Sim-Piece: Highly Accurate Piecewise Linear Approximation through Similar Segment Merging

Xenophon Kitsios
Athens University of Economics and Business
Athens, Greece
xkitsios@aueb.gr

Panagiotis Liakos
Athens University of Economics and Business
Athens, Greece
panagiotisliakos@aueb.gr

Katia Papakonstantinopoulou
Athens University of Economics and Business
Athens, Greece
katia@aueb.gr

Yannis Kotidis
Athens University of Economics and Business
Athens, Greece
kotidis@aueb.gr

ABSTRACT

Approximating series of timestamped data points using a sequence of line segments with a maximum error guarantee is a fundamental data compression problem, termed as piecewise linear approximation (PLA). Due to the increasing need to analyze massive collections of time-series data in diverse domains, the problem has recently received significant attention, and recent PLA algorithms that have emerged do help us handle the overwhelming amount of information, at the cost of some precision loss. More specifically, these algorithms entail a trade-off between the maximum precision loss and the space savings achieved. However, advances in the area of lossless compression are undercutting the offerings of PLA techniques in real datasets. In this work, we propose `Sim-Piece`, a novel lossy compression algorithm for time-series data that optimizes the space requirements of representing PLA line segments, by finding the minimum number of groups we can organize these segments into, to represent them jointly. Our experimental evaluation demonstrates that our approach readily outperforms competing techniques, attaining compression ratios with more than twofold improvement on average over what PLA algorithms can offer. This allows for providing significantly higher accuracy with equivalent space requirements. Moreover, our algorithm, due to the simplicity of its merging phase, imposes little overhead while compacting the PLA description, offering a significantly improved trade-off between space and running time. The aforementioned benefits of our approach significantly improve the efficiency in which we can store time-series data, while allowing a tight maximum error in the representation of their values.

PVLDB Reference Format:
Xenophon Kitsios, Panagiotis Liakos, Katia Papakonstantinopoulou, and Yannis Kotidis. Sim-Piece: Highly Accurate Piecewise Linear Approximation through Similar Segment Merging. PVLDB, 16(8): 1910 - 1922, 2023.
doi:10.14778/3594512.3594521

Proceedings of the VLDB Endowment, Vol. 16, No. 8 ISSN 2150-8097.
doi:10.14778/3594512.3594521

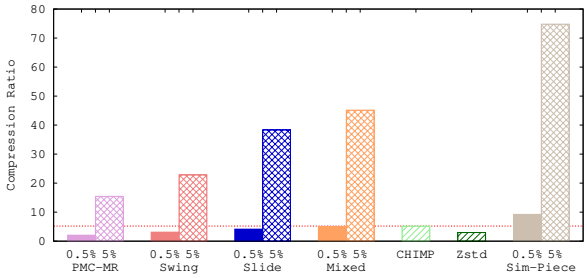


Figure 1: Compression ratio comparison of four lossy PLA approaches for two error thresholds ϵ: a modest 5% and a strict one 0.5% of the dataset's range, against Chimp, the state-of-the-art lossless algorithm, ZStandard, a general purpose compressor, and our novel `Sim-Piece` algorithm.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/xkitsios/sim-piece.

1 INTRODUCTION

Contemporary applications generate massive amounts of streaming data, often depicted as series of timestamped records. In many cases, storing the data is quite challenging due to their volume, and applying compression techniques as a means to reduce the respective storage requirements is deemed necessary. Depending on the problem at hand, one could use either lossless or lossy compression techniques. The former reduce the size of data without loss of information, whereas the latter aim for larger space savings while tolerating a bounded maximum error.

Piecewise linear approximation (PLA) is a fundamental data compression problem dating back to the 1960s [3], commonly used to approximate time-series data. PLA algorithms represent time-series measurements using a sequence of line segments, while keeping the approximation error within a predetermined acceptable threshold. Clearly, these algorithms are associated with a trade-off between space efficiency and precision loss, i.e., the space savings grow with the value of the error threshold.

PLA techniques have been shown to efficiently support the storage of voluminous historical time-series data while also addressing

many data analytics tasks, such as detecting seasonality and forecasting without sacrificing effectiveness [32]. Depending on the selected maximum error tolerance, PLA methods can drastically reduce the size of time-series data achieving compression ratio values that are far beyond the capabilities of state-of-the art lossless compression algorithms. In many applications however, there is certain amount of imprecision in the collected data. For instance sensor temperature measurements often have a $0.1 - 0.5$ degree accuracy. For such applications, we may want to use a tight error threshold with respect to the range of values in order to reduce data size without sacrificing fidelity. Unfortunately, the merits of PLA approaches are not that evident in such a scenario and lossless compression techniques may provide larger space savings.

Figure 1 illustrates the compression ratio of four PLA approaches with two error thresholds ϵ, a modest one equal to 5%, and a strict one equal to 0.5%, of the signal's range (defined as the difference between its maximum and minimum value). Additionally, Figure 1 shows the space requirements of Chimp [21], the state-of-the-art approach for streaming lossless compression, and ZStandard [1], a general purpose compression algorithm targeting real-time scenarios. As we can see, for the smallest value of ϵ (0.05%) the space requirements of the two lossless approaches are comparable or even smaller than those of the PLA approaches.

In this paper we propose `Sim-Piece` that seeks to exploit similarities among PLA-produced line segments. As we see in Figure 1, `Sim-Piece` greatly extends the space-savings of lossy approximations for the same error bounds. Moreover, `Sim-Piece` comes up with *lossy* compressed representations that provide impressive space savings, even in cases where the acceptable error threshold is very small. When considering a line segment, `Sim-Piece` fixes its starting point to a quantized value that is within ϵ of the original value. Then, we add subsequent data points to the segment by maintaining a pair of slopes, i.e., the extreme upper and lower slopes that satisfy the required approximation guarantees, until a point falls out of the area between them. As any of the lines between the two slopes can approximate the data points of the segment, we can find groups of segments with intersecting sets of candidate lines and represent them jointly, to reduce the overall space requirements. The latter process is illustrated in Figure 2. `Sim-Piece` computes the optimal solution to this problem, coming up with the minimum possible number of groups to induce impressive compression ratios.

The performance gains we achieve over a set of several real-world datasets are evident, regardless of whether we seek to maximize the space savings or to achieve high accuracy by setting a large or small value of ϵ, respectively. More specifically, we show that our improvement over the best known PLA approaches in terms of space requirements [10, 24, 28], is consistent as ϵ grows, for values of ϵ that reach 30% of the dataset's range of values. In cases where accuracy is more important, we show that we achieve equivalent compression ratio with the second best approach while setting ϵ below 37% of the threshold that this approach uses on average. This is particularly important as the space requirements of earlier approaches were no longer competitive with lossless compression for small values of ϵ [21]. Finally, we investigate the execution time of `Sim-Piece` against previous approaches and show that we outperform competitive approaches in terms of speed as well, being on average 55× and 6× faster than the second and third best

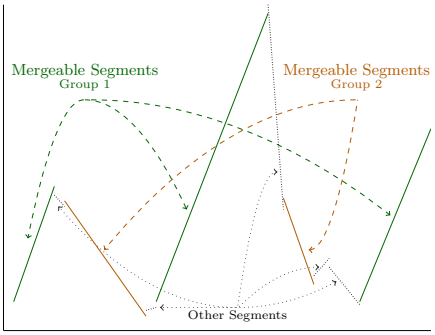


Figure 2: Groups of segments that we can represent jointly with `Sim-Piece`.

approaches with regard to space efficiency, respectively. In fact, the running time of `Sim-Piece` is comparable to that of Swing algorithm [10], which however offers very modest space savings.

We summarize here the key contributions of our proposed compression algorithm. In particular, we:

- propose `Sim-Piece`, a novel lossy compression algorithm that significantly reduces the space requirements of PLA approximations by identifying similar segments and merging their descriptions. Our space gains allow for significantly increasing the accuracy we can obtain, while offering better compression ratio than lossless approaches.
- show that our solution is optimal in terms of always finding the minimum number of groups of segments that can be represented jointly.
- demonstrate that by operating on independent groups of PLA segments, `Sim-Piece` is very efficient in terms of its running time.
- achieve better space-efficiency than what earlier approaches can offer even when combined with general purpose compression. Moreover, compressing the results of `Sim-Piece` allows for providing even more impressive space savings.

2 PRELIMINARIES

Let us now outline the background that will be needed in order to follow and understand our approach.

2.1 PLA methods with Error Threshold ϵ

PLA techniques read as input a potentially infinite stream of timestamped values $(t_i, v_i)_{i \geq 0}$ and generate a number of line segments. For the setting we consider, these lines approximate the original values within a maximum error tolerance ϵ selected by the application.

Key to PLA segmentation is to derive the location and type of the knots [29, 31]. There are methods that enforce continuity conditions [29] at the knots (joint knots [10, 14, 16]), methods that allow discontinuity [29] (disjoint knots [10, 28, 31]) and techniques that actually consider both [24].

As mentioned in [29] the continuity requirement at the knots increases the complexity of the problem. Our segment matching algorithm can be adapted to work upon the output of an existing

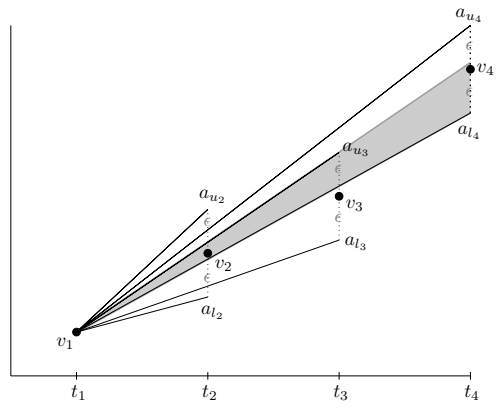


Figure 3: Angle-based PLA. The gray area denotes all possible line segments starting from (t_1, v_1) that approximate the original values v_i within ϵ [10, 28].

PLA segmentation (with joint or disjoint knots) by i) adjusting the value at the starting knot, ii) reading the existing values and calculating upper and lower slopes of admissible approximation lines, for a given maximum error threshold, and iii) in the case of joint knots, trimming the last point of the segment, making it essentially disjoint [29].

For ease of exposition in the next section, we initiate the discussion of our algorithm via a pre-processing step ($\texttt{Sim-Piece}_{phase1}$) using a greedy PLA algorithm, which is an adaptation of Swing [10] that uses disjoint instead of joint knots. Our segment matching algorithm operates directly on the output of this greedy algorithm without further modifications. In our exposition, a segment is described by i) the timestamp t_i and value v_i of its starting point, and ii) the slope a_i of its line.

2.2 Angle-based Greedy PLA

A common approach for constructing an approximate segment with a maximum error guarantee is to calculate extreme slope lines while adding new points. These lines help evaluate whether a new point can be approximated by the current segment, or is a *break-up point* that will trigger the creation of a new segment.

Figure 3 illustrates a process that follows this approach to approximate signal $< (t_1, v_1), (t_2, v_2), (t_3, v_3), (t_4, v_4) >$ using a fixed origin. This origin is set to be (t_1, v_1), i.e., the first point of the signal. When adding (t_2, v_2), the process of Figure 3 creates an *angle* formed by the bounding slope lines a_{u_2} and a_{l_2} connecting (t_1, v_1) with $(t_2, v_2 + \epsilon)$ and $(t_2, v_2 - \epsilon)$, respectively. This angle specifies all lines that may approximate the two points within error threshold ϵ. The next point, i.e., (t_3, v_3) is more than ϵ away from both the upper and lower slopes. Therefore, when adding this point we need to reduce the angle so that the new slopes a_{u_3} and a_{l_3} pass through $(t_3, v_3 + \epsilon)$ and $(t_3, v_3 - \epsilon)$, respectively. Finally, adding (t_4, v_4) requires that the angle is further reduced, as the approximations provided by a_{l_3} are more than ϵ away from (t_4, v_4). Thus, a_{l_4} connecting (t_1, v_1) with $(t_4, v_4 - \epsilon)$ is set as the new lower slope. The upper slope is not updated, as the approximations provided by a_{u_3} are less than ϵ away from (t_4, v_4). The grey area of Figure 3 bounded by a_{u_3} and a_{l_4} illustrates the final candidate lines that are less than ϵ away from all the points of the signal. If the distance of a point encountered was not within ϵ from either of the two slopes, it would trigger the creation of a new segment.

There has been extensive work following an approach such as the one described above, with the resulting segments being joint (Swing [10]), disjoint (Slide [10]) or both (Mixed [24]). In the case of disjoint knots, the break up point is set as the origin of a new segment, whereas in the case of joint knots the immediately previous point is used. Our suggested approach is an adaptation of Swing that produces disjoint knots with quantized associated values based on a maximum error threshold, and exploits the angle formed by the upper and lower slopes of each segment to represent them as intervals of slope values. As we will explain, these intervals will be used to *merge* the descriptions of different segments into a compact representation.

2.3 Interval Graphs

The grey area formed by the angle of slopes a_{u_3} and a_{l_4} in Figure 3 illustrates all the candidate lines that we may use to describe the respective segment within the defined accuracy. Thus, we can come up with an *interval* of slope values that captures all possible lines within the grey area. Naturally, the intervals of different segments will often intersect, enabling us to represent them *jointly*. We will next present the theorem that we will use to justify our method, along with the required definitions, to ease the understanding of our approach.

Definition 2.1. Consider a set of intervals $I = \{I_1, I_2, \ldots, I_n\}$ on a line for $n \in \mathbb{N}$, where $I_j = [a_{l_j}, a_{u_j}]$ and $a_{l_j} \leq a_{u_j}$, for $j \in \mathbb{N}$ and $j \leq n$. The interval graph $G = (V, E)$ formed by I contains one vertex v_j for each interval I_j, so $V = \{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$, and there is an edge between vertices v_i and v_j if and only if I_i and I_j intersect, i.e., $E = \{(x_i, x_j) \mid I_i \cap I_j \neq \emptyset\}$.

Definition 2.2. A vertex is called *simplicial* if its neighbors form a clique, i.e., its neighbors are all linked to one another by edges.

Definition 2.3. A *perfect elimination scheme* in a graph with n vertices is an ordering $\mathsf{v}_1, \ldots, \mathsf{v}_n$ of the vertices such that v_i is simplicial in the graph that contains only vertices $\mathsf{v}_i, \ldots, \mathsf{v}_n$.

Interval graphs have the following characterization [13] that we are going to use later on to justify the optimality of our method.

Theorem 1. A graph is an interval graph if, and only if, a perfect elimination scheme exists.

3 OVERVIEW

We now discuss the details of our approach for high-precision storage reduction of time-series data that is based on the idea that the different line segments produced by PLA techniques exhibit *similarities*. We first present a PLA technique that produces sets of candidate lines for each segment, that we call *intervals*. Then, we seek to *merge* intervals that look alike in terms of their slope as well as their starting point, by identifying intersecting intervals.

In what follows, we provide algorithms for both these procedures which constitute the two phases of our novel Sim-Piece approach for highly accurate PLA with small storage footprint.

3.1 Interval Extraction

The first phase of Sim-Piece considers as input a data signal in the form of a sequence of discrete data points (t_i, v_i), where $i \in \{1, \ldots, n\}$. The respective pseudocode is given with Algorithm 1.

We use the first point of the signal as the first point of the starting segment (Line 3). The number of discrete values that the starting points of our segments may take, can be arbitrarily large. However, we need to come up with a limited amount of *discrete* starting point values that multiple different segments may share, so that we are able to *jointly* represent them. In our context, we are interested in providing *approximations* within an error threshold ϵ. Therefore, we do not need to consider *all* the different original values. Instead, we can apply a quantization function such as the following for a value v, to come up with a quantized value b:

$$b = \lfloor v/\epsilon \rfloor \times \epsilon \tag{1}$$

As an example, values 1.1 and 1.4 would both result in $b = 1$ for $\epsilon = 0.5$. Using Eq. (1) we convert the first value v_s to the smallest multiple of ϵ that is within ϵ from the original value v_s (Line 4). We also initialize the upper (Line 5) and lower (Line 6) slopes forming the angle of the segment with the maximum and minimum possible values, respectively, so that we make sure the second point of the segment lies within the angle. Then we proceed by processing subsequent points in the signal, following the procedure discussed in Section 2.2 and depicted in Fig. 3. More specifically, we examine whether the distance of each point encountered is *not* within ϵ from the angle formed by the bounding lines with slopes a_u and a_l (Line 9). If so, we terminate the formation of the current segment by producing an interval for starting point b denoted by the final lower and upper slopes a_l and a_u and the initial timestamp t_s (Line 10), and we repeat the process considering the newly encountered point, i.e., (t_c, v_c), as the starting point of the next segment. Otherwise, depending on the position of the newly encountered point, we may need to update the angle by lowering the upper bounding slope or increasing the lower bounding slope. We achieve this by properly adjusting values a_u (Lines 15- 16) and a_l (Lines 17- 18), respectively.

Following this procedure for all points of the signal, we come up with a list of intervals associated with each quantized b value encountered in the signal. The elements of the final lists comprise the upper and lower slope of each interval, which form the angle of each respective segment as illustrated in Fig. 3, as well as the initial timestamp of each segment.

Algorithm 1: Sim-Piece$_{phase1}$

```
   Input: A data signal s: (t_i, v_i) ∀i ∈ {1,...,n}
   Output: A list of intervals b_intervals for each quantized b value
1  Function Sim-Piece_phase1 (s)
2      b_intervals ← {};
3      (t_s, v_s) ← s.next();
4      b ← ⌊v_s/ε⌋ε;
5      a_u ← ∞;
6      a_l ← −∞;
7      while s.hasNext() do
8          (t_c, v_c) ← s.next();
9          if v_c > a_u(t_c − t_s) + b + ε or v_c < a_l(t_c − t_s) + b − ε then
10             b_intervals[b].add(⟨a_l, a_u, t_s⟩);
11             (t_s, v_s) ← (t_c, v_c);
12             b ← ⌊v_s/ε⌋ε;
13             a_u ← ∞;
14             a_l ← −∞;
15         if v_c < a_u(t_c − t_s) + b − ε then
16             a_u ← (v_c + ε − b)/(t_c − t_s);
17         if v_c > a_l(t_c − t_s) + b + ε then
18             a_l ← (v_c − ε − b)/(t_c − t_s);
19     b_intervals[b].add(⟨a_l, a_u, t_c⟩);
20     return b_intervals;
```

3.2 Merging Lists of Intervals

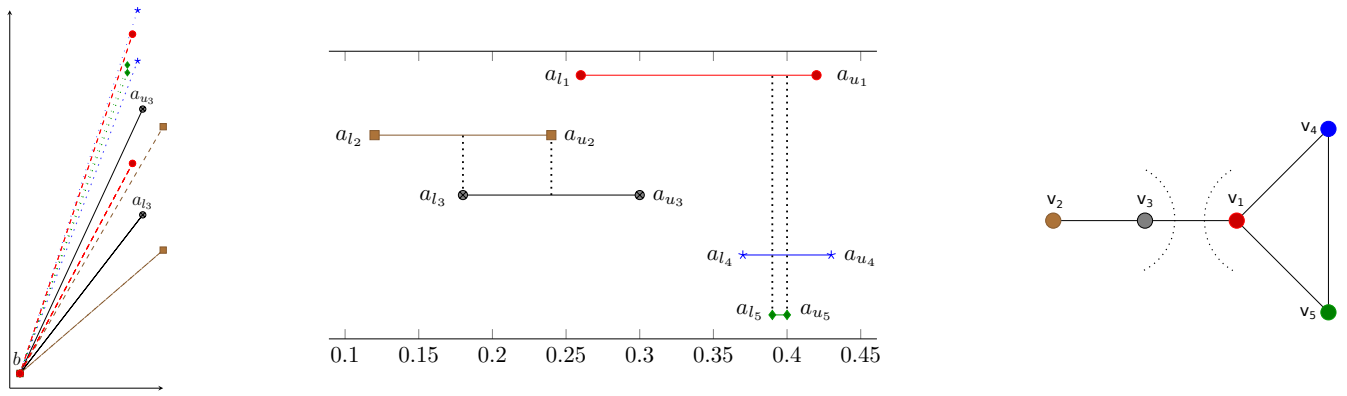
Using the intervals extracted through Algorithm 1, we aim to find the optimal groups of intervals for each quantized b value, so that we can represent similar intervals jointly and induce space savings.

Figure 4a depicts the angles of five intervals that share a common starting point b. We observe that the angles formed by the intervals may overlap, and thus, there exist candidate lines that may represent more than one intervals. The flexibility of using any of the candidate lines of each interval –as they all satisfy the required approximation guarantees– enables us to group different overlapping intervals to minimize the space requirements of their representation. More specifically, our goal is to come up with the minimum number of groups of intersecting intervals for each starting point b. A formal description of this problem follows.

Problem description 1. Given a set of intervals $I = \{I_1, I_2, \ldots, I_n\}$ as described in Definition 2.1, where a_{l_j} and a_{u_j} denote the minimum and maximum slopes of interval $I_j = [a_{l_j}, a_{u_j}]$, respectively, partition I into disjoint sets, so that all intervals in each set intersect, and the number of partitions is minimized.

In Figure 4b, we plot the sets of candidate lines of Figure 4a as slope intervals, so that the overlapping areas are more evident. We observe that if we choose to group the interval $[a_{l_2}, a_{u_2}]$ with $[a_{l_3}, a_{u_3}]$, and the interval $[a_{l_1}, a_{u_1}]$ with $[a_{l_4}, a_{u_4}]$ and $[a_{l_5}, a_{u_5}]$ we will come up with a total of two groups. However, if we choose to group $[a_{l_1}, a_{u_1}]$ with $[a_{l_3}, a_{u_3}]$, we will come up with three groups in total, as we will form one with intervals $[a_{l_4}, a_{u_4}]$ and $[a_{l_5}, a_{u_5}]$ and one with a single interval $[a_{l_2}, a_{u_2}]$. That is, if we select to merge $[a_{l_1}, a_{u_1}]$ with $[a_{l_3}, a_{u_3}]$, we forfeit the possibility of merging all three $[a_{l_1}, a_{u_1}]$, $[a_{l_4}, a_{u_4}]$ and $[a_{l_5}, a_{u_5}]$ segments together, which forces us to come up with at least three merged groups, instead of just two which is the optimal solution for this example.

The intersections among the different intervals can be represented even more clearly using the interval graph of Figure 4c. An interval $[a_{l_i}, a_{u_i}]$ is represented by vertex v_i, and there exists an edge between two vertices v_i and v_j if the respective $[a_{l_i}, a_{u_i}]$ and $[a_{l_j}, a_{u_j}]$ intervals are overlapping. In this way, the problem of finding the minimum possible number of groups of overlapping intervals becomes equivalent with partitioning the vertices of the interval graph of Figure 4c into the minimum number of groups such that the vertices in each group are all linked to one another by edges. This is a well studied problem, known as the 'minimum covering by disjoint completely connected sets or cliques' [15]. As the graph of Figure 4c is an interval graph, there exists a perfect elimination scheme, which is very easy to determine. We simply need to choose a simplicial vertex, i.e., one whose neighbors are all linked to one another, and place it in the first position of our scheme. We then delete this vertex from the graph and look for a



(a) (b) (c)

Figure 4: Sets of candidate lines for 5 line segments, depicted by the maximum and minimum possible slope for each segment (4a), the respective set of intervals (4b), and the respective interval graph (4c). There is no need to construct the interval graph which is featured here only to ease understanding.

Algorithm 2: `Sim-Piece`$_{phase2}$

Input: A list of intervals *b_intervals* for each quantized b value
Output: A list *groups* containing one group $\langle b, a_l, a_u, t \rangle$ for each quantized b value

```
1  Function Sim-Piece_phase2 (b_intervals)
2      groups ← {};
3      for intervals_{b_i} ∈ b_intervals do
4          group ← ⟨b = b_i, a_l = −∞, a_u = ∞, t = [ ]⟩;
           // sort by ascending a_l order
5          sort(intervals_{b_i});
6          for interval ∈ intervals_{b_i} do
7              if interval.a_l ≤ group.a_u and interval.a_u ≥ group.a_l then
8                  group.a_u ← min(group.a_u, interval.a_u);
9                  group.a_l ← max(group.a_l, interval.a_l);
10                 group.t.add(interval.t);
11             else
12                 groups.add(group);
13                 group ←
                     ⟨b = b_i, a_l = interval.a_l, a_u = interval.a_u, t = [interval.t]⟩;
14         groups.add(group);
15     return groups;
```

simplicial vertex in the remaining graph, which we place in the second position of our scheme and also delete from the graph. We continue doing this until the remaining graph is empty. Indeed, we can see that vertex v_2 is simplicial, as it has only one neighbor, so we can choose it for the first position of the elimination scheme. After deleting v_2, we can choose v_3, which is also simplicial in the remaining graph, as it has only one neighbor left, i.e., v_1. Then, we can choose v_1, as its two remaining neighbors, i.e., v_4 and v_5 are all linked to one another. Next, we can choose v_4 and finally v_5. The final order of the perfect elimination scheme described above is:

$$v_2 \rightarrow v_3 \rightarrow v_1 \rightarrow v_4 \rightarrow v_5$$

An optimal algorithm for coming up with a perfect elimination scheme is to sort the intervals in ascending order of the lower point of their interval, a_{l_i} [15]. As we can see in Figure 4b, this order of the intervals matches the order of the perfect elimination scheme given above. Therefore, there is no need to construct the actual interval graph. We can simply follow the procedure described in Algorithm 2. We first initialize an empty list of groups to be produced (Line 2). Then, we iterate over the list of intervals of every b (Line 3), initialize the first group (Line 4) and order the respective intervals in ascending value of the lower point of their interval, a_{l_i} (Line 5). We go through the intervals in this order (Line 6) and consider them for placement in the current group. If the bounds of this group overlap with the bounds of the currently considered interval (Line 7), we add the interval to the group, by adjusting the bounds of the group accordingly and adding the timestamp of the interval to the group's list of timestamps (Lines 8-10). Otherwise, we close the group, placing it to the list of groups to be returned at the end of Algorithm 2 (Line 12), and place the interval in question in a new group (Line 13).

3.3 Output of `Sim-Piece`

The output of Algorithm 2 is a list of groups that comprise the quantized value of b, the upper and lower bounds a_u and a_l, and the starting-point timestamps of the merged intervals of the group. As we can use any of the lines within the area defined by the upper and lower bounds, we choose to use the line in the middle of this area, with $a = \frac{a_u + a_l}{2}$. The final output of `Sim-Piece` uses the following compact representation:

$$output = [b_1, [a_{1,1}, [t_{1,1}, \ldots]], \ldots], [b_2, [a_{2,1}, [t_{2,1}, \ldots]], \ldots], \ldots$$

It is evident that the proposed representation does not alter the original PLA segmentation, i.e., the position of knots, as denoted by the included timestamps. Moreover, by construction, the maximum error threshold guarantee still holds for the entire history of the time-series.

3.4 Analysis of `Sim-Piece`

The correctness and optimality of our segment merging approach is guaranteed by the perfect elimination scheme property for interval graphs (Theorem 1). According to this property, which serves as the greedy choice of Algorithm 2, there is a certain order to examine

Table 1: Details about our time-series datasets, including the number of measurements (length), the minimum value, the number of decimal places, the range, the median value, the standard deviation, the mean delta and the probability of a data point to be higher ($p_\uparrow$), equal ($p_=$) or lower ($p_\downarrow$) than that of the previous data point.

	Dataset	Length	Min Value	Decimal Places	Range (×0.5%)	Median	σ	$p_\uparrow - p_= - p_\downarrow$	Mean Delta
UCR	Cricket	702,000	-10.19918800	8	22.9 (0.115)	-0.041	0.9	49% − 0% − 51%	0.03
	FaceFour	39,200	-4.68758570	8	10.5 (0.053)	-0.097	0.9	26% − 47% − 27%	0.03
	Lightning	122,694	-1.78116300	8	24.9 (0.125)	-0.235	0.9	41% − 17% − 42%	0.04
	MoteStrain	106,848	-8.63799570	8	17.1 (0.086)	-0.003	0.9	55% − 1% − 44%	0.08
	Wafer	1,088.928	-3.0539799	7	15.1 (0.076)	0.281	0.9	25% − 50% − 25%	0.05
NEON	Wind Speed	4,119,081	0.00	2	20.3 (0.102)	1.380	1.9	47% − 8% − 45%	0.09
	Wind Dir.	1,169,510	0.00	2	360 (1.800)	186.850	107.1	50% − 0% − 50%	22.96
	Pressure	12,098,677	90.99386	5	13 (0.065)	113.079	3.2	9% − 82% − 9%	0.000004

the vertices, i.e., by ascending value of the lower point of their corresponding interval, so that each group formed contains the maximum possible number of vertices and no group is created unless it is really necessary. Therefore we come up with the minimum possible number of intervals.

Algorithm 1 groups the n input time-series values in batches of intervals, and then Algorithm 2 sorts the batches of intervals that share the same starting point (quantized b value) and merges them. For each interval produced by Algorithm 1, we store exactly 4 elements: the a_{l_i} and a_{u_i} that define the slope of the interval, as well as the t_i and v_i that comprise the timestamped value. Hence the processing of each data point of the time series requires a constant amount of memory, and the number of intervals produced cannot exceed n, hence $O(n)$ space is needed. Moreover, as Algorithm 2 merges intervals, the space needed can only reduce, progressively. Therefore, the space complexity of our approach is $O(n)$.

Regarding the time complexity of the end-to-end process, Algorithm 1 is $O(n)$ time, since it processes the input values sequentially. Assume now that the number of intervals produced by Algorithm 1 is k. In the worst case with respect to running time, all intervals may be placed in the same batch, and the sorting step of Algorithm 2 will be computed in $O(k \log k)$ time. In practice, input segments are dispersed among multiple b values based on their starting points, resulting in even faster execution. The scanning and merging of the sorted lists requires $O(k)$ time.

4 EXPERIMENTAL RESULTS

We implemented our `Sim-Piece` algorithm using Java and tested its performance against four PLA algorithms and one lossless floating point data compression method. Our implementation as well as reproducible tests are publicly available.[1] In this section we first present the dataset and technical details on our experiments. Then, we evaluate our algorithm by answering the following questions: i) What is the compression ratio that `Sim-Piece` achieves compared to earlier approaches? ii) What is the highest level of accuracy that `Sim-Piece` may offer when we cannot afford to employ lossless compression? iii) How does `Sim-Piece` perform as we vary the desired error threshold? iv) How do monotonicity and seasonality impact `Sim-Piece`? v) Can we induce further savings through general purpose compression? vi) For the same maximum error threshold, is `Sim-Piece` as fast as other PLA algorithms?

4.1 Experimental Setting

We ran our experiments on a computer with an Intel® Core™ i7-10510U, with a Max Turbo Frequency of 4.90GHz, a 8MB L3 cache, and a total of 16GB DDR3 2133MHz RAM. We implemented `Sim-Piece`, PMC-MR [20] and Swing [10] using Java and we used a C++ implementation for Slide [10], and Mixed [24] PLA algorithms, which the authors of [24] have generously provided. We note that based on the observations made by the authors of [24], Slide computes the minimum number of segments with disjoint knots for a maximum error bound and is thus optimal space-wise in this setting [28]. Moreover, Mixed has been shown [24] to provide even better savings, as it further considers joint knots.

Our dataset consists of 8 time-series from two sources. The first has been used in prior efforts [24, 34], and features datasets from the UCR time-series data archive [2]. We also use 3 time series that exhibit different characteristics in terms of standard deviation and mean delta (absolute difference between consecutive values), and are made available by the National Ecological Observatory Network (NEON)[2] The properties of the dataset are listed in Table 1, and the different time-series are thoroughly discussed below:

- UCR Time Series Classification Archive [2]
  - Cricket: Accelerometer data taken from actors performing cricket gestures.
  - FaceFour: Face outlines modelled as time-series data.
  - Lightning: Transient electromagnetic events associated with lightning using a suite of optical and radio-frequency (RF) instruments.
  - MoteStrain: Humidity and temperature sensor data.
  - Wafer: A collection of inline process control measurements recorded from various sensors during the processing of silicon wafers for semiconductor fabrication.
- NEON datasets
  - WindSpeed [26]: Wind speed observations made by a wind monitor consisting of a propeller and vane design on lake and river buoys.

[1] https://github.com/xkitsios/sim-piece

[2] https://data.neonscience.org/data-products/

- WindDirection [26]: Wind direction observations made by a wind monitor consisting of a propeller and vane design on lake and river buoys.
- Pressure [25]: Barometric pressure on the meteorology station on the buoy in lakes and rivers.

We also utilize synthetic datasets in order to examine the effect of time series properties, such as monotonicity and seasonality, on the different PLA algorithms.

4.2 Compression Ratio Comparison

We start our evaluation by measuring the space required to compress each of the 8 datasets. Figure 5 shows the compression ratio achieved by our Sim-Piece compared to Mixed, Slide and Swing PLA algorithms. The results of PMC-MR are omitted as they were clearly inferior compared to that of the other approaches, as illustrated in Figure 1. We report the compression ratio for varying error threshold ϵ values, expressed as a percentage of the total range of each dataset, listed in Table 1. More specifically, we report results for $0.5\% \times range \leq \epsilon \leq 5\% \times range$. We also provide for every dataset the compression ratio achieved by the lossless Chimp compression algorithm. We see in Figure 5 that our Sim-Piece clearly stands out as the most space-efficient algorithm, offering compression ratios that are far superior than the second best approach. The improved performance of Sim-Piece over earlier PLA approaches is evident for all values of ϵ investigated. The only exception is the *Pressure* dataset for which all approaches manage to offer unusually large compression ratios. This is due to the very high probability of identical consecutive values and extremely small mean delta this dataset exhibits, as reported in Table 1. Thus, we additionally investigate the performance of all algorithms for smaller values of ϵ for this dataset, in the last plot of Figure 5. As we can see there, Sim-Piece again outperforms all earlier approaches by a large margin.

The horizontal red dotted lines in the plots of Figure 5 is the compression ratio achieved by the Chimp compression algorithm. We see that in many cases a large value of ϵ is required for the use of earlier PLA algorithms to make sense, as the recent advances in the field of floating point compression offer very competitive space requirements. Table 2 reports for every time series of our dataset the smallest value of ϵ for which Sim-Piece and the second best approach produce an equivalent compression ratio to that of Chimp. In cases when one cannot satisfy the space requirements of lossless compression and needs to employ a lossy representation, our results clearly show how our Sim-Piece can provide significantly higher accuracy than earlier PLA approaches, thus offering improved space requirements even under very strict error threshold limits.

4.3 Quality of Approximation

Sim-Piece is designed to offer maximum error guarantees to each and every value of the time-series. Thus, it is worth exploring i) how the actual Mean Absolute Error (MAE) compares to the maximum error threshold used, and ii) what is the Root Mean Square Error (RMSE) of the approximation. For the latter we used an unmodified version of Sim-Piece that does not seek to optimize this metric by adjusting the computation of the slope values accordingly.

In Table 3 we report results for all datasets and PLA techniques when the input maximum error threshold ϵ is set at 5%. We also report MAE as a fraction of the dataset range (MAEr%), in order to have a comparison of the obtained approximation with respect to the maximum error threshold ϵ used. We notice that all methods provide approximations that are significantly more accurate than the requested threshold. In fact, the average MAEr% in all algorithms is about half of the ϵ value requested. Moreover, the reported RMSE values are very close to the measured MAE. This implies that there is very small deviation among the errors obtained on the individual measurements. Compared to the other techniques, Sim-Piece provides the second-best average MAE and RMSE values. Swing provides marginally better approximation; however, we need to keep into account that, on the average, Sim-Piece achieves a compression ratio that is more than 3.2 times higher than that of Swing in this setting.

In order to put into perspective the obtained accuracy of each algorithm, we plot, in Figure 6, the compression ratio obtained by the different methods with respect to the MAE of their approximations, for the Wafer dataset. We notice that for the same MAE (x-axis value), Sim-Piece achieves significantly higher compression ratio, compared to earlier PLA algorithms. Figures for other datasets were similar and are omitted for brevity.

Table 2: Sim-Piece and Mixed (2^{nd} best) ϵ values that produce an equivalent compression ratio to that of Chimp.

Dataset		ϵ	
		Sim-Piece	Mixed
UCR	Cricket	0.15%	0.53%
	FaceFour	1.25%	2.19%
	Lightning	0.12%	0.42%
	MoteStrain	0.15%	0.39%
	Wafer	0.05%	0.20%
NEON	Wind Speed	0.43%	2.69%
	Wind Dir.	0.44%	1.84%
	Pressure	0.03%	0.04%

4.4 Impact of Error Threshold

We continue investigating the performance of Sim-Piece by examining the impact of the error threshold ϵ on its compression ratio. For every approach we measure the compression ratio achieved as ϵ grows up to 50% of each dataset's range. Larger values of ϵ are not meaningful as they would enable us to represent the entire signal using a single constant value. The performance is similar for all datasets and thus, for brevity, we illustrate (in logarithmic scale) the results for only two of our datasets in Figure 7.

We see that Sim-Piece maintains its advantage over earlier approaches for large values of ϵ, around 30% of the entire range. Therefore, not only does Sim-Piece provide improved space efficiency when high accuracy is required, but it also outperforms earlier approaches when the error threshold is large. For values of ϵ that are larger than 30% of the dataset's range, only a handful of PLA segments are produced by Angle-based PLA, resulting in fewer opportunities for merging similar segments. As a result, the Mixed and Slide algorithms perform better. However, the accuracy in such settings is clearly very low.

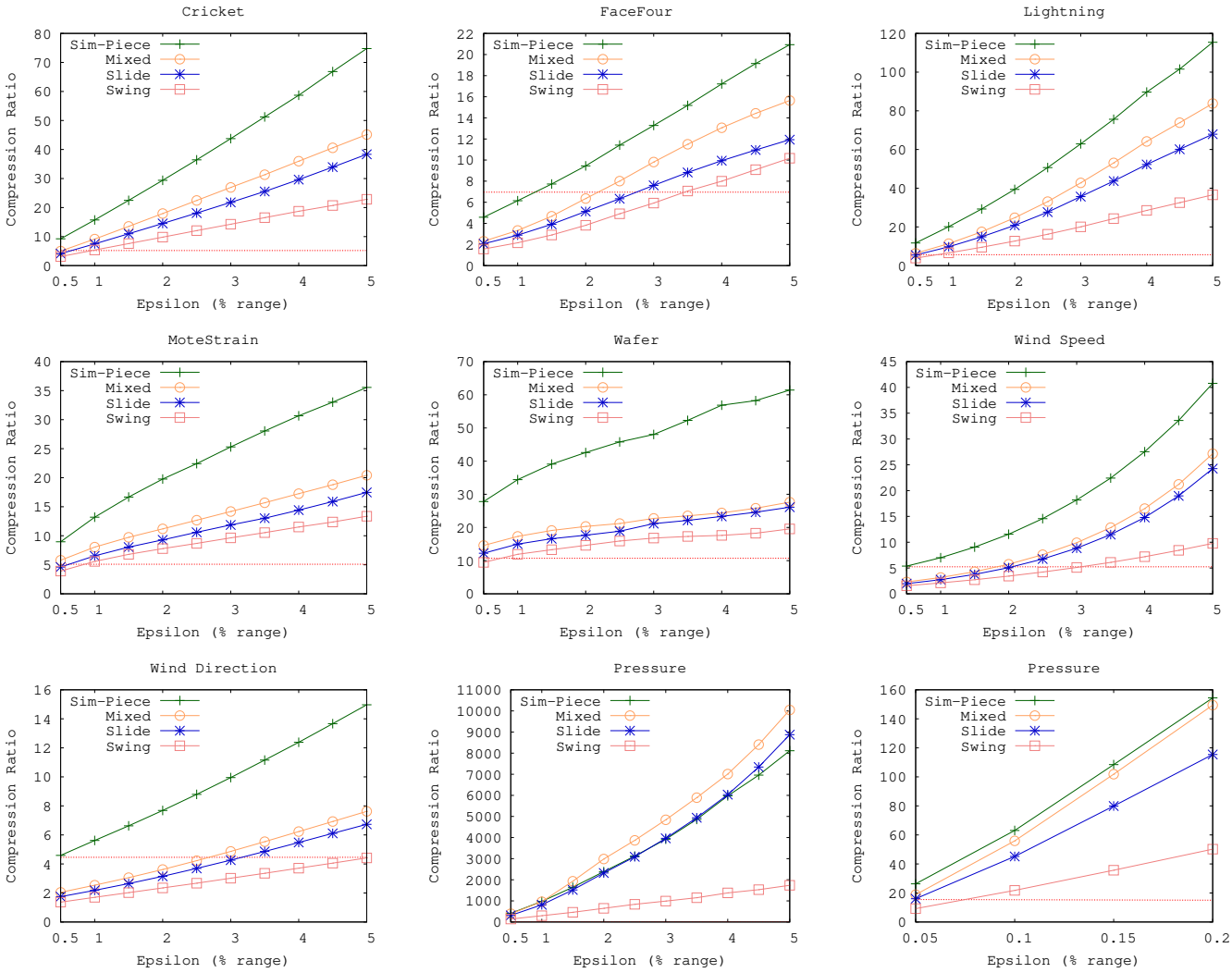


Figure 5: Compression ratio results varying error threshold ϵ.

4.5 Impact of the Degree of Monotonicity

Sim-Piece applies a quantization function to the starting points of segments, so that we come up with a limited amount of discrete starting point values that multiple line segments may share. The number of values we come up with depends on the error threshold as well as the degree of monotonicity the input signal exhibits. Our next experiment investigates the impact the latter has on the performance of Sim-Piece.

We generate a synthetic signal that follows a random-walk-like model. The value for each data point can be lower than or higher than that of the previous data point according to the probabilities p and $(1-p)$, respectively. The magnitude of decrease/increase in the value follows a uniform distribution $U(0, v_{max})$, where v_{max} is set to be up to 300% of the error threshold ϵ. Figure 8 illustrates the impact of the degree of the signal's monotonicity on the compression ratio. The probability p is varied from 0 to 0.5, as the plot is symmetric for higher values. When p is set to 0, the signal is monotonically increasing.

Figure 8 shows that Sim-Piece performs best when the probability is close to 0.5, as is the case with many real-world datasets. Indeed, as we see in Table 1 the probability of a data point to be lower or higher than that of the previous data point is almost identical for all datasets. The compression ratio of Sim-Piece decreases with the probability p; however, Sim-Piece outperforms earlier approaches for $0.45 < p \leq 0.5$. When the probability of decrease p is less than 0.45 the signal exhibits strong upwards trend with very few chances of repeating starting points among segments that Sim-Piece could utilize in the second phase of the algorithm.

In Figure 8 we also present results of Sim-Piece when we perform *detrending* to the input signal. There any many ways to remove trend, we here assume an additive model and simply subtract a linear regression line from the entire signal before applying

Table 3: Compression Ratio (CR), measured MAEr%, MAE and RMSE for $\epsilon = 5\%$.

Dataset		Swing				Slide				Mixed				Sim-Piece			
		CR	MAEr%	MAE	RMSE	CR	MAEr%	MAE	RMSE	CR	MAEr%	MAE	RMSE	CR	MAEr%	MAE	RMSE
UCR	Cricket	22.8	2.48%	0.567	0.657	38.3	2.34%	0.535	0.626	45.2	2.32%	0.532	0.624	74.8	2.21%	0.506	0.597
	FaceFour	10.2	2.49%	0.262	0.306	11.9	2.68%	0.281	0.327	15.6	2.62%	0.275	0.322	20.9	2.46%	0.258	0.302
	Lightning	36.6	2.44%	0.608	0.704	67.9	2.72%	0.677	0.765	83.8	2.47%	0.615	0.707	115.4	2.26%	0.563	0.655
	MoteStrain	13.4	2.50%	0.428	0.503	17.5	2.91%	0.497	0.562	20.4	2.97%	0.507	0.572	35.6	2.71%	0.464	0.528
	Wafer	19.6	2.03%	0.307	0.374	26.1	2.98%	0.449	0.507	27.6	2.87%	0.434	0.434	61.4	2.77%	0.418	0.471
NEON	Wind Speed	9.8	2.63%	0.533	0.619	24.3	2.42%	0.491	0.583	27.2	2.30%	0.466	0.557	40.8	2.30%	0.467	0.560
	Wind Dir.	4.4	2.29%	8.233	10.082	6.7	2.66%	9.576	11.320	7.6	2.64%	9.521	11.267	15.0	2.46%	8.841	10.451
	Pressure	235.7	2.39%	0.310	0.362	534.2	2.10%	0.273	0.324	699.8	3.60%	0.468	4.296	769.4	2.12%	0.276	0.326
Average		44.1	2.41%	1.406	1.701	90.9	2.60%	1.597	1.877	115.9	2.72%	1.602	2.355	141.7	2.41%	1.474	1.736

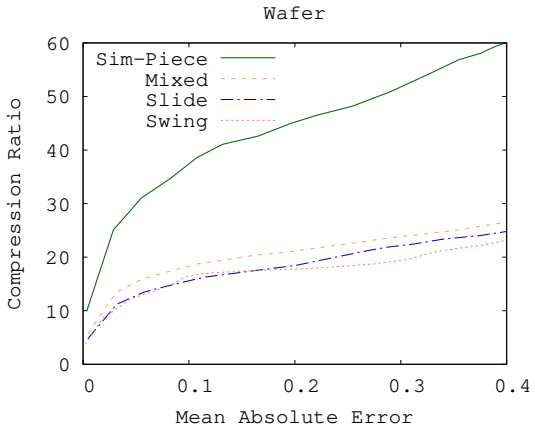


Figure 6: Compression ratio results over the MAE of the approximation achieved by each PLA technique.

Sim-Piece [4, 17, 33]. This step requires linear time. Moreover, storing this line in order to decompress the signal requires just two values that are accounted for in the calculation of the compression ratio. The result is that multiple segments share the same starting point which substantially increases performance of Sim-Piece, even in extreme cases of strongly monotonically increasing or decreasing input signals. It should be noted that this simple workaround has no effect on the competitive PLA methods, since subtracting a line simply rotates the input signal over the starting point (altering the slope of all points by the same amount). Thus, all these methods produce exactly the same number of segments (rotated) compared to the initial signal. Moreover, the total time required to come up with the detrended signal and process it is 6.7*ms*, a mere 4.4% over the running time of Sim-Piece, which is 154*ms*.

4.6 Effect of Seasonality

One key idea that Sim-Piece builds upon and contributes to its performance is the existence of *seasonal patterns* in real-world time series. Many similar repeated sequences can be represented in a single group to help increase the compression ratio. Here, we investigate how seasonality in the input signal impacts the compression ratio of Sim-Piece, as well as earlier PLA methods.

In particular, we created a synthetic signal using the popular additive model [17, 33], where a seasonal component is added together with a random walk model. For the latter each point is generated to be higher or lower (with the same probability) from the previous one, by a magnitude selected uniformly in a range that is 10× the error threshold value used. We generate a data point every 1 minute, using a sine function to simulate a daily pattern.

Figure 9 shows the impact of the signal's seasonality on the compression ratio overtime. As expected, the Mixed, Slide and Swing PLA algorithms cannot take advantage of the seasonal pattern, resulting in an almost constant compression ratio. On the contrary, Sim-Piece quickly discovers similarities between the produced segments and utilizes them in order to achieve higher compression results. In the same figure we also depict the compression ratio of an alternative implementation of Sim-Piece (denoted as Sim-Piece-delta), which instead of processing all previous records, it merges the newly created PLA intervals from the last day, with the previously computed ones. This version of Sim-Piece necessitates a small change in the compressed representation, were both the upper and lower slopes are kept, instead of the middle one (see Subsection 3.3). The modified algorithm achieves slightly smaller compression ratio due to the increased space used for representing the PLA segments and the sub-optimal merging of intervals in phase-2. However is still manages to take advantage of the seasonal pattern in the data and outperforms earlier PLA techniques by a large margin.

4.7 General Purpose Compression Gains

Our next experiment focuses on the use of general purpose compression on top of PLA approaches, that may help further reduce the space requirements of storing the input signal. More specifically, we use the ZStandard [1] algorithm to compress the output of Swing, Slide and Mixed PLA algorithms, as well as our novel Sim-Piece approach. Figure 10 illustrates the space gains we obtain for each algorithm for all time series of our dataset. We set $\epsilon = 0.5\% \times range$ for all time series, except Pressure, for which we use $\epsilon = 0.05\% \times range$, as it exhibits very small mean delta. For the earlier PLA approaches we see that ZStandard does help increase the compression ratio, offering additional space savings. The improvement is most evident for Slide, as when using Zstandard its performance matches that of Mixed. However, even with

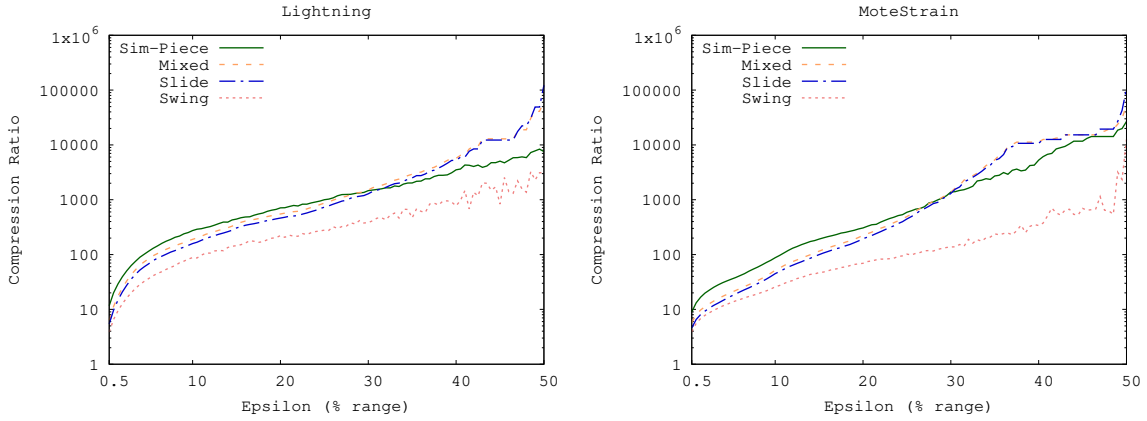


Figure 7: Exploring larger error tolerances. Sim-Piece outperforms earlier approaches up to very large values of ϵ.

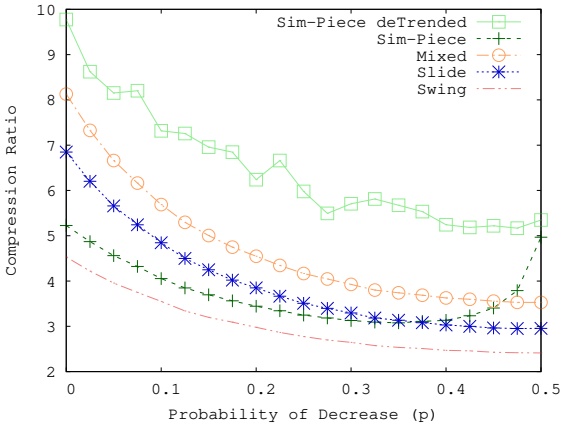


Figure 8: Impact of the degree of monotonicity. Sim-Piece performs best when the probability of decrease is set to values that resemble real-world datasets. A simple *detrending* procedure applied on the input signal significantly boosts the performance of Sim-Piece even at extreme cases of monotonicity, but has no effect on the alternative PLA methods.

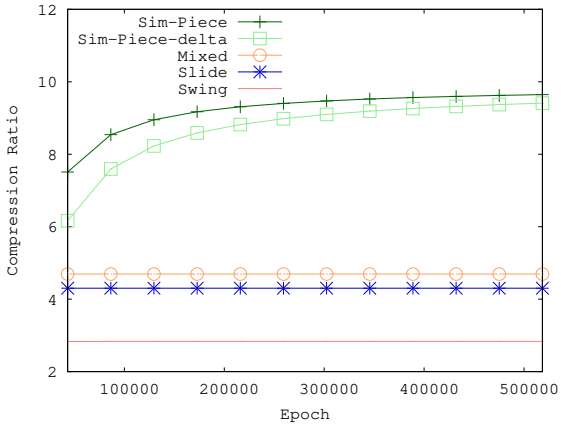


Figure 9: Effect of seasonality. The performance of both variations of Sim-Piece increases overtime, as they exploit similarities due to the seasonal component.

the benefit of using an additional general purpose compression algorithm, none of the earlier PLA approaches is able to compete against our newly proposed Sim-Piece algorithm, which clearly offers a better compression ratio *without* any added help. Moreover, using Zstandard on top of Sim-Piece provides improvements that are on par with those achieved for other PLA algorithms, leading to far greater compression ratios. This shows that the advancements offered by Sim-Piece on exploiting patterns that appear in time sequences can be further strengthened by general purpose compression algorithms, just as earlier PLA approaches.

4.8 Execution Time

Table 4 reports the time required to process each of the time-series datasets using the Slide [10], Swing [10], Mixed [24], and Sim-Piece algorithms. The results are averages of multiple executions and consider two settings: the first one considers an error threshold ϵ equal to 0.5% of the dataset's range, whereas the second one sets ϵ to be equal to 5% of the dataset's range. Moreover, for Sim-Piece we report the time needed overall, as well as for each of its two phases.

We observe in Table 4 that Sim-Piece is faster by at least one order of magnitude than the second best approach in terms of compression ratio, i.e., Mixed. Furthermore, our results show that the performance of Sim-Piece is most of the times faster than the Slide algorithm as well. The only exceptions are for the WindSpeed and WindDirection datasets, when the error threshold is set to $\epsilon = 0.5\% \times range$. In this case, as we see in Table 5, the number of intervals produced after the first phase of Sim-Piece is very large. This slows down the execution of phase-2, leading to an overall execution time that is slightly larger than that of Slide. The Swing algorithm, that provides the worst compression ratio among the approaches investigated here, is shown to be the fastest approach. However, the results of Table 4 clearly show that the

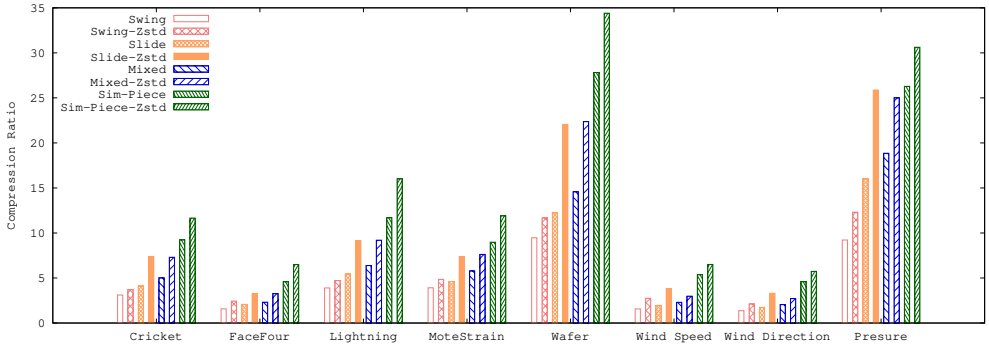


Figure 10: Effect of additionally applying general purpose compression to the output of PLA algorithms.

Table 4: Execution time (in ms.) for 0.5% and 5% Epsilon.

	Dataset	Swing		Slide		Mixed		Sim-Piece					
								Phase 1		Phase 2		Total	
		0.5%	5%	0.5%	5%	0.5%	5%	0.5%	5%	0.5%	5%	0.5%	5%
UCR	Cricket	53	48	475	538	3,936	4,910	78	104	306	69	384	173
UCR	FaceFour	2	2	28	28	223	219	3	3	9	2	12	5
UCR	Lightning	8	5	63	62	532	536	9	7	16	1	25	8
UCR	MoteStrain	6	5	79	76	516	530	6	8	27	8	33	16
UCR	Wafer	35	26	829	817	4,882	4,869	50	45	99	48	149	93
NEON	Wind Speed	283	312	1,815	1,827	22,380	22,333	232	170	2,578	286	2,810	456
NEON	Wind Dir.	101	60	733	728	6,651	6,590	93	69	829	218	922	287
NEON	Pressure	190	228	3,840	3,919	50,299	49,027	319	326	32	1	351	327

execution time of `Sim-Piece` decreases as the value of ϵ grows. This is due to the smaller number of segments produced during the first phase of the algorithm, as shown in the results of Table 5. Thus, even though `Sim-Piece` is not always faster than Slide for small values of ϵ, it becomes much more efficient as the value of ϵ grows, providing significantly better compression ratio for all possible error-thresholds values (e.g. Figure 7).

The superiority of `Sim-Piece` is evident in Figure 11, that illustrates the average trade-off between compression time and ratio achieved for all algorithms of our experimental setting, when ϵ is set at 5%. We clearly see how `Sim-Piece` provides impressive space savings while also being almost equivalently efficient with the fastest approach. The findings depicted in Figure 11 establish our algorithm as the undisputed preferable option for space-efficient approximation of time series.

5 RELATED WORK

We review here existing approaches that are used to approximate time-series data. We also discuss compression techniques studied in relevant fields, as well as two lossless compression algorithms.

An optimal algorithm for approximating sensor data using Piecewise Constant Approximation is given in [20]. A cache filter is used, which predicts that the next data point will have a value within the error threshold from the previous one. Thus, a new data point needs to be recorded only when it violates the error constraint. A similar approach is discussed in [27].

Elmeleegy et al. [10] propose Swing and Slide, a novel joint- and a disjoint-segment PLA algorithm, respectively. Similar algorithms had been independently discovered in earlier works, by Gritzali and Papakonstantinou [14] and O'Rourke [28]. Swing constructs the longest possible line segment starting from a fixed origin point by adjusting two bounding slope lines until reaching a break-up point, which will generate a new joint segment, starting from the last point of the previous segment. Slide filters are different as they generate disjoint segments as an approximation for the original data points. This gives them more flexibility at the expense of having to store an additional value for each segment. Updating the bounding slope lines can be optimized with the use of the convex hull of the observed data points, which allows for checking only against the points of the convex hull, instead of the significantly larger number of all the data points observed in the current filtering interval.

GreedyPLR [34] is a variant of Swing in which the point used to *swing* up and down is set to be the intersection of the upper and lower bounding slopes, instead of the starting point of a segment. This variation provides a wider angle than Swing while preserving a worst-case O(1) complexity. Similar to Slide [10] and the work in [28], OptimalPLR [34] uses convex hulls to find the optimal PLA with regard to the number of line segments using only disjoint segments and may offer faster processing time [8].

Hakimi and Schmeichel [16] solve optimally the continuous PLA problem, in which the approximation segments are forced to form a continuous function. This allows for representing the segments

Table 5: Intervals before and after the execution of phase-2 and respective reduction percentage.

Dataset	0.5%			5%		
	Before	After	Red.%	Before	After	Red.%
Cricket	139,112	10,293	93%	16,660	1,678	90%
FaceFour	13,831	2,559	81%	2,944	631	79%
Lightning	18,012	2,330	87%	1,597	413	74%
MoteStrain	18,276	4,383	76%	5,260	591	89%
Wafer	70,636	6,046	91%	33,955	1,184	97%
Wind Speed	1,511,313	17,833	99%	195,216	5,358	97%
Wind Dir.	474,160	28,333	94%	149,325	5,543	96%
Pressure	36,646	17,588	52%	1,445	1,220	16%

with two values, instead of the three required when using disjoint segments. The process is similar with that of OptimalPLR; however, each new segment starts from the previous generated line instead of the break-up point. The work in [16] is based on an algorithm proposed by Imai and Iri [18]. A variation of the work in [16] is given in [9], where a simple regression model is used to obtain the best-fit line at the cost of increased complexity.

Luo et al. [24] introduce the problem of *mixed-type* PLA, aiming to optimize the representation size through an adaptive solution that uses a mixture of joint and disjoint segments. Their novel approach allows for enjoying the *best of two worlds*, i.e., the cheaper representation of joint segments and the fewer number of segments produced by disjoint-segment approaches. A dynamic programming algorithm is presented that finds the optimal sized PLA in under these settings. Moreover, the authors avoid wasting a bit for each segment to differentiate between joint and disjoint segments. Instead, the proposed representation exploits the strictly increasing sequence of positive timestamps and uses negative values to indicate a disjoint segment.

Compression of time series has been studied in the context of other work as well. The authors of [5, 6] proposed a technique that exploits the correlation and redundancy among multivariate sensor readings to construct a dictionary of real measurements, used for encoding piece-wise linear correlations among the collected data values. [7, 12] extend these techniques to work in a distributed setting of network-connected nodes, while also accounting for the presence of outliers. In [35], the authors propose an approach for efficiently processing time-series *similarity* matching queries, by dividing time series into a fixed number of equal sized segments. [19] also focuses on indexing time series and shows that APCA, a piecewise constant approximation approach can be indexed using a multidimensional index structure to allow for fast approximate querying. Guerts [11] focuses on classification of time series and uses regression trees to perform piecewise constant modelling of temporal signals. Patterns are extracted from these models and are combined in decision trees to give interpretable classification rules. SAX [23] introduces a symbolic approach for time-series dimensionality reduction that allows distance measures to be defined on the symbolic representation. This enables running certain data mining algorithms on the symbolic representation, while producing results that are similar to what the algorithms that operate on the original data produce.

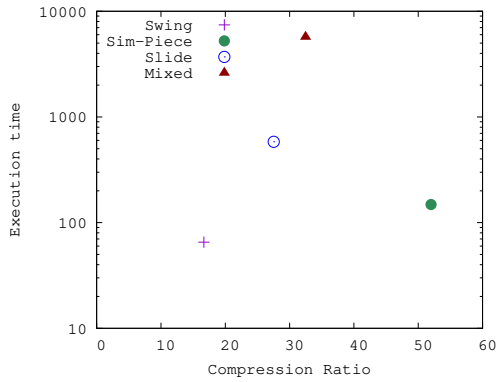


Figure 11: Trade-off between compression time (ms in log scale) and space (compression ratio) for various algorithms.

In the field of lossless time series compression, Gorilla [30] is a compression algorithm, particularly suited for floating-point time series in which the neighboring data points do not change significantly. Timestamps and values are compressed separately, similarly to other time-aware schemes[22]. The more recent Chimp [21] lossless compression algorithm has been shown to be very competitive in terms of space requirements with *lossy* PLA approaches. Our `Sim-Piece` algorithm is motivated by the impressively low space requirements of Chimp and achieves significant improvements with regards to the state-of-the-art in PLA algorithms, offering better compression than Chimp, even when the precision error threshold is set extremely low.

6 CONCLUSIONS

In this paper we introduced `Sim-Piece`, a novel approach that exploits similarities among the line segments produced by PLA in order to provide a joint representation for multiple segments. Our `Sim-Piece` algorithm, offers substantially better space savings on a large range of maximum error values, including when the acceptable error bound is small. This is particularly important as recent lossless algorithms have been shown to outperform earlier lossy approaches in such settings.

The experimental evaluation of our algorithm using real and synthetic datasets shows that we attain compression ratios that are far superior than the second best approach, for all the acceptable error bounds that were investigated. Moreover, `Sim-Piece` outperforms past PLA methods even in extreme cases of strongly monotonically increasing or decreasing input signals, and manages to exploit seasonal components to provide additional savings. As far as the execution time is concerned, we show that `Sim-Piece` is up to 150× faster than the second best approach in terms of compression ratio, offering overall significantly improved efficiency with regard to both space and speed. We plan to extend `Sim-Piece` by investigating the parallelization opportunities of our scheme.

ACKNOWLEDGMENTS

This work has received funding from the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under grant agreement No 779.

REFERENCES

[1] 2015. *Zstd.* Retrieved May 21, 2022 from https://facebook.github.io/zstd

[2] Dau Hoang Anh, Keogh Eamonn, Kamgar Kaveh, Yeh Chin-Chia Michael, Zhu Yan, Gharghabi Shaghayegh, Ratanamahatana Chotirat Ann, Yanping, Hu Bing, Begum Nurjahan, Bagnall Anthony, Mueen Abdullah, Batista Gustavo, and Hexagon-ML. 2018. The UCR Time Series Classification Archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018.

[3] Scott H. Cameron. 1966. *Piece-wise linear approximations.* Technical Report. IIT RESEARCH INST CHICAGO IL COMPUTER SCIENCES DIV.

[4] Robert B Cleveland, William S Cleveland, Jean E McRae, and Irma Terpenning. 1990. STL: A seasonal-trend decomposition. *Journal of Official Statistics* 6, 1 (1990), 3–73.

[5] Antonios Deligiannakis and Yannis Kotidis. 2005. Data Reduction Techniques in Sensor Networks. *IEEE Data Eng. Bull.* 28, 1 (2005), 19–25. http://sites.computer.org/debull/A05mar/kotidis.ps

[6] Antonios Deligiannakis, Yannis Kotidis, and Nick Roussopoulos. 2004. Compressing Historical Information in Sensor Networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004.* ACM, 527–538. https://doi.org/10.1145/1007568.1007628

[7] Antonios Deligiannakis, Yannis Kotidis, and Nick Roussopoulos. 2007. Dissemination of compressed historical information in sensor networks. *VLDB J.* 16, 4 (2007), 439–461. https://doi.org/10.1007/s00778-005-0173-5

[8] Romaric Duvignau, Vincenzo Gulisano, Marina Papatriantafilou, and Vladimir Savic. 2018. Piecewise Linear Approximation in Data Streaming: Algorithmic Implementations and Experimental Analysis. *CoRR* abs/1808.08877 (2018). arXiv:1808.08877

[9] Romaric Duvignau, Vincenzo Gulisano, Marina Papatriantafilou, and Vladimir Savic. 2019. Streaming piecewise linear approximation for efficient data management in edge computing. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019.* ACM, 593–596. https://doi.org/10.1145/3297280.3297552

[10] Hazem Elmeleegy, Ahmed K. Elmagarmid, Emmanuel Cecchet, Walid G. Aref, and Willy Zwaenepoel. 2009. Online Piece-wise Linear Approximation of Numerical Streams with Precision Guarantees. *Proc. VLDB Endow.* 2, 1 (2009), 145–156. https://doi.org/10.14778/1687627.1687645

[11] Pierre Geurts. 2001. Pattern Extraction for Time Series Classification. In *Principles of Data Mining and Knowledge Discovery.* 115–127.

[12] Nikos Giatrakos, Yannis Kotidis, Antonios Deligiannakis, Vasilis Vassalos, and Yannis Theodoridis. 2010. TACO: tunable approximate computation of outliers in wireless sensor networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010.* ACM, 279–290. https://doi.org/10.1145/1807167.1807199

[13] Martin Charles. Golumbic. 1980. *Algorithmic graph theory and perfect graphs / Martin Charles Golumbic.* Academic Press New York. xx, 284 p. : pages.

[14] F. Gritzali and G. Papakonstantinou. 1983. A fast piecewise linear approximation algorithm. *Signal Processing* 5, 3 (1983), 221–227. https://doi.org/10.1016/0165-1684(83)90070-1

[15] Udaiprakash I. Gupta, D. T. Lee, and Joseph Y.-T. Leung. 1982. Efficient algorithms for interval graphs and circular-arc graphs. *Networks* 12, 4 (1982), 459–467. https://doi.org/10.1002/net.3230120410

[16] S. Louis Hakimi and Edward F. Schmeichel. 1991. Fitting polygonal functions to a set of points in the plane. *CVGIP Graph. Model. Image Process.* 53, 2 (1991), 132–136. https://doi.org/10.1016/1049-9652(91)90056-P

[17] Charles C. Holt. 2004. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting* 20, 1 (2004), 5–10. https://doi.org/10.1016/j.ijforecast.2003.09.015

[18] Hiroshi Imai and Masao Iri. 1986. An optimal algorithm for approximating a piecewise linear function. *Journal of information processing* 9, 3 (1986), 159–162.

[19] Eamonn J. Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael J. Pazzani. 2001. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In *Proc. of the 2001 ACM SIGMOD Int. Conf. on Management of Data, Santa Barbara, CA, USA, May 21-24, 2001.* ACM, 151–162. https://doi.org/10.1145/375663.375680

[20] Iosif Lazaridis and Sharad Mehrotra. 2003. Capturing Sensor-Generated Time Series with Quality Guarantees. In *Proc. of the 19th Int. Conf. on Data Engineering, March 5-8, 2003, Bangalore, India.* IEEE Computer Society, 429–440. https://doi.org/10.1109/ICDE.2003.1260811

[21] Panagiotis Liakos, Katia Papakonstantinopoulou, and Yannis Kotidis. 2022. Chimp: Efficient Lossless Floating Point Compression for Time Series Databases. *Proc. VLDB Endow.* 15, 11 (2022), 3058–3070.

[22] Panagiotis Liakos, Katia Papakonstantinopoulou, Theodore Stefou, and Alex Delis. 2022. On Compressing Temporal Graphs. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022.* IEEE, 1301–1313. https://doi.org/10.1109/ICDE53745.2022.00102

[23] Jessica Lin, Eamonn J. Keogh, Stefano Lonardi, and Bill Yuan-chi Chiu. 2003. A symbolic representation of time series, with implications for streaming algorithms. In *Proc. of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, DMKD 2003, San Diego, California, USA, June 13, 2003.* ACM, 2–11. https://doi.org/10.1145/882082.882086

[24] Ge Luo, Ke Yi, Siu-Wing Cheng, Zhenguo Li, Wei Fan, Cheng He, and Yadong Mu. 2015. Piecewise linear approximation of streaming time series data with max-error guarantees. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015.* IEEE Computer Society, 173–184. https://doi.org/10.1109/ICDE.2015.7113282

[25] National Ecological Observatory Network (NEON). 2022. Barometric pressure above water on-buoy (DP1.20004.001). https://doi.org/10.48443/V4AP-NY05

[26] National Ecological Observatory Network (NEON). 2022. Windspeed and direction above water on-buoy (DP1.20059.001). https://doi.org/10.48443/E16C-8686

[27] Chris Olston, Jing Jiang, and Jennifer Widom. 2003. Adaptive Filters for Continuous Queries over Distributed Data Streams. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003.* ACM, 563–574. https://doi.org/10.1145/872757.872825

[28] Joseph O'Rourke. 1981. An On-Line Algorithm for Fitting Straight Lines Between Data Ranges. *Commun. ACM* 24, 9 (1981), 574–578. https://doi.org/10.1145/358746.358758

[29] Theodosios Pavlidis. 1973. Waveform Segmentation Through Functional Approximation. *IEEE Trans. Computers* 22, 7 (1973), 689–697. https://doi.org/10.1109/TC.1973.5009136

[30] Tuomas Pelkonen, Scott Franklin, Paul Cavallaro, Qi Huang, Justin Meza, Justin Teller, and Kaushik Veeraraghavan. 2015. Gorilla: A Fast, Scalable, In-Memory Time Series Database. *Proc. VLDB Endow.* 8, 12 (2015), 1816–1827. https://doi.org/10.14778/2824032.2824078

[31] Henry Stone. 1961. Approximation of curves by line segments. *Math. Comp.* 15, 73 (1961), 40–47.

[32] Haoyu Wang and Shaoxu Song. 2022. Frequency Domain Data Encoding in Apache IoTDB. *Proc. VLDB Endow.* 16, 2 (2022), 282–290. https://www.vldb.org/pvldb/vol16/p282-song.pdf

[33] Peter R. Winters. 1960. Forecasting Sales by Exponentially Weighted Moving Averages. *Management Science* 6, 3 (1960), 323–342. https://doi.org/10.1287/mnsc.6.3.324

[34] Qing Xie, Chaoyi Pang, Xiaofang Zhou, Xiangliang Zhang, and Ke Deng. 2014. Maximum error-bounded Piecewise Linear Representation for online stream approximation. *VLDB J.* 23, 6 (2014), 915–937. https://doi.org/10.1007/s00778-014-0355-0

[35] Byoung-Kee Yi and Christos Faloutsos. 2000. Fast Time Sequence Indexing for Arbitrary Lp Norms. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt.* Morgan Kaufmann, 385–394.