A Performance Study of Piecewise Constant Approximation in Streaming Data Compression

Huan[2], Thanh-Dang Diep[1,3,⋆], and Nam Thoai[1,2,3]

[1] Faculty of Computer Science and Engineering,
[2] Advanced Institute of Interdisciplinary Science and Technology,
Ho Chi Minh City University of Technology (HCMUT),
268 Ly Thuong Kiet Street, District 10, Ho Chi Minh City, Vietnam
[3] Vietnam National University Ho Chi Minh City,
Linh Trung Ward, Thu Duc District, Ho Chi Minh City, Vietnam
`{huan,dang,namthoai}@hcmut.edu.vn`

Abstract. Data streaming applications proliferate, making ubiquitous collection devices easily overload server storage, network bandwidth, *etc.* and hence requiring data to be compressed on the fly. Among various techniques for compressing streaming data, Piecewise Approximation, categorized as either Piecewise Constant Approximation (PCA), Piecewise Linear Approximation (PLA) or Piecewise Polynomial Approximation (PPA), is the most popular one. Further, PCA which splits data into multiple segments and represents them with constant values, can potentially overcome Big Data challenges due to its lightweight feature. This paper is to serve as a recommendation to select the most well-suited method for particular applications by providing a comprehensive performance study between PCA algorithms in the context of data stream compression. Additionally, the performance of PPA techniques is provided as a baseline for PCA. Our experimental results show that even a simple PCA outperforms PPA in some circumstances.

Keywords: Data stream compression, piecewise constant approximation, piecewise polynomial approximation.

1 Introduction

The advent of data streaming applications leads to the ubiquity of continuous data collection in high fidelity, which triggers numerous Big Data challenges [3]. The massive amount of data continuously generated and accumulated over time can easily overload network bandwidth and system storage. Despite the fact that hardware prices are falling, the investment costs to transmit and store raw data are still unaffordable. In many cases, these streams of data must be compressed on the fly to benefit the overall system cost by reducing transfer bandwidth at edge devices, expanding sensor battery life [8], saving storage space [12], *etc.*

⋆ Corresponding author.

Compression algorithms are often categorized as either lossless or lossy techniques. In the lossless compression, original data can be completely reconstructed, while lossy compression can tolerate some data errors under an acceptable rate to achieve better execution time and compression ratio. This particular trade-off is essential especially in Big Data applications. Among lossy algorithms, Piecewise Approximation (PA) is the most popular technique [9] which splits streaming data into separated segments and approximates them individually. PA can be grouped as either (1) Piecewise Constant Approximation (PCA) which represents each segment with a horizontal line, (2) Piecewise Linear Approximation (PLA) that uses a straight line or (3) a line with a higher degree for each segment in case of Piecewise Polynomial Approximation (PPA). Since virtually all algorithms must be fast enough to deal with Big Data challenges, we concentrate on evaluating PCA – the most lightweight technique. Besides, the performance of PPA is given as a baseline to show PCA's efficiency.

In case of PCA, Keogh et al. [5] first introduced the Piecewise Aggregate Approximation (PAA) – a PCA method which divides data into equal segments of length k and approximates them with the corresponding mean values. Later, Chakrabarti et al. [1] devised Adaptive Piecewise Constant Approximation (APCA) which improves the compression ratio of PAA by allowing segments with different lengths. Both PAA and APCA are designed for databases that require all data to be ready in advance and are not suitable in streaming fashions because of high latency for buffering incoming data. Moreover, PAA and APCA do not provide any guarantee of approximation quality. For compressing streaming data under a bounded error rate, Lazaridis et al. [6] provided Poor Man Compression (PMC) – an online algorithm which can not only process data one-by-one in a one-pass manner but also ensure that each approximation data point does not exceed a predefined ϵ threshold. However, PMC only compresses data when the current segment cannot be approximated any further. Because of this greedy strategy, PMC may suffer from losing data due to the delay of storing or transferring process. Lately, Mahbub et al. [7] proposed a Hybrid-PCA which combines PMC and PAA to solve the delay problem by triggering segment split when either approximation error or buffer length exceeds a particular threshold. On the other hand, the theory of PPA can be preceded back to the 1970s [11, 10] to solve the curve fitting and segmentation problems. However, applying PPA for compressing streaming data has only emerged recently. A previous work [4] leverages the smart caching mechanism in the traditional PPA technique to cope with streaming settings, which approximates each segment by a faster series of matrix products and provides the error guarantee as well by following the greedy segment splitting strategy. In general, PCA tends to ignore some data details to boost execution time while PLA in contrast, sacrifices the execution time to attain higher approximation quality. Nevertheless, all of the aforementioned studies lack a cross-comparison and only focus on evaluating compression ratio, compression time and approximation error.

As a consequence, this paper revisits the PPA and PCA techniques for compressing streaming data but with more aspects and criteria in order to provide a

Table 1. Algorithm summary.

Algorithm	L_1 Bound	L_∞ Bound	Segment Bound	Time Complexity	Memory Complexity
PMC-MR [6]	✓	✓		$O(1)$	$O(1)$
PMC-Mean [6]	✓	✓		$O(1)$	$O(1)$
Hybrid-PCA [7]	✓	✓	✓	$O(1) \sim O(m \times w)$	$O(m \times w)$
PPF-L_1 [4]	✓			$O(k \times n) \sim O(k \times n^2)$	$O(k \times m \times n)$
PPF-L_∞ [4]	✓	✓		$O(k \times n) \sim O(k \times n^2)$	$O(k \times m \times n)$

more comprehensive view. Our results show that not only each of the algorithms in question can reduce data size up to several hundred times under an acceptable error rate, but also highlight that even a simple PCA algorithm can sometimes surpass complex PPA algorithms in all of the considered metrics. The rest of this paper is organized as follows. Section 2 reviews PCA and PPA algorithms in the literature, and then briefly discusses the set of metrics used for evaluation in Section 3. Section 4 elaborates the experiments, together with our analysis on their findings and Section 5 concludes our study.

2 Methods

This section revisits the PCA and PPA algorithms in the literature with some modifications where applicable to cope with the streaming data compression problem. Since compression can occur at edge side before transmitting or at server side before archiving, we use the **yield** operation to describe both the compression-transmission and compression-storing processes. Table 1 gives a brief overview of the PCA and PPA algorithms.

2.1 Poor Man Compression (PMC)

The first method to be considered is PMC [6], which is proven to produce minimal segments among PCA algorithms. As depicted in Algorithm 1, PMC keeps track of a pair of (max, min) values in each segment, and this can be done incrementally in $O(1)$ time. When the range between max and min exceeds 2ϵ at time t, the segment of preceding points til $t-1$ is yielded with the corresponding mid-range value and its last endpoint.

The error of each individual data point is proven to always lie between $[-\epsilon, \epsilon]$ with the minimum number of segments. To reduce the accumulated error, the authors suggested approximating each segment with the $mean$ value instead. The splitting condition will now be whether $max - mean > \epsilon$ or $mean - min > \epsilon$. This can also be done incrementally in $O(1)$ time. We label the PMC variants which use mid-range and mean value as PMC-MR and PMC-Mean, respectively. Moreover, since PMC only needs to maintain representative values for each segment, the space complexity is $O(1)$ as well.

[4] PPF-L_∞ is adapted from PPF-L_1 to offer L_∞ bound.

Algorithm 1: PMC-MR

```
input  : Infinite stream $T$, error bound $\epsilon$.
output : Constant value and end point of each segment.
set $-\infty$ to $max$;
set $\infty$ to $min$;
repeat
    set $next(T)$ to $p$;
    if $p > max$ then
        set $p$ to $max$;
    end
    if $p < min$ then
        set $p$ to $min$;
    end
    if $max - min > 2\epsilon$ then
        yield $(prev\_max + prev\_min)/2,\ prev\_p$;
        set $p$ to $max$;
        set $p$ to $min$;
    end
    set $max$ to $prev\_max$;
    set $min$ to $prev\_min$;
    set $p$ to $prev\_p$;
until $empty(T)$;
```

2.2 Hybrid-PCA

The next approach is Hybrid-PCA [7] aiming to solve the delay problem of PMC by maintaining a window of size w where incoming data is cached and a buffer contains at most m windows indicating the maximum length of each segment.

The respective pseudocode is given in Algorithm 2. Hybrid-PCA will form a window from arriving data, temporally move the data to a buffer when the window is full, and $max - min$ is then calculated. If the buffer reaches its limit, then yield the corresponding mid-range value along with the buffer size. On the other hand, if the buffer range exceeds the error threshold indicating that it cannot be approximated together with the current window, then remove this window and yield the buffer. Next, we have to check whether this current window can be approximated with its mid-range value. Store this window in the buffer if the condition is not violated or else apply the PMC algorithm to approximate window with the minimum segments.

Unlike PMC, Hybrid-PCA has a fixed bound on the length of each segment enforcing it to "early output" even when the error rate is not violated. Hence, the maximum delay is at most $m \times w$. Moreover, since the algorithm complexity is dominated by the max and min operations, the complexity varies from $O(1)$ to $O(m \times w)$ depending on the implementation. The space complexity of Hybrid-PCA is $O(m \times w)$ in worst cases when its buffer is full.

Algorithm 2: Hybrid-PCA

```
input  : Infinite stream $T$, error bound $\epsilon$, window size $w$, window count $m$.
output : Constant value and length of each segment.
1  set $\emptyset$ to buffer;
2  set $\emptyset$ to window;
3  repeat
4      set $next(T)$ to $p$;
5      append $p$ to window;
6      if $size(window) = w$ then
7          append window to buffer;
8          if $max(buffer) - min(buffer) <= 2\epsilon$ then
9              if $size(buffer) = m \times w$ then
10                 yield $(max(buffer) + min(buffer))/2,\ size(buffer)$;
11                 delete buffer and window;
12             end
13         else
14             delete window from buffer;
15             yield $(max(buffer) + min(buffer))/2,\ size(buffer)$;
16             set window to buffer;
17             delete window;
18             if $max(buffer) - min(buffer) > 2\epsilon$ then
19                 Apply PMC-MR algorithm with buffer and $\epsilon$;
20                 delete buffer;
21             end
22         end
23     end
24 until $empty(T)$;
```

2.3 Piecewise Polynomial Fitting (PPF)

The final algorithm is PPF [4] which leverages the traditional curve-fitting technique with a slight modification to cope with streaming constraints. In detail, suppose a discrete data point $(time, value)$ is formulated as (t, y) and a polynomial function with degree k as $f_k(t) = \sum_{i=0}^{k} \theta_i t^i$ where θ_i are coefficients. Assume that n data points can be successfully approximated by a polynomial function then $y_n = f_k(t_n)$ or it can be written as $\vec{y} = X\vec{\theta}$ where

$$X = \begin{bmatrix} 1 & t_1 & \dots & t_1^k \\ 1 & t_2 & \dots & t_2^k \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & \dots & t_3^k \end{bmatrix}, \quad \vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_k \end{bmatrix}, \quad \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

The best approximation in case of minimizing the square error can be found by utilizing least square method. In other words, we have to find $\vec{\theta}$ to minimize $||X\vec{\theta} - \vec{y}||^2$. By using normal equation, we can derive $\vec{\theta} = (X^T X)^{-1} X^T \vec{y}$.

The procedure of compressing streaming data with PPF is demonstrated in Algorithm 3. The logic here is simple, move ahead and keep fitting until the seg-

Algorithm 3: PPF-L_1

```
input  : Infinite stream $T$, error bound $\epsilon$, polynomial degree $k$.
output : Coefficients of polynomial function and length of each segment.
1  set $\emptyset$ to seg;
2  repeat
3      append $next(T)$ to seg;
4      if $size(seg) == k+1$ then
5          Approximate seg with model by algorithm 4;
6      elif $size(seg) > k+1$ then
7          if $error(seg, model) > size(seg) \times \epsilon$ then
8              Approximate seg with new model by algorithm 4;
9              if $error(seg, new\ model) > size(seg) \times \epsilon$ then
10                 yield $model, size(seg) - 1$;
11                 Retain only the last value in seg;
12             end
13         end
14     end
15 until $empty(T)$;
```

Algorithm 4: Enhance normal equation

```
input  : Data segment seg, polynomial degree $k$.
output : Polynomial function best fit data in seg.
1  Initialize vector $\vec{y}$ with seg;
2  if cache does not include $key = size(seg)$ then
3      Initialize matrix $X$ with $size(seg)$ and $k$;
4      Calculate $(X^T X)^{-1} X^T$;
5      append $(X^T X)^{-1} X^T$ to cache with $key = size(seg)$;
6  else
7      set $(X^T X)^{-1} X^T$ to cache with $key = size(seg)$ ;
8  end
9  return $(X^T X)^{-1} X^T \vec{y}$
```

ment cannot be approximated any further, we then yield it and start a new one. The segment split condition (line 6 and 9) originally proposed is tight with accumulated error, but it can be adjusted to $max\_error(seg, model) > \epsilon$ to provide L_∞ bound. We name the PPF variants with accumulated bound and individual bound as PPF-L_2 and PPF-L_∞, respectively. Besides, since the most intensive work of this algorithm concentrates on the series of matrix multiplication, a caching technique is proposed to speed up the fitting process. An observation from the authors is that the matrix X depends only on the number of data points within the segment and the degree k. Since each segment is approximated independently and if data is collected periodically with a fixed interval, then the segments with the same size will produce the same $(X^T X)^{-1} X^T$ intermediate results. The intermediate results are cached in association with its segment size as detailed in Algorithm 4, the final complexity varies from $O(k \times n^2)$ to $O(k \times n)$

with n indicates the length of the current segment and k is the polynomial degree. In terms of memory usage, the most space consumption is from the caching process. Thus, the space complexity is $O(k \times m \times n)$ with m is the number of saved entries.

3 Evaluation Metrics

We assess compression algorithms using a wide range of metrics, each of which can benefit different scenarios.

- **Compression ratio (CR):** The most important metric which induces the total amount of space saved after compression. The higher compression ratio, the better compression capability and its formula is straightforwardly defined as the division of original data size by the compression output:

$$CR = \frac{size\_of\_original\_data}{size\_of\_compress\_data} \tag{1}$$

- **Reconstruction error:** This criterion is also one of the most crucial ones which denotes the quality of lossy algorithms. We represent the reconstruction error using Root Mean Square Error ($RMSE$) and Maximum Error (Max_Error) for algorithms that are not naturally L_∞ bound. The lower error rate, the higher approximation quality and can be calculated from the deviation between the decompressing (v') against original data (v):

$$Max\_Error = max_i(|v_i - v_i'|) \tag{2}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_i - v_i')^2} \tag{3}$$

- **Compression time:** The average time to process each coming data point is also considered since data in streaming only comes at once. The lower time used for compressing each data point, the lower probability of missing data.
- **Decompression time:** In some scenarios for example when data is compressed at edges before transmitting to servers, a lower decompressing time can prevent servers from overwhelming and avoid missing new coming data. As a consequence, the average time taken to decompress a compressed record, *i.e.* data segment in case of PA, needs to be tracked as well.
- **Memory usage:** This metric monitors the peak memory usage during both compression and decompression phases through Virtual Memory Size (VSZ) and Resident Set Size (RSS) measures. The less resource usage the more types of devices the algorithm can support. This metric is critical when dealing with resource-limited devices. Additionally, memory usage also denotes the power of parallelism of the algorithm which can potentially solve the problem of compressing multivariate streaming data.

4 Experiment

4.1 Experimental setting

All the experiments are run on a computer with an Intel® Core™ i7-6700 processor and 8 GB DDR3 1600 MHz RAM. We evaluate PMC-MR, PMC-Mean, Hybrid-PCA as well as PPF with both accumulated and maximum error, all are implemented with C++11. The compressed outputs are all in binary format, each segment contains an integer and a floating-point number for PMC and Hybrid-PCA; an integer and $k+1$ floating-point numbers for PPF.

In case of algorithms with hyper-parameters, PPF with a degree of 2 is chosen since it yields the best results; the window size and number of windows in Hybrid-PCA are selected based on the average and maximum length of segments produced by PMC respectively. The error tolerance is set to 5% of $max - min$ for each dataset for all experiments.

Our tests use multiple datasets from URC time series classification archives [2] which are widely used to evaluate time series algorithms. In each dataset, we first group the data based on its classes then append all of them horizontally, and finally generate the synthetic timestamp started from 1 with an interval of 1. The chosen datasets are listed below:

- **Crop:** Dataset generated from Earth Observation (EO) satellites captures the same geometrical area for each class over time.
- **EthanolLevel:** Dataset represents the spectrograph of wine bottles in different levels of alcohol corresponding to each class.
- **Fungi:** Dataset contains melt curves of the rDNA internal transcribed spacer (ITS) region of fungus where each species is grouped into the same class.
- **PowerCons:** Dataset contains the electric power consumption of an individual household in one year. Class 1 is the warm season (from April to September) and 2 is the cold season (from October to March).
- **StarLightCurves:** Dataset monitors the discrete brightness of a celestial object over time whose class has been determined by an expert.

4.2 Evaluation Results

Table 2 presents the results of all datasets and algorithms. Although there is no algorithm that can outperform the others in all datasets, the PCA algorithms produce quite good results compared to the PPA algorithms.

In case of PCA, PMC-Mean and Hybrid-PCA cannot output better compact results than PMC-MR, which is proven to produce the least number of segments. While in PPA, since PPF-L_1 is not L_∞ guaranteed, thus it can approximate a longer segment and hence has the highest compression ratio but is not suitable for such applications having a strict error constraint. PPF-L_∞ is a better solution which can output both L_∞ guaranteed and a good enough compression ratio. Even though PPA can approximate more data points under fixed error tolerance, PPA requires more values to represent a single segment than PCA. Thus PCA

Table 2. Experimental results.

Dataset	Algorithm	CR	RMSE	Max Error	Compress Time (ns)	Decompress Time (μs)	Compress VSZ (MB)	Compress RSS (MB)	Decompress VSZ (MB)	Decompress RSS (MB)
Crop ($\epsilon = 0.103$)	*PMC-MR*	18.1	0.054	0.103	55	30	212.87	138.25	79.61	4.47
	PMC-Mean	14.43	0.044	0.103	76	32	212.87	138.25	79.61	4.46
	Hybrid-PCA	13.57	0.051	0.103	143	36	212.87	138.25	81.11	5.46
	PPF-L_1	91.47	0.135	1.21	896	128	233.23	159	78.49	4
	PPF-L_∞	14.75	0.04	0.103	885	68	312.64	239.12	79.61	4.7
EthanolLevel ($\epsilon = 0.153$)	*PMC-MR*	187.5	0.105	0.153	39	91	292.66	218.12	78.48	3.75
	PMC-Mean	158.02	0.064	0.153	55	74	292.66	218.12	78.48	3.75
	Hybrid-PCA	146.64	0.08	0.153	94	68	292.66	218.12	78.48	3.75
	PPF-L_1	939.97	0.21	1.926	796	884	424.57	350.12	78.17	3.75
	PPF-L_∞	298.35	0.056	0.153	1332	314	762.07	687.5	78.3	3.875
Fungi ($\epsilon = 4.355$)	*PMC-MR*	43.46	2.64	4.353	48	21	83.06	8.5	78.02	3.625
	PMC-Mean	37.32	1.43	4.354	64	18	83.06	8.5	78.16	3.625
	Hybrid-PCA	34.52	1.97	4.353	108	17	83.06	8.5	78.16	3.625
	PPF-L_1	247.21	8.93	82.01	815	261	83.06	8.75	78.02	3.625
	PPF-L_∞	54.4	1.32	4.35	846	57	86.03	11.88	78.02	3.625
PowerCons ($\epsilon = 0.39$)	*PMC-MR*	10.56	0.22	0.39	69	8	84.35	9.875	78.16	3.625
	PMC-Mean	9	0.14	0.39	88	7	84.35	9.875	78.16	3.625
	Hybrid-PCA	7.58	0.179	0.39	151	6	84.35	9.875	78.29	3.75
	PPF-L_1	36.2	0.567	6.72	853	56	86.41	12	78.02	3.75
	PPF-L_∞	9.52	0.15	0.39	911	17	87.96	13.375	78.17	3.75
StarLightCurves ($\epsilon = 0.407$)	*PMC-MR*	262.7	0.24	0.407	38	136	1232.51	1158	79.6	4.46
	PMC-Mean	215.66	0.184	0.407	53.55	115	1232.51	1158	79.6	4.59
	Hybrid-PCA	191.68	0.211	0.407	97.88	107	1232.51	1158	79.6	4.59
	PPF-L_1	948.68	0.63	5.884	665.58	1011	1963.86	1889.25	78.5	4
	PPF-L_∞	387.57	0.14	0.407	938	394	2825.38	2750.75	78.86	4.22

can beat PPA in case of compression ratio when data is stable and this situation is quite common in real-world scenarios.

PPF-L_∞ generates the most accurate results but with the highest compression time while PMC-MR in contrast has the lowest memory usage and execution time but with the largest approximation error. The decompression time depends on the length and type of segments. In general, PPA costs more time to decompress data than PCA since it needs more computational steps. While memory usage of the decompression phase is not very different between all algorithms and mainly depends on the size of compressed file, the memory usage of PPA algorithms in the compression phase is significantly larger than PCA. As depicted in Table 1 and results from Table 2, the space complexity of Hybrid-PCA depends on the size and number of windows but in practice, they usually stay constant. But in PPF algorithms, the memory costs are huge due to the caching mechanism.

It is clearly seen that PPA can produce better output but with more overhead. Nevertheless, PCA can outperform PPA in all criteria under some specific circumstances. The selection of algorithms in case of PCA will depend on whether to either maximize only compression ratio or minimize approximation error. Furthermore, segment bound also needs to be considered.

5 Conclusion

This paper presented an evaluation of PCA and PPA algorithms for compressing streaming data to learn about their efficiency. While PPA in general leans towards the error rate and compression ratio, and PCA leans towards the processing time and resource usage, our experimental results emphasize the power of PCA which sometimes outperforms PPA in all criteria.

Acknowledgments. We acknowledge the support of time and facilities from Ho Chi Minh City University of Technology (HCMUT), VNU-HCM for this study.

References

[1] Kaushik Chakrabarti et al. "Locally adaptive dimensionality reduction for indexing large time series databases". In: *ACM Transactions on Database Systems (TODS)* 27.2 (2002), pp. 188–228.

[2] Hoang Anh Dau et al. *The UCR Time Series Classification Archive.* `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`. Oct. 2018.

[3] Romaric Duvignau et al. "Streaming piecewise linear approximation for efficient data management in edge computing". In: *Proceedings of the 34th ACM/SIGAPP symposium on applied computing.* 2019, pp. 593–596.

[4] Jianhua Gao et al. "Fast piecewise polynomial fitting of time-series data for streaming computing". In: *IEEE Access* 8 (2020), pp. 43764–43775.

[5] Eamonn Keogh et al. "Dimensionality reduction for fast similarity search in large time series databases". In: *Knowledge and information Systems* 3 (2001), pp. 263–286.

[6] Iosif Lazaridis and Sharad Mehrotra. "Capturing sensor-generated time series with quality guarantees". In: *Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405).* IEEE. 2003, pp. 429–440.

[7] Atik Mahbub et al. "Improved Piecewise Constant Approximation Method for Compressing Data Streams". In: *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT).* IEEE. 2019, pp. 1–6.

[8] Chris Olston, Jing Jiang, and Jennifer Widom. "Adaptive filters for continuous queries over distributed data streams". In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data.* 2003, pp. 563–574.

[9] Themistoklis Palpanas et al. "Online amnesic approximation of streaming time series". In: *Proceedings. 20th International Conference on Data Engineering.* IEEE. 2004, pp. 339–349.

[10] Theodosios Pavlidis and Steven L Horowitz. "Segmentation of plane curves". In: *IEEE transactions on Computers* 100.8 (1974), pp. 860–870.

[11] Michael Plass and Maureen Stone. "Curve-fitting with piecewise parametric cubics". In: *Proceedings of the 10th annual conference on Computer graphics and interactive techniques.* 1983, pp. 229–239.

[12] Huanyu Zhao et al. "An optimal online semi-connected PLA algorithm with maximum error bound". In: *IEEE Transactions on Knowledge and Data Engineering* 34.1 (2020), pp. 164–177.