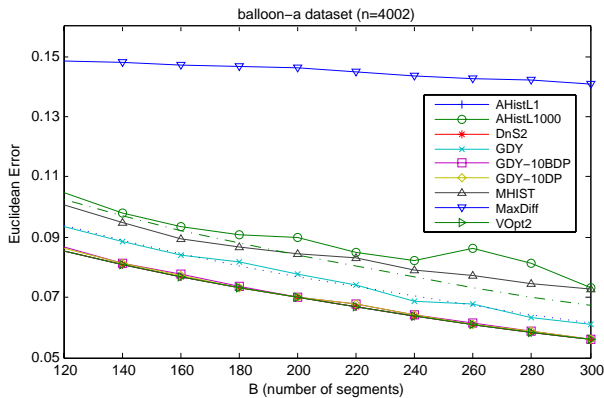


Figure 4: Quality comparison: Balloon

Figure 4 presents the results with the *aggregated* balloon data set for a selected range of $B = 120 \dots 300$. We observe that all of DnS, AHistL- Δ for $\epsilon = 0.01$, GDY_DP, and GDY_BDP achieve practically indistinguishable near-optimal error. There are four outliers. The performance of GDY lies reliably along the 10%-off-optimal line; this is the best-performing outlier. The second outlier is MHIST; as expected, it does not produce histograms of near-optimal quality. Still, the variant of AHistL- Δ for $\epsilon = 10$ is even worse. This result is significant from the point of view of the tradeoff between quality and time-efficiency that AHistL- Δ achieves. We shall come back to it later. The MaxDiff heuristic had the worst performance.

¹The data are also available at <http://felix-halim.net/histogram/>.

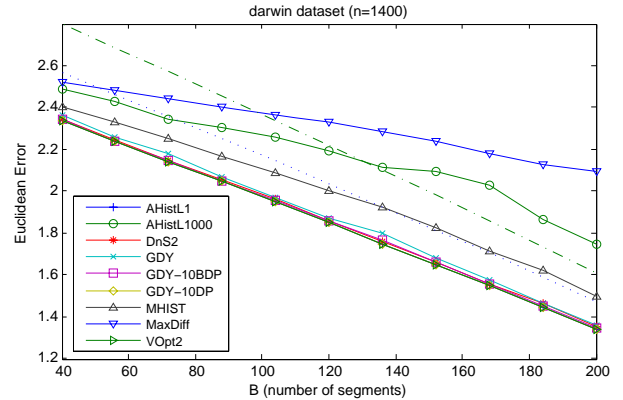


Figure 5: Quality comparison: Darwin

Figure 5 shows the error results with the *darwin* data set for $B = 40 \dots 200$. The picture exhibits a pattern similar to the previous one. Still, with this easier to approximate data set, GDY approaches the quality of the other near-optimal algorithms. Furthermore, the quality of MHIST now follows the 10%-off line. Now the MaxDiff heuristic achieves a smaller accuracy gap from the other contenders, but still has the largest gap. The low-quality version of AHistL- Δ is again an outlier with unreliable performance. The performance with other values of ϵ was falling in between these two extremes. Naturally, the value of ϵ can be tuned so as to allow for quality that matches any of the other algorithms. We do not present these versions in order to preserve the readability of the graph.

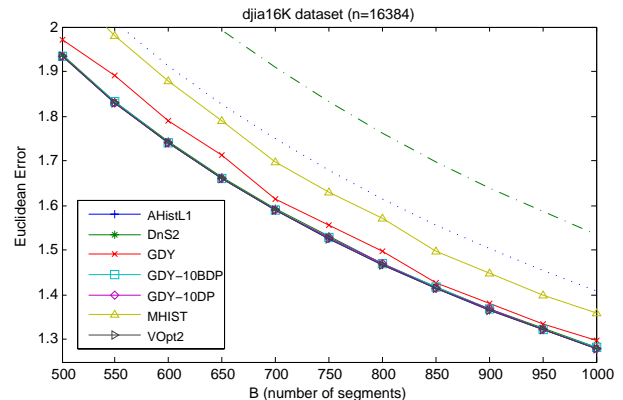


Figure 6: Quality comparison: DJIA

In Figure 6 we present the results for the *filtered* version of the DJIA data set, for a range of $B = 500 \dots 1000$. This version, also used in [9], contains the first 16384 closing values of the Dow-Jones Industrial Average index from 1900

to 1993; a few negative values were removed. The performance evaluation with this data set follows the same pattern as before. In this case, neither MaxDiff nor the low-quality version of AHistL- Δ are depicted, as they are outliers. On the other hand, the MHIST heuristic performs slightly better than it did in previous cases. Still, our GDY algorithm performs even better, while the performance of both GDY_DP and GDY_BDP is almost indistinguishable from that of both high-performing approximation algorithms in this scale.

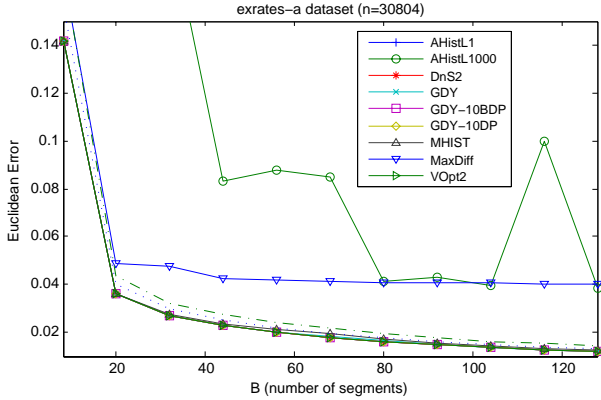


Figure 7: Quality comparison: Exrates

Figure 7 depicts the quality results with the aggregated exrates data set for $B = 8 \dots 128$. This range of B at the lower values of the domain presents an interesting picture. Not only our advanced greedy techniques, but also the simple GDY can achieve error very close to the optimal. So does the MHIST heuristic as well, which follows at a very close distance. Still, MaxDiff and the low-quality version of AHistL- Δ remain low-performing outliers.

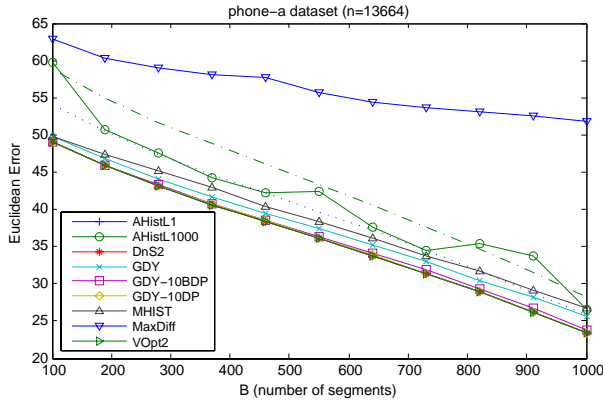


Figure 8: Quality comparison: phone

Next, we depict the Euclidean error results with the aggregated phone data set (Figure 8). The relative performance of different techniques appears clearly in this figure. GDY_DP can almost match the optimal error at least as well as the tight- ϵ variant of AHistL- Δ and DnS, while GDY_BDP and GDY follow closely behind. MHIST does not perform as well as our algorithms, while the slack- ϵ version of AHistL- Δ and MaxDiff are again poor performers.

So far we have presented quality results in terms of Euclidean error as a function of B , with the range of B zoomed in so as to allow the discernment of subtle differences. Still,

we would also like to get a view of the larger picture: the shape of the $E = f(B)$ function, indicating how the Euclidean error varies over a the full domain of practical B values. We do so with the synthetic data set. This data set appears highly periodic, but never exactly repeats itself.

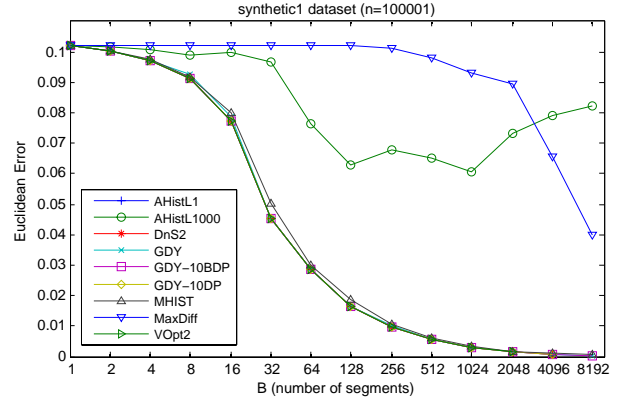


Figure 9: Quality comparison: Synthetic

The results are illustrated in Figure 9, plotting the error for a range of $B = 1 \dots 8192$ in a *logarithmic* x-axis. These results reconfirm our previous micro-scale observation at a larger scale. The performance of the approximation algorithms *as well as* all versions of our greedy approaches closely follow the optimal-error performance. MHIST follows them at a discernible distance. On the other hand, MaxDiff performs poorly, while the slack- ϵ version of AHistL- Δ performs unreliably. The error function is not even monotonic for the low-quality AHistL- Δ ; this defect has also appeared in our earlier graphs.

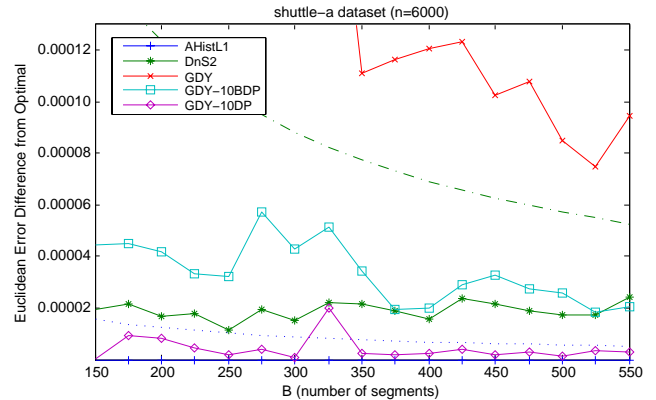


Figure 10: Quality comparison: shuttle

Our hitherto results establish the practical performance of our greedy algorithms in relation to the optimal error. Still, we would like to gain a clear view at a very close level of resolution. Thus we measure the actual *difference* of the Euclidean error achieved with the tested algorithms from the optimal error. Figure 10 presents the results with the shuttle data set, for $B = 150 \dots 550$. The dotted line presents the position of 0.1% off the optimal error (achieved by V-Optimal), while the dash-dotted line traces the position of 1% off the optimal.

What was not clear before becomes apparent in this figure. AHistL- Δ matches the optimal quality, as its ϵ value predisposes it to do. The second-best performance is that

GDY_DP, while DnS and GDY_BDP follow closely after. Our simple GDY algorithm does not achieve error as tightly close to the optimal, while the other heuristics are far-off, and do not fall in this figure. Still, it is remarkable that our heuristics can match and exceed the quality of DnS.

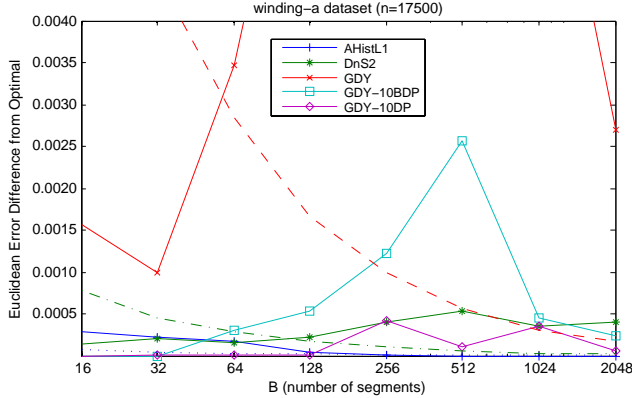


Figure 11: Quality comparison: winding

We elaborate on this line of comparison, presenting the difference from the optimal error with the aggregated winding data set (Figure 11). Now we use a logarithmic x-axis to present a wider range of $B = 16 \dots 2048$. We add a dash-dash line that denotes the position of 10% off the optimal. GDY_DP exceeds the quality of DnS, while GDY_BDP comes close. GDY is more far-off, while other heuristics are distant outliers, not seen in the figure. Thus, the injection of DP capacity into GDY indeed adds a sophistication that affords performance similar or superior to that of the guarantee-providing approximation schemes.

5.4 Runtime Comparison

Now we turn our attention to the other side of the quality-efficiency tradeoff, that of runtime performance. To get a full picture of the runtime state of affairs, we measure runtime as a function of B for constant data set size n , for varying data set size n under constant segmentation size B , as well as for B linearly varying with n .

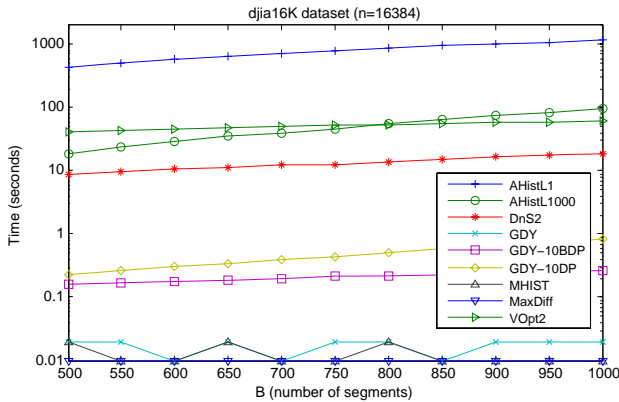


Figure 12: Runtime comparison vs. B : DJIA

Figure 12 plots the results with the *filtered* version of the DJIA data set, for $B = 500 \dots 1000$, same as the one used for quality assessment in Figure 6. The time axis is logarithmic. A comparison of Figures 6 and 12 reveals some interesting findings. If the quality-efficiency tradeoff were equitably resolved by all tested techniques, then we would

expect the good quality performers to be bad runtime performers, and vice versa. Still, the presented picture does not follow such a pattern. On the contrary, some of the best quality performers are also among the best runtime performers as well. That is, remarkably, our GDY_DP and GDY_BDP algorithms, which gave almost optimal quality results, also achieve satisfactorily low and scalable runtime. Moreover, GDY, which achieves next-to-optimal quality performance, is also one of the runtime champions, along with the lower-quality MHIST and the worst-quality MaxDiff heuristic. In contrast, it is clear that other high-quality performers such as AHistL- Δ and DnS pay a high runtime price for the quality they deliver. The same holds for the V-Optimal algorithm itself. Furthermore, the high-quality variant of AHistL- Δ takes runtime even higher than that of V-Optimal; the loose- ϵ variant of AHistL- Δ , which does not perform well on quality, does not gain in runtime from this looseness, and eventually exceeds the runtime of V-Optimal too. The lines in the figure indicate the cubic $O(B^3)$ complexity factor of AHistL- Δ .

Our implementation of V-Optimal, as well as of all algorithms that employ its DP scheme, follows the simple pruning step suggested in [20] (see Section 2.1). It is not clear whether the experimental evaluations in [38] and [10] have used this step. Hence, our runtime results may diverge from those reported in these works.

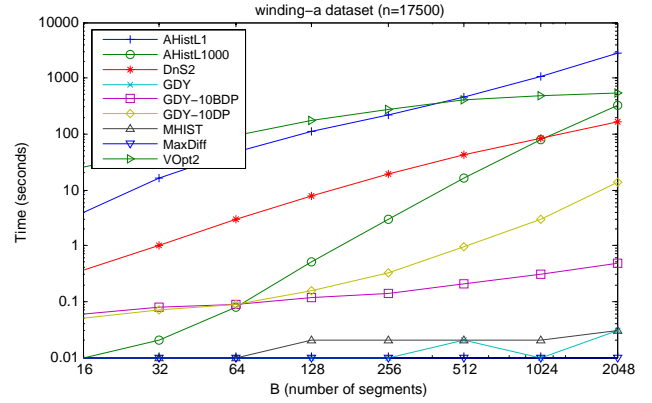


Figure 13: Runtime comparison vs. B : winding

Next we illustrate runtimes with the aggregated winding data set (Figure 13), which present the runtime side of the evaluation for which Figure 11 shows the quality side. Axes are logarithmic, while $B = 16 \dots 2048$. The emerging picture follows a pattern similar to the previous figure. GDY_DP and GDY_BDP achieve a remarkably attractive resolution of the quality-efficiency tradeoff, while GDY is one of the efficiency champions without sacrificing quality as MHIST and MaxDiff do. The cubic growth of AHistL- Δ is clear; both variants eventually exceed the runtime of V-Optimal (which employs pruning), while DnS also pays a high efficiency cost.

Figure 14 displays runtime results with the synthetic data set, i.e., the other side of the quality evaluation depicted in Figure 9, with $B = 1 \dots 8192$ on logarithmic axes. Growth trends are now more accentuated. The growth of DnS, arising from its $O(B^{5/3})$ runtime factor, is also apparent; thus, not only AHistL- Δ , but also DnS exceeds the runtime of V-Optimal for large enough B . GDY_DP also assumes an unfavorable growth trend after the pivot point of $B = \sqrt{n}$ (see Section 4.2). GDY_BDP and GDY stand out as scalable algorithms that also achieve high quality.

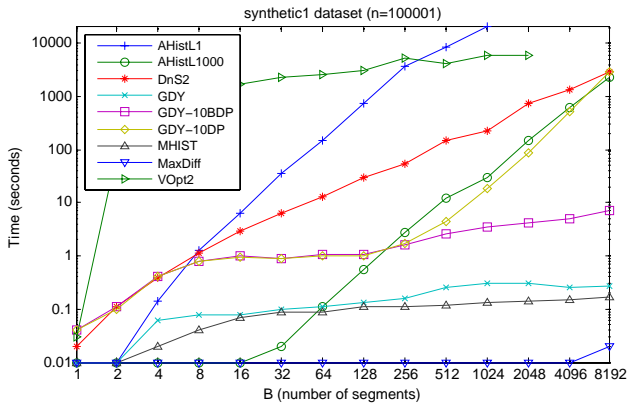


Figure 14: Runtime comparison vs. B : Synthetic

We also evaluate the scalability of different techniques with respect to data set size n . Figure 15 outlines the runtime results for constant $B = 512$, on different prefixes of the same synthetic data set. Both axes are logarithmic. In this case, the growth of AHistL- Δ is not as severe as it was vs. B , but still stands out as the least scalable algorithm, surpassing the runtime of V-Optimal and DnS. We surmise that the $O(B^3 \log^2 n)$ worst-case complexity factor of AHistL- Δ , arising from the burden of approximating the error function itself, works out its impact more saliently as n grows. On the other hand, the impact of pruning within the DP in V-Optimal and DnS allows these algorithms to scale better, although not adequately either. In order to illustrate the impact of pruning, we also include a version of V-Optimal *without* the pruning step in this experiment (labeled VOpt as opposed to VOpt2 in the figure). The non-pruning version exhibits not only higher runtime, but also more accentuated growth with n . The champions of scalability in this experiment are again our greedy algorithms, as well as the heuristics that perform poorly on the quality side.

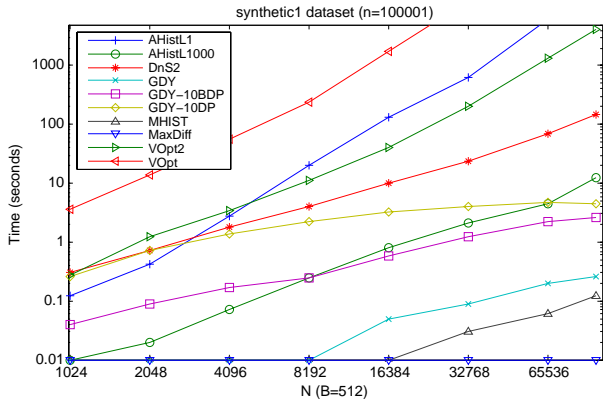


Figure 15: Runtime vs. n , $B = 512$: Synthetic

Finally, we measure the runtime performance with the synthetic data set when n and B grow in parallel. Figure 16 plots the results on logarithmic axes, where $B = \frac{n}{32}$. The unscalable growth of AHistL- Δ is conspicuous; V-Optimal and DnS do not scale well either. GDY-DP almost parallels the growth of the poorly scaling algorithms in this figure. On the other hand, the batch version, GDY-BDP, presents affordable runtime growth in this experiment too; its growth is minimally affected by the growing B , as a comparison of Figures 15 and 16 indicates, and Figure 14 corroborates.

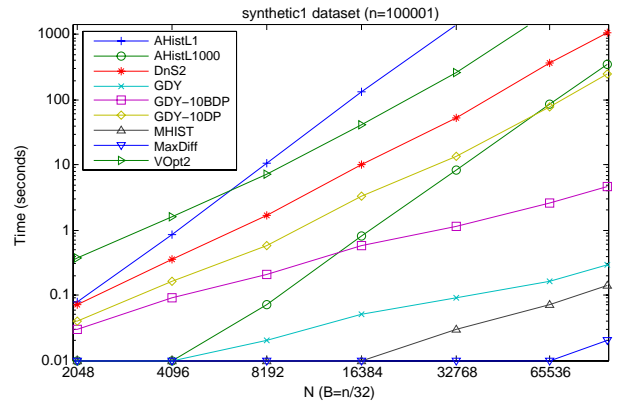


Figure 16: Runtime vs. n , $B = \frac{n}{32}$: Synthetic

Similar observations hold for GDY, MHIST and MaxDiff. Still, given their respective quality performance, GDY-BDP emerges from our assessment as the algorithm of choice.

5.5 Delineating the Tradeoff

In the previous sections we have measured the performance of the examined algorithms in both quality and runtime, and we have inferred that some algorithms resolve this tradeoff in a more satisfactory manner than others. In particular, we have argued that algorithms like AHistL- Δ do not address this tradeoff in an attractive manner; that is, a benefit in quality comes at a high cost of runtime, while a reduction of runtime requires a severe sacrifice in quality. Still, we would like to trace this tradeoff more concretely. In this section we plot both the quality and runtime performance of examined algorithms on the same graph.

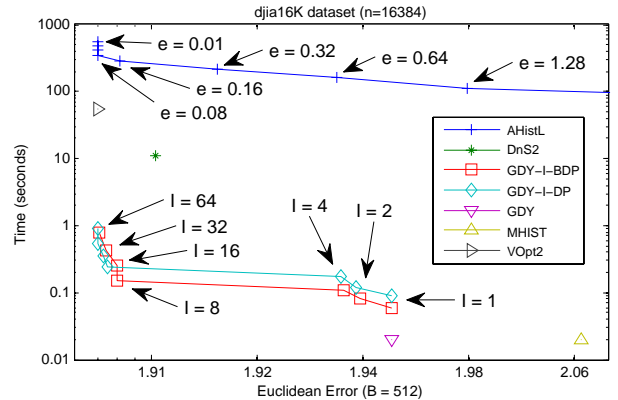


Figure 17: Tradeoff Delineation, $B = 512$: DJIA

Figure 17 traces the quality-efficiency tradeoff with the DJIA data set and $B = 512$. The runtime axis is logarithmic, so that small runtime increases in the lower part of the y-axis are rendered noticeable. Single-version techniques are represented by a single dot. For AHistL- Δ , each variant for a different value of ϵ gets its own dot. Lower values of ϵ allow for higher accuracy at the price of extra runtime. Likewise, several variants of GDY-DP and GDY-BDP are presented, based on the number I of iterations of GDY that they use for collecting sample boundaries. More available samples enable these algorithms to achieve higher quality at the cost of efficiency. For $I = 1$ the histogram given by both these schemes is reduced to that of GDY, since one iteration produces only B samples to choose from.

This figure reconfirms that AHistL- Δ performs poorly at resolving the quality-efficiency tradeoff. An attempt to gain quality by lowering ϵ renders the runtime higher than that of V-Optimal; an effort to improve time-efficiency by increasing ϵ is not effective in its objective, while it deteriorates quality. DnS dominates almost all versions of AHistL- Δ in runtime and quality. In contrast, GDY_DP and GDY_BDP resolve the tradeoff attractively, while they can improve on quality by investing the extra time required by higher values of I .

6. DISCUSSION

Our experimental study has led to some remarkable findings. First, despite their elegance, approximation schemes for histogram construction do not achieve an attractive resolution of the tradeoff between efficiency and quality. Among the proposed schemes, DnS achieves a slightly better position in the tradeoff than AHistL- Δ . Still, both can be superseded in time efficiency by V-Optimal, whom they are meant to approximate, due to their super-linear dependence on B . Secondly, we have determined that our enhanced heuristics, employing greedy local search *in combination with* a DP segmentation on sample boundaries, address the tradeoff more satisfactorily. Our best performing algorithm, GDY_BDP, consistently achieves near-optimal quality in near-linear time.

7. CONCLUSIONS

In this paper we offer a fresh approach to histogram construction or sequence segmentation that addresses a critical gap in related research. We develop segmentation algorithms that are both fast and effective in quality, as measured in terms of Euclidean error. Our best-performing method is based on an application of greedy local search that generates sample boundaries. These sample boundaries are then used as input points for the dynamic-programming segmentation algorithm that selects an optimal subset among them. Moreover, in a divide-and-conquer fashion, this algorithm processes the candidates in *batches*, so that its time complexity is kept constrained. Besides, we have conducted the *first*, to our knowledge, experimental comparison of proposed heuristics and approximation schemes for sequence segmentation. This study shows that our mixed approach achieves near-optimal quality in near-linear runtime. Thus, it outperforms previously proposed heuristics in quality and recently suggested approximation schemes in time efficiency. In conclusion, our solutions provide a highly recommendable choice for all areas where segmentation or histogram construction finds application. In the future we intend to extend our schemes to the multidimensional case.

8. REFERENCES

- [1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, 1999.
- [2] R. Bellman. On the approximation of curves by line segments using dynamic programming. *Communications of the ACM*, 4(6):284, 1961.
- [3] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *VLDB Journal*, 10(2-3), 2001.
- [4] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *TODS*, 27(2), 2002.
- [5] M. Garofalakis and A. Kumar. Wavelet synopses for general error metrics. *TODS*, 30(4), 2005.
- [6] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. *TODS*, 27(3), 2002.
- [7] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *STOC*, 2002.
- [8] S. Guha. On the space-time of optimal, approximate and streaming algorithms for synopsis construction problems. *VLDB Journal*, 17(6), 2008.
- [9] S. Guha and B. Harb. Approximation algorithms for wavelet transform coding of data streams. *IEEE Transactions on Information Theory*, 54(2):811–830, 2008.
- [10] S. Guha, N. Koudas, and K. Shim. Approximation and streaming algorithms for histogram construction problems. *TODS*, 31(1):396–438, 2006.
- [11] S. Guha, K. Shim, and J. Woo. REHIST: Relative error histogram construction algorithms. In *VLDB*, 2004.
- [12] J. Himberg, K. Korpioho, H. Mannila, J. Tikanmäki, and H. Toivonen. Time series segmentation for context recognition in mobile devices. In *ICDM*, 2001.
- [13] Y. E. Ioannidis. Universality of serial histograms. In *VLDB*, 1993.
- [14] Y. E. Ioannidis. Approximations in database systems. In *ICDT*, 2003.
- [15] Y. E. Ioannidis. The history of histograms (abridged). In *VLDB*, 2003.
- [16] Y. E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *SIGMOD*, 1995.
- [17] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *VLDB*, 1999.
- [18] H. V. Jagadish. personal communication, 2008.
- [19] H. V. Jagadish, H. Jin, B. C. Ooi, and K.-L. Tan. Global optimization of histograms. In *SIGMOD*, 2001.
- [20] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, 1998.
- [21] P. Karras. Multiplicative synopses for relative-error metrics. In *EDBT*, 2009.
- [22] P. Karras. Optimality and scalability in lattice histogram construction. In *VLDB*, 2009.
- [23] P. Karras and N. Mamoulis. One-pass wavelet synopses for maximum-error metrics. In *VLDB*, 2005.
- [24] P. Karras and N. Mamoulis. The Haar⁺ tree: a refined synopsis data structure. In *ICDE*, 2007.
- [25] P. Karras and N. Mamoulis. Hierarchical synopses with optimal error guarantees. *TODS*, 33(3):1–53, 2008.
- [26] P. Karras and N. Mamoulis. Lattice histograms: a resilient synopsis structure. In *ICDE*, 2008.
- [27] P. Karras, D. Sacharidis, and N. Mamoulis. Exploiting duality in summarization with deterministic guarantees. In *KDD*, 2007.
- [28] R. Kooi. *The Optimization of Queries in Relational Databases*. PhD thesis, 1980.
- [29] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k -Anonymity. In *ICDE*, 2006.
- [30] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *SIGMOD*, 1998.
- [31] M. Muralikrishna and D. J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *SIGMOD*, 1988.
- [32] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *SIGMOD*, 1984.
- [33] V. Poosala, V. Ganti, and Y. E. Ioannidis. Approximate query answering using histograms. *IEEE Data Eng. Bull.*, 22(4):5–14, 1999.
- [34] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*, 1997.
- [35] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *SIGMOD*, 1996.
- [36] F. Reiss, M. Garofalakis, and J. M. Hellerstein. Compact histograms for hierarchical identifiers. In *VLDB*, 2006.
- [37] M. Salmenkivi, J. Kere, and H. Mannila. Genome segmentation using piecewise constant intensity models and reversible jump MCMC. In *ECCB*, 2002.
- [38] E. Terzi and P. Tsaparas. Efficient algorithms for sequence segmentation. In *SIAM SDM*, 2006.
- [39] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD*, 1999.