# BacktrackSTL: Ultra-Fast Online Seasonal-Trend Decomposition with Backtrack Technique

Haoyu Wang
Alibaba Group
Hangzhou, China
lingzun.why@alibaba-inc.com

Hongke Guo
Alibaba Group
Beijing, China
guohongke.ghk@alibaba-inc.com

Zhaoliang Zhu
Alibaba Group
Hangzhou, China
rongdi.zzl@alibaba-inc.com

You Zhang
Alibaba Group
Hangzhou, China
zhangyou.zy@alibaba-inc.com

Yu Zhou
Alibaba Group
Beijing, China
tuhu@alibaba-inc.com

Xudong Zheng
Alibaba Group
Hangzhou, China
xudong.zxd@alibaba-inc.com

## Abstract

Seasonal-trend decomposition (STD) is a crucial task in time series data analysis. Due to the challenges of scalability, there is a pressing need for an ultra-fast online algorithm. However, existing algorithms either fail to handle long-period time series (such as OnlineSTL), or need time-consuming iterative processes (such as OneShotSTL). Therefore, we propose BacktrackSTL, the first non-iterative online STD algorithm with period-independent $O(1)$ update complexity. It is also robust to outlier, seasonality shift and trend jump because of the combination of outlier-resilient smoothing, non-local seasonal filtering and backtrack technique. Experimentally, BacktrackSTL decomposes a value within $1.6\mu s$, which is 15× faster than the state-of-the-art online algorithm OneShotSTL, while maintaining comparable accuracy to the best offline algorithm RobustSTL. We have also deployed BacktrackSTL on the top of Apache Flink to decompose monitoring metrics in Alibaba Cloud for over a year. Besides, we have open-sourced the artifact of this proposal on GitHub.

## CCS Concepts

• **Information systems** → **Data stream mining**; • **Computer systems organization** → **Real-time systems**.

## Keywords

Online seasonal-trend decomposition; backtrack; streaming computing

**ACM Reference Format:**

Haoyu Wang, Hongke Guo, Zhaoliang Zhu, You Zhang, Yu Zhou, and Xudong Zheng. 2024. BacktrackSTL: Ultra-Fast Online Seasonal-Trend Decomposition with Backtrack Technique. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3671510


*KDD '24, August 25–29, 2024, Barcelona, Spain.*

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3671510

**Table 1: Comparison of different STD algorithms ($T$ is the period length and $I$ is the maximum iterations)**

| Algorithm | Trend Jump | Seasonality Shift | Outlier Tolerance | Online Complexity |
|---|---|---|---|---|
| STL | No | No | No | - |
| TBATS | Yes | No | No | - |
| STR | No | Yes | Yes | - |
| SSA | No | No | No | - |
| RobustSTL | Yes | Yes | Yes | - |
| OnlineSTL | No | No | No | $O(T)$ |
| OneShotSTL | Yes | Yes | No | $O(I)$ |
| **BacktrackSTL** | **Yes** | **Yes** | **Yes** | $\boldsymbol{O(1)}$ |

## 1 Introduction

Many time series exhibit repeating segments, in other words, periodicity. This periodicity can stem from human activities, such as website visits, or from timed behavior, such as machine loads with scheduled tasks. Many methods including frequency domain analysis [24, 33], wavelet analysis [29, 31], and correlation analysis [25, 30] are commonly used in the analysis of periodic time series. Besides, seasonal-trend decomposition (STD) is also widely used. STD decomposes a periodic time series into three components: trend, seasonality, and residual. The decomposition plays a crucial role in downstream time series analysis tasks, such as data repair [32, 34, 39], anomaly detection [15, 21, 38], and forecasting [20, 35].

For leading cloud service providers such as Alibaba Cloud, the challenge of scalability necessitates a focused investment in AIOps technologies including STD. A case in point is Alibaba Cloud's Elastic Compute Service (ECS) [2], which oversees tens of millions of virtual machine instances, generating a vast amount of periodic time series. In such an industrial context, **time efficiency has the highest priority in the design of STD algorithm,** even taking precedence over accuracy.

There are numerous existing works on STD, some of which are compared in Table 1. STD algorithms can be broadly categorized into two groups. The first group consists of offline algorithms, such as STL [16], TBATS [26], STR [17], SSA [22] and RobustSTL [35], which process a batch of values in one go and output their decompositions. These algorithms, however, do not support incremental updates, making their time complexity unacceptable in the scalable

**Table 2: Notations**

| Symbol | Description |
|---|---|
| $y_t$ | Raw value at time $t$ |
| $\tau_t$ | Trend component at time $t$ |
| $s_t$ | Seasonality component at time $t$ |
| $r_t$ | Residual component at time $t$ |
| $T$ | Period length of time series |
| $N$ | Number of points in offline STD algorithm |
| $K$ | Number of considered past neighborhoods |
| $W$ | Constant window size in online STD algorithm, $W = (K+1)T$ in BacktrackSTL |
| $H$ | One-side width of a neighborhood |
| $\delta$ | Standard deviation of seasonal component variations |
| $L$ | Consecutive outlier threshold for trend jump |
| $n$ | Parameter for N-Sigma detection |
| $\hat{y}_t$ | Reference value of $y_t$ |
| $\mathcal{Y}$ | Circular queue for moving average |
| $\mathcal{R}$ | Circular queue for N-Sigma |
| $\lambda_1, \lambda_2$ | Regularization parameters of RobustSTL |
| $I$ | Maximum number of iterations |

scenario. Despite this limitation, these algorithms can achieve high decomposition accuracy. For instance, RobustSTL utilizes $l_1$-norm optimization and non-local seasonal filtering to enhance robustness to trend jumps, seasonal shifts, and outliers.

The second group consists of online algorithms, such as OnlineSTL [27] and OneShotSTL [23], which decompose values incrementally. OnlineSTL employs a simple tri-cube filter and exponential smoothing to extract trend and seasonality; however, its time complexity is $O(T)$, unsuitable for long-period series. On the other hand, OneShotSTL proposes an incremental variant of $l_1$-norm optimization and realizes a period-independent complexity, yet still requires iterative approximation of the optimal solution.

As discussed above, the time efficiency of existing algorithms still need improvements. In Section 3, a detailed analysis of the current approaches is conducted. To fulfill multiple requirements simultaneously, a high-complexity $l_1$-norm optimization is employed. To reduce the complexity further, our insight is that **combining various low-complexity methods may be more effective in addressing complex requirements than using a single high-complexity method**.

Based on the insight, we present BacktrackSTL, a novel online STD algorithm with $O(1)$ time complexity. Specifically, BacktrackSTL incorporates an outlier-resilient smoothing. It combines the strengths of anomaly detection and smoothing, which proficiently manages outliers across a spectrum of severity while extracting the trend. Moreover, non-local seasonal filtering is integrated to capture seasonality and accommodate shifts. Additionally, jump detection reveals the underlying principles obscured by complex optimization objectives, resolving trend jumps via a novel backtrack technique. As a result, BacktrackSTL can decompose a new value from the stream in constant time based on the decomposition results in the sliding window. This efficient computational performance positions BacktrackSTL as a robust solution for real-time periodic time-series analysis in large-scale streaming environments.

Our contributions are summarized as follows:

- To the best of our knowledge, BacktrackSTL is the first non-iterative online seasonal-trend decomposition algorithm with period-independent $O(1)$ time complexity. It is 400× and 15× faster than OnlineSTL and OneShotSTL, respectively, when the period is 12800.
- BacktrackSTL combines outlier-resilient smoothing, non-local seasonal filtering and backtrack technique to achieve the robustness to outlier, seasonality shift and trend jump, respectively. The accuracy of BacktrackSTL is comparable to existing online and offline STD algorithms.
- We have successfully deployed the BacktrackSTL algorithm based on Apache Flink in the production environment of Alibaba Cloud, where it has been used to perform real-time decomposition of monitoring metrics for over a year.

The artifact of BacktrackSTL is open-sourced on GitHub [8]. The frequently used notations are listed in Table 2.

## 2 Preliminary

### 2.1 Decomposition Model

As a traditional problem, seasonal-trend decomposition is defined as follows:

$$y_t = \tau_t + s_t + r_t, \quad 1 \le t \le N \tag{1}$$

where $y_t$ corresponds to the original value at time $t$, $\tau_t$, $s_t$ and $r_t$ is the trend, seasonal component and the residual, respectively.

Usually, the trend $\tau_t$ changes not very fast. However, $\tau_t$ may still have abrupt changes at a low frequency, which is called *trend jump*. It occurs at a very low frequency, typically manifesting once over multiple periods. The seasonality component $s_t$ is a repeated pattern with period $T$. To ensure the uniqueness of decomposition, the sum of all seasonality components in a period is fixed to 0. Formally, $\sum_{i=t}^{t+T-1} s_i = 0$ is satisfied for any $i$. Due to *seasonality shift*, the period varies slightly in the time domain. Besides, the residual $r_t$ can be decomposed further into two parts:

$$r_t = a_t + n_t, \quad 1 \le t \le N \tag{2}$$

where $a_t$ is the *outlier* part and $n_t$ is the white noise. The occurrence of outliers is random.

In short, our assumption is that: the original value $y_t$ can be decomposed into three components, $\tau_t$ with trend jumps, $s_t$ with seasonality shifts and $r_t$ with outliers.

### 2.2 Online Decomposition

In the online scenario, the original values arrive continuously as a stream, while the decomposition proceeds incrementally. For each value $y_t$, we decompose it into trend $\tau_t$, seasonal component $s_t$, and residual $r_t$, just as the model in formula (1). Due to limited memory, the decomposition relies on the recent history, i.e., $\tau_{t-W..t-1}$, $s_{t-W..t-1}$ and $r_{t-W..t-1}$ in a window with constant size $W$.

## 3 Motivation

To the best of our knowledge, RobustSTL is currently the most accurate STD algorithm. In this section, we take RobustSTL as an example to explore potential directions for further enhancing time efficiency.

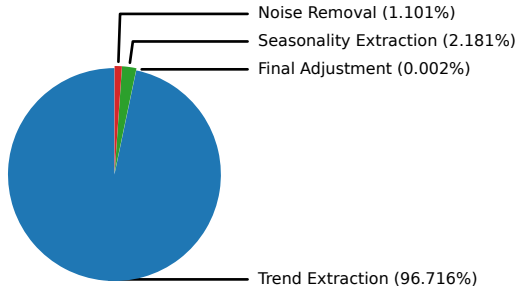
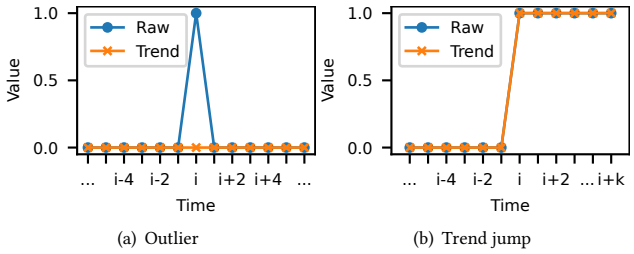



**Figure 1: Time cost breakdown of RobustSTL**




(a) Outlier (b) Trend jump

**Figure 2: Effectiveness of trend extraction with $l_1$-norm**

## 3.1 Analysis on Time Complexity

RobustSTL consists of four stages: noise removal, trend extraction, seasonality extraction, and final adjustment. We evaluate the time cost of each stage and present the results in Figure 1. It is evident that the trend extraction stage consumes the majority of the time, accounting for over 96%. In fact, this stage solves an optimization problem based on the $l_1$-norm. Since $l_1$-norm optimization does not have a closed-form solution, RobustSTL utilizes a computation-intensive numerical method to obtain the solution.

To improve time efficiency further, we identify two promising avenues for exploration. The first one entails the utilization of approximation to lower the computational burden of $l_1$-norm optimization. An illustration of this approach is OneShotSTL, which introduces an incremental variant. Despite this innovation, it still requires iterations for approximated results, consequently its level of complexity optimization remains somewhat inadequate. Conversely, the second one involves investigating a synthesis of low-complexity methods aimed at delivering comparable effectiveness. Owing to its expansive potential on efficiency, we opt for the second avenue for our research endeavors.

## 3.2 Analysis on Effectiveness

Now, let's look at how well $l_1$-norm optimization works. In RobustSTL [35], the objective function of the optimization is a minimum weighted sum function, which is defined as follows:

$$\min_{\tau_{1\dots N}} \sum_{t=T+1}^{N} |(y_t - \tau_t) - (y_{t-T} - \tau_{t-T})| + \lambda_1 \sum_{t=2}^{N} |\tau_t - \tau_{t-1}| + \lambda_2 \sum_{t=3}^{N} |\tau_t - 2\tau_{t-1} + \tau_{t-2}| \tag{3}$$

The first term helps to make the differences between periods smaller after we remove the trends. The subsequent two terms represent the first and second-order differences of the trend, aiming to smooth the trend. As illustrated in the paper of RobustSTL, this optimization exhibits robustness to outliers and trend jumps.

Firstly, consider a simple case about outliers in Figure 2(a). Ignoring all seasonal components for simplicity, there is an outlier at time $i$, e.g., $y_t = 1$ for only $t = i$ and $y_t = 0$ otherwise. Obviously, the objective function formula (3) reaches the minimum when all other $\tau_t = 0$ if $t \neq i$. Therefore, when incorporating all the aforementioned values, $\tau_i$ is the only variable for the following minimum objective function:

$$\min_{\tau_i} 2|1 - \tau_i| + 2\lambda_1|\tau_i| + 4\lambda_2|\tau_i| \tag{4}$$

The solution is as follows:

$$\tau_i = \begin{cases} 0, & \lambda_1 + 2\lambda_2 > 1 \\ 1, & \lambda_1 + 2\lambda_2 \leq 1 \end{cases} \tag{5}$$

Besides, the optimization is also suitable for trend jump. Consider the trend jump (shown by the overlapped blue line) in Figure 2(b), suppose all values after time $i$ are 1 and the last value is at $i + k$. Similarly, calculating the objective function when $\tau$ is jump or not, we find that the extracted trend is the same to the raw value when

$$\lambda_1 + 2\lambda_2 < k + 1 \tag{6}$$

Otherwise, the jump is not detected and the extracted trend stays 0.

In summary, $l_1$-norm-based optimization can handle both outliers and trend jumps, as shown by the orange line in Figure 2. This prompts us to employ two low-complexity methods each tailored to a specific challenge and then integrate them. Motivated by this insight, we apply outlier-resilience smoothing and backtrack technique that is robust against jumps, achieving comparable performance while significantly reducing computational effort.

# 4 BacktrackSTL Decomposition

## 4.1 Overview

Based on the analysis in Section 3.2, we propose a new online decomposition algorithm, BacktrackSTL. Figure 3 provides an overview. Similar to other online algorithms, such as OnlineSTL and OneShotSTL, it consists of two stages: initialization and online update.

For initialization, we can use any offline STD algorithm to decompose the values in the window, including STL and RobustSTL. This stage is conducted only once for a time series. Appendix A introduces the initialization algorithm utilized in BacktrackSTL.

In the online update stage, we maintain a sliding window whose length is an integer multiple of period $T$. This stage involves four steps. In Section 4.2, we use outlier-resilient smoothing to extract the trend with robustness to outliers. In Section 4.3, we employ the non-local seasonal filtering to extract the seasonality component and handle seasonality shifts simultaneously. After that, in Section 4.4, we detect trend jumps based on the idea of concept shift. If a jump is detected, we utilize the backtrack technique in Section 4.5 to correct the past decompositions.

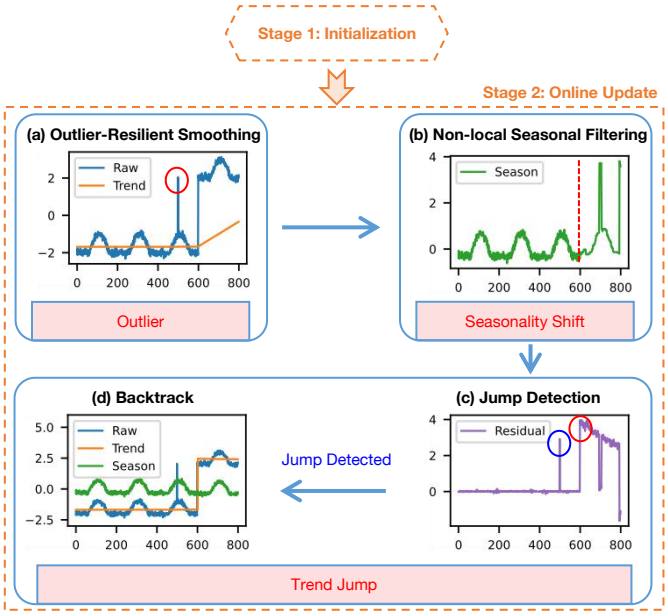



**Figure 3: Overview of BacktrackSTL**

## 4.2 Outlier-Resilient Smoothing

First, let us consider a simple case without outliers. To extract the trend in online update, we employ a moving average [28], as described below:

$$\tau_t = \frac{1}{W}\sum_{i=t-W+1}^{t} y_i = \frac{1}{W}\sum_{i=t-W+1}^{t}(\tau_i + s_i + r_i) \tag{7}$$

Since $W$ is an integer multiple of $T$ and the residual is nearly a white noise, the terms of seasonality and residual can both be removed from formula (7). Furthermore, due to the very slow change of the trend components, formula (7) is a proper approximation.

To address outliers, we employ a dynamic N-Sigma mechanism with parameter $n$ on $s_{t-W\dots t-1}$ to detect outliers. For each $y_t$, we calculate its reference value $\hat{y}_t$. If the difference between $y_t$ and $\hat{y}_t$ is above the N-Sigma threshold, it will be classified as an outlier and $\hat{y}_t$ will be used in moving average instead. Additionally, if the difference is below the threshold, moving average can effectively smooth them. Therefore, this method is called *outlier-resilient smoothing*.

When calculating the reference value $\hat{y}_t$, we estimate the trend component with $\tau_{t-1}$. For seasonality, $K$ past neighborhoods of decomposed seasonality, centered at $s_{t-KT}, \cdots, s_{t-T}$ with width $H$, are considered. Specially, each neighborhood contains $2H+1$ components, e.g., $s_{t-T-H}, \cdots, s_{t-T}, \cdots, s_{t-T+H}$ belong to the neighborhood centered at $s_{t-T}$. The one closest to $y_t - \tau_{t-1}$ is selected. Formally, the equation is as follows:

$$\hat{y}_t = \tau_{t-1} + \arg\min_{s_i, i\in\Omega} |s_i - (y_t - \tau_{t-1})| \tag{8}$$

$$\Omega = \{i | (t' = t - kT, i = t' \pm h)\} \tag{9}$$
$$k = 1, 2, \dots, K; h = 0, 1, \dots, H$$

In the implementation, we maintain two circular queues $\mathcal{Y}$ and $\mathcal{R}$, both of length $W$, which includes all $y_i$ (or $\hat{y}_i$ if outlier detected) and $s_i$ in the sliding window, respectively. For $\mathcal{Y}$, we maintain the size $|\mathcal{Y}|$ and the sum $\sum_{y_i\in\mathcal{Y}} y_i$ as the inner state variable, and update them whenever the elements in $\mathcal{Y}$ are changed. As a result, the operation complexities of addition, removal and moving average are all $O(1)$. Similarly, N-Sigma detection with $\mathcal{R}$ can also be completed in $O(1)$. Algorithm 1 shows the main procedure.

**Algorithm 1:** Outlier Resilient Smoothing

**Data:** $y_{t-W\dots t}, \tau_{t-W\dots t-1}, s_{t-W\dots t-1}, r_{t-W\dots t-1}, \mathcal{Y}, \mathcal{R}$

Obtain $\hat{y}_t$ according to formula (8)-(9) ;
$detected \leftarrow NSigma(y_t - \hat{y}_t)$;
**if** *detected* **then**
  Add $\hat{y}_t$ to $\mathcal{Y}$;
**else**
  Add $y_t$ to $\mathcal{Y}$;
$\tau_t \leftarrow mean(\mathcal{Y})$;
**return** $\tau_t, detected$

Figure 3(a) illustrates the trend extracted by outlier-resilient smoothing. For clarity, we present the results for all data points, though the decomposition is performed one by one. Neither outliers (encircled in red) nor noise significantly impact the accuracy of the trend extraction. However, smoothing alone is unable to extract trend accurately after jump, thus the supports of jump detection in Section 4.4 and backtrack in Section 4.5 are needed.

## 4.3 Non-local Seasonal Filtering

To extract the seasonality component, we directly utilize the non-local seasonal filtering proposed by RobustSTL [35], which is robust to seasonality shifts. It is weighted average of neighborhoods $\Omega$ shown in formula (9), which is defined as follows:

$$s_t = \sum_{j\in\Omega} w_j^t y'_j \tag{10}$$

$$w_j^t = \frac{1}{z}\exp\{-\frac{(j-t')^2}{2H^2} - \frac{(y'_j - y'_t)^2}{2\delta^2}\} \tag{11}$$

$$y'_j = y_j - \tau_j \tag{12}$$

where $z$ is the normalization factor, $t'$ is the center of the neighborhood $j$ belongs to and $\delta$ is a parameter which controls how different the seasonal components are in various periods. Since the weights $w_j^t$ are given by two Gaussian functions. The component close to neighborhood center $t'$ and de-trend value $y'_t$ is large.

Figure 3(b) illustrates the seasonality extracted by non-local seasonal filtering. Similarly, the accurate seasonality turns to inaccurate after the jump.

## 4.4 Jump Detection

Before discussing the specific method for jump detection, let us first consider a simple example illustrated in Figure 4 to demonstrate the challenge of jump detection. Suppose that the seasonality components have been removed, and at time $i$, we observe that $y_i = 1$, which deviates clearly from the normal trend. However, we still cannot decide whether it is an outlier (following the blue line in the future) or a trend jump (following the orange line). Since we

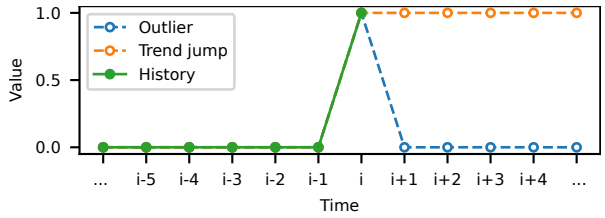



**Figure 4: Challenge of jump detection**

are unable to predict the future, **a delayed decision is naturally embedded in the online scenario**.

As shown in Figure 3(c), a trend jump leads to consecutive high residuals (encircled in red), which is greatly different from the outlier (encircled in blue). We employ the concept drift detection idea to detect it. If an extremely rare event occurs under the given model or assumption, it suggests the presence of a concept drift problem. Specifically, we introduce a parameter $L$, and assume that there is no trend jump with outlier probability $p$. We classify all deviated values as outliers. When the number of consecutive outliers exceeds $L$, i.e., the probability $p^k$ is less than the threshold $p^L$, we consider that an extreme event has taken place, resulting in a violated assumption and a detected trend jump.

In fact, referring to formula (6), the optimization objective of RobustSTL also implies a similar judgment. Only after detecting $\lceil \lambda_1 + 2\lambda_2 \rceil$ consecutive deviations can it be determined as a jump and provide the correct decomposition. Compared with the regularization parameter in RobustSTL, consecutive outlier threshold $L$ is more intuitive and easier to set.

## 4.5 Backtrack

When a trend jump is detected at time $t$, all values after the jump, i.e., $y_{t-L+1\ldots t}$, have been decomposed in the wrong way due to the detection delay. Since each decomposition relies on the previous ones, a backtrack is necessary to correct them.

Specifically, we still use the average method to estimate the trend. Considering that the seasonality components are similar between different periods, i.e., $s_t \approx s_{t-T}$, we estimate all trends $\tau_{t-L+1\ldots t}$ as a constant $\overline{\tau}$:

$$\overline{\tau} = \frac{1}{L} \sum_{i=t-L+1}^{t} y_i - s_{i-T} \tag{13}$$

Then, we use non-local seasonal filtering in formula (10)-(12) to extract the seasonality components $s_{t-L+1\ldots t}$.

Besides, there is a noticeable difference in values before and after the jump, which can result in an incorrect result for the moving average. To address it, we make a compensation by adding the gap $\overline{\tau} - \tau_{t-L}$ to all the elements before jump in $\mathcal{Y}$. The main procedure of backtrack is shown in Algorithm 2. Figure 3(d) shows the correct decompositions when backtrack is applied.

*Discussion on Periodic Context Disruption.* In practice, the seasonal component of a series may change significantly, called periodic context disruption. For example, when an API is transformed from private test to public release, the series of its request count may meet such a disruption. This situation violates the outlier-only assumption in Section 4.4, which may be misjudged as a jump, triggering a backtrack. Therefore, if the majorities of residuals after backtrack still violate the N-Sigma constraint, a periodic context disruption is suggested. It is necessary to discard all values and regard the series as a new one. Because periodic context disruption is out of the scope of the model in Section 2, we leave it as part of our future work.

**Algorithm 2:** Backtrack

**Data:** $y_{t-W\ldots t}, \tau_{t-W\ldots t}, s_{t-W\ldots t}, r_{t-W\ldots t}, \mathcal{Y}, \mathcal{R}$

$\tau_{t-L+1\ldots t} \leftarrow mean(y_{t-L+1\ldots t} - s_{t-T-L+1\ldots t-T})$;
Estimate $s_{t-L+1\ldots t}$ according to formula (10)-(12) ;
$r_{t-L+1\ldots t} \leftarrow y_{t-L+1\ldots t} - \tau_{t-L+1\ldots t} - s_{t-L+1\ldots t}$;
Update $y_{t-L+1\ldots t}$ and compensate other elements in $\mathcal{Y}$;
Update $r_{t-L+1\ldots t}$ in $\mathcal{R}$ ;

**Algorithm 3:** BacktrackSTL, Online Update

**Data:** $y_{t-W\ldots t}, \tau_{t-W\ldots t-1}, s_{t-W\ldots t-1}, r_{t-W\ldots t-1}, \mathcal{Y}, \mathcal{R}$

Obtain $\tau_t$ and *detected* with outlier-resilient smoothing;
Estimate $s_t$ according to formula (10)-(12) ;
$r_t \leftarrow y_t - \tau_t - s_t$;
$anomalyCnt \leftarrow detected \ ? \ anomalyCnt + 1 : 0$;
**if** $anomalyCnt \geq L$ **then**
    Backtrack the decompositions of last $L$ values;
    $anomalyCnt \leftarrow 0$;
**return** $\tau_t, s_t, r_t$

## 4.6 Summary

In this section, we introduce an online decomposition algorithm called BacktrackSTL. The sliding window length $W = (K+1)T$, since it is the minimum length containing all considered past neighborhoods. Algorithm 3 shows the online update procedure of BacktrackSTL.

Proposition 4.1 (Update complexity). *In online update stage, BacktrackSTL updates the decomposition of a single value within amortized $O(1)$ time complexity.*

Proof. There are four steps of BacktrackSTL update.

*Outlier-resilient smoothing:* First, we calculate the reference value by traversing all values in $K$ neighborhoods with $O(KH)$ complexity. Second, N-Sigma detection costs $O(1)$ time since useful inner state variables are precomputed. Third, the complexity of updating $\mathcal{Y}$ and getting average is $O(1)$ due to precomputed states as well. Thus, the total complexity of outlier-resilient smoothing is $O(KH)$.

*Non-local seasonal filtering:* This step is a weighted linear combination of $K$ neighborhoods. Thus, the computational complexity is $O(KH)$.

*Jump detection:* This step is a comparison with complexity $O(1)$.

*Backtrack:* First, calculating $\overline{\tau}$ needs $O(L)$ time. Second, since the computational complexity of a non-local seasonal filtering is $O(KH)$, extracting the seasonality of $L$ values costs $O(KHL)$ time. Then, all elements in $\mathcal{Y}$ need update or compensation, which costs $O(W)$ time. Thus, the total complexity of backtrack is $O(W + KHL)$.

Finally, since $K$ and $H$ are both fixed constant parameters, the complexities of outlier-resilient smoothing and non-local seasonal filtering can be considered as $O(1)$. Similarly, also due to the constant $L$ threshold, the complexity of backtrack is $O(W)$. Considering the rarity of jumps, e.g., lower than $\frac{1}{W}$, its amortized complexity is still $O(1)$. Therefore, the total complexity of BacktrackSTL is $O(1)$, independent of the period length $T$. □

## 5 Deployment

Stability is crucial for cloud companies. In Alibaba Cloud, ECS anomaly scheduling platform is built to monitor metrics and execute operational workflows such as restarting and migrating. According to our statistics, billions of metric series are monitored and approximately 20.8% of them are seasonal. To handle these sequence more effectively, the platform employs the BacktrackSTL algorithm for decomposition.

In ECS anomaly scheduling platform, period identification is operationalized as an offline process, distinct from the online decomposition. Executed within MaxCompute [11], the period identification algorithm is implemented as a user-defined function (UDF), which is harnessed daily to calculate the period length of all monitoring metric series. The results are then synced to the cloud-native memory database Tair [14].

For online decomposition, BacktrackSTL consumes metric series from Simple Log Service (SLS) [5], retrieves associated period length from Tair, and leaves its decomposition results for downstream tasks, e.g., anomaly detection. Considering the independence between series, it is deployed on Apache Flink [1] due to its horizontal scalability. Specifically, it is implemented as a keyed function with states, whose parameters are obtained from a broadcast stream. The task is hosted on Ververica Platform (VVP) [4]. According to the tests, each compute unit (CU)[1] can support about 127K throughput per second, greatly reducing the machine cost.

On ECS anomaly scheduling platform, BacktrackSTL has been continuously and stably running for over one year. During this time, it helped the platform discover some real problems. For example, the release of a control software may bring unexpected computational overhead on specific machines, leading to the feature of high utilization on a certain CPU core. As shown in Figure 7(a), by decomposing the count of node controllers with this feature, we discovered this trend jump and timely notified the software owner.

## 6 Experimental Evaluation

### 6.1 Experiment Setup

*6.1.1 Dataset.* In this section, we conduct evaluations using three datasets, including one synthetic dataset and two real datasets. Figure 5 displays the synthetic dataset named SYNTHETIC, which has a period of 200, with four trend jumps and a severe outlier. Additionally, its seasonality components experience shifts with a maximum of 5. As shown in Figure 6 and Figure 7, the two real datasets are referred to as REAL1 and REAL2, which represent the counts of logs with specific features from Alibaba Cloud. Their period lengths are both 24.

[1]CU is the resource unit of VVP. One CU is equal to 1 CPU core, 4 GiB of memory, and 20 GB of local storage.

*6.1.2 Baselines.* Refer to Table 1, we compare the proposed BacktrackSTL with existing STD algorithms. The offline algorithms include STL [16], SSA [22], TBATS [26] and RobustSTL [35, 37]. STR [17] is not included due to its extremely high complexity. Based on a sliding window, these offline algorithms can be applied to online scenarios, named Window-STL, Window-SSA, Window-TBATS and Window-RobustSTL. Additionally, Online-RobustSTL, an online variant of RobustSTL, and the native online algorithms OnlineSTL [27] and OneShotSTL [23] are used for comparison as well.

In the experiment, we implement BacktrackSTL and reproduce OnlineSTL and SSA faithfully in Java. We also use the public Java implementations of STL [7] and OneShotSTL [3]. Moreover, since other baselines have no public Java implementation and are difficult to reproduce because of dependencies, we have to use their public implementations in other languages. For example, the public repository SREWorks [6] provides the Python implementations of RobustSTL and its variants while R package *forecast* [12] provides the implementation of TBATS.

*6.1.3 Hyper-Parameters.* All STD algorithms require the period length $T$ as a parameter. For generated datasets, without special instructions, we always use the ground truth value, i.e., $T = 200$. For real datasets, RobustPeriod [36] is used to estimate their period lengths.

For BacktrackSTL, we set $K = 2$, $L = 4$ and $n = 6$ for all datasets. Meanwhile, we set $H = 5$ for the generated dataset and $H = 2$ for the real ones. As for $\delta$, we automatically determine it from the initialization data. For each $y_i$, we calculate its minimum distance to its previous neighborhood, i.e., $\min |y_i - y_j|$ when $i - T - H \leq j \leq i - T + H$. Their standard deviation is used as the value of $\delta$.

RobustSTL and its variants have dozens of parameters. For those parameters with similar meanings to BacktrackSTL, we adopt the same settings, such as $K$ and $H$. Meanwhile, we set $\delta_i$ and $\delta_d$ to $H$ and $\delta$ in BacktrackSTL, respectively. The default values are used for the remaining parameters.

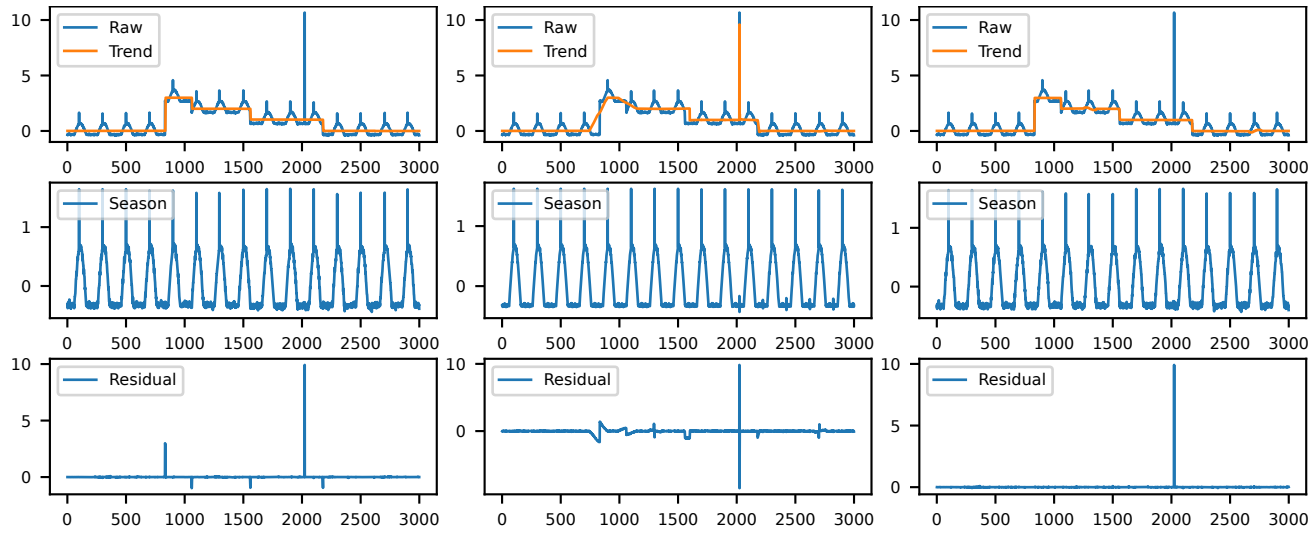
Besides, STL does not require any parameters to be set. The smoothing parameter $\gamma$ of OnlineSTL is set to the recommended value of 0.7 by the author. $H$ in OneShotSTL is consistent with BacktrackSTL, while the other parameters are the default values or determined automatically using the author-provided method.

*6.1.4 Environment.* The experimental evaluations are conducted on an ECS (ecs.re4.10xlarge, 40 vCPU cores, 480 GiB memories) from Alibaba Cloud. The operation system is 64-bit CentOS 7.9. All implementations are run on JDK 1.8.0_382 or Python 3.6.8.

### 6.2 Evaluation on Accuracy

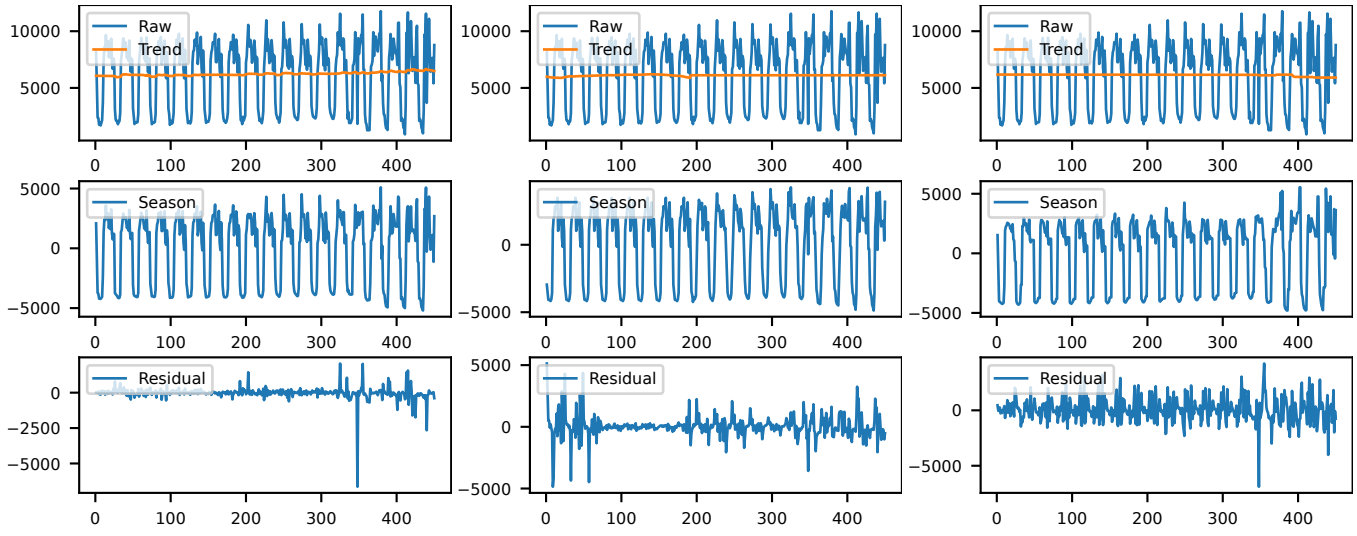
We first evaluate the accuracy of the decomposition algorithms. In line with existing work like RobustSTL and OneShotSTL, we can only provide quantitative results for synthetic datasets due to the absence of decomposition ground truth in real datasets. Meanwhile, visual results for all datasets are provided. Besides, to emphasize the generalizability of BacktrackSTL, results for more real datasets are shown in Appendix B.2.

Figure 5 visually displays the decomposition results of BacktrackSTL, RobustSTL and OneShotSTL over SYNTHETIC. As shown in Figure 5(a), BacktrackSTL shows a similar result to RobustSTL in (c),




(a) BacktrackSTL (b) OneShotSTL (c) RobustSTL

**Figure 5: Decomposition results on dataset SYNTHETIC**

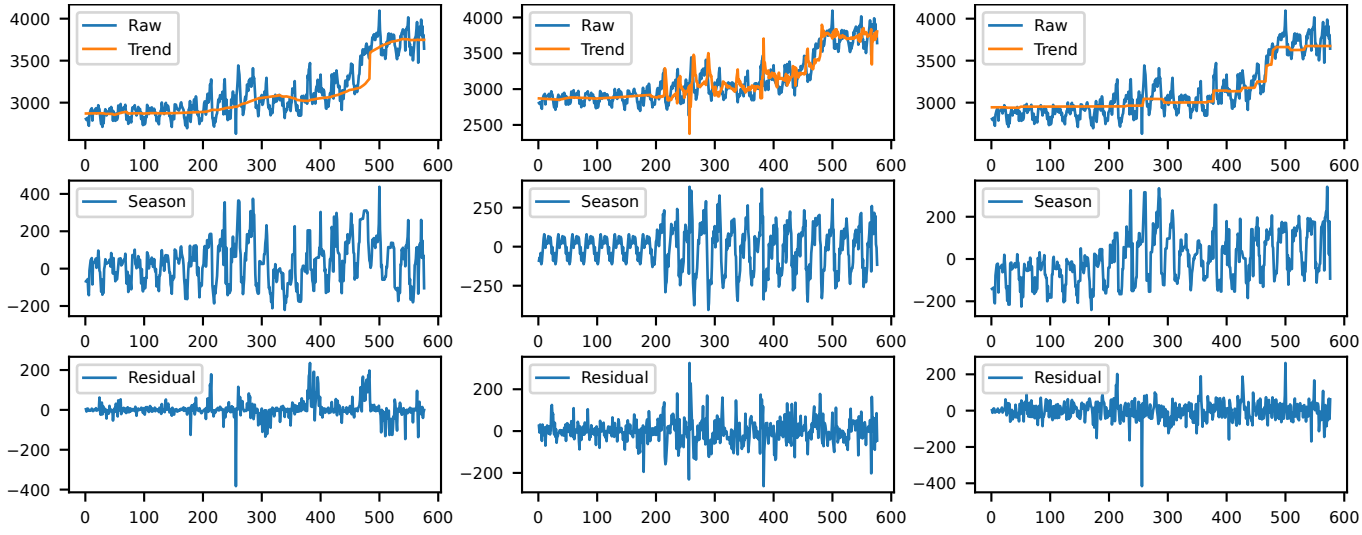



(a) BacktrackSTL on REAL1 (b) OneShotSTL on REAL1 (c) RobustSTL on REAL1

**Figure 6: Decomposition results on dataset REAL1**

capturing trend jumps and outliers well, while tolerating seasonality shift. It is worth noting that the figures of online algorithms are drawn using the decomposition results without delay. Therefore, in (a) and (b), outlier in the residual near each detected jump attests to the inherent algorithm-independent presence of latency within online scenarios. In contrast, this occurrence is notably absent for offline RobustSTL in (c). Besides, for OneShotSTL in (b), due to the $l_2$-norm of the residual term in the optimization objective, it cannot handle severe outliers well. At the same time, it also fails to capture trend jumps, because the automatically determined regular parameters are not suitable. Moreover, Table 3 shows the mean absolute error (MAE) between the decomposition results and the ground truth. The performance of BacktrackSTL is comparable to RobustSTL, much better than other online algorithms.

The visual results of two real datasets are presented in Figure 6-7. Similar to the results of SYNTHETIC, the decomposition of BacktrackSTL is comparable to that of RobustSTL, and significantly outperforms OneShotSTL. Specifically, as shown in Figure 6(a), the dip at time 348 is left in the residual, which makes it easy to be detected in downstream anomaly detection tasks. However, since




(a) BacktrackSTL on REAL2 (b) OneShotSTL on REAL2 (c) RobustSTL on REAL2

**Figure 7: Decomposition results on dataset REAL2**

**Table 3: Decomposition comparison over SYNTHETIC**

| Algorithm | Type | Trend MAE | Seasonality MAE |
|---|---|---|---|
| STL | Offline | 0.085 | **0.017** |
| SSA | Offline | 0.169 | 0.152 |
| TBATS | Offline | 0.066 | 0.064 |
| RobustSTL | Offline | **0.010** | 0.027 |
| Window-STL | Online | 0.165 | 0.066 |
| Window-SSA | Online | 0.444 | 0.426 |
| Window-TBATS | Online | 0.339 | 0.115 |
| Window-RobustSTL | Online | 0.071 | 0.030 |
| Online-RobustSTL | Online | 0.073 | 0.030 |
| OnlineSTL | Online | 0.368 | 0.303 |
| OneShotSTL | Online | 0.150 | 0.077 |
| BacktrackSTL | Online | **0.012** | **0.023** |

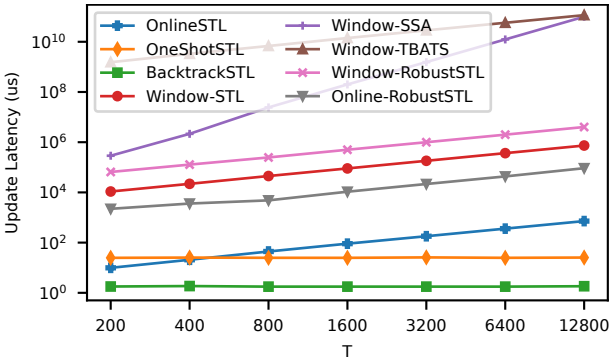



**Figure 8: Comparison on update latency**

the optimization objective of OneShotSTL does not sufficiently tolerate the fluctuation of seasonality, there are still some periodic components in the residuals in Figure 6(b). Additionally, at time 0-300 in Figure 7(a), BacktrackSTL effectively decomposes the series with a smoothly rising trend. Outlier-resilient smoothing regards the moving average without outliers as the extracted trend, thereby accurately capturing the smooth ascent of the series.

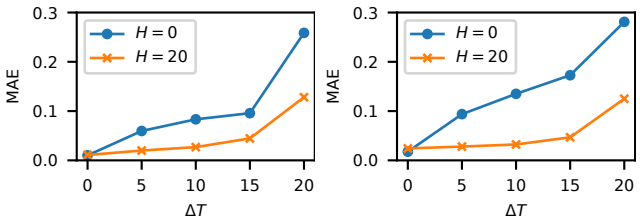
## 6.3 Evaluation on Time Efficiency

Next, we evaluate the time efficiency of the algorithms. We extend the dataset SYNTHETIC to obtain a sufficient long time series. Since the time complexity of most algorithms is related to the period length $T$, Figure 8 shows the online update latency of a single value with respect to $T$, where $T$ takes values from 200 to 12800.

The latency of BacktrackSTL is approximately 1.6$\mu s$ per value, which represents a $10^3 - 10^{11}\times$ improvement over offline algorithms' online variants, such as Window-STL, etc. This substantial improvement is chiefly attributable to the inherent algorithmic time complexities rather than variations across programming languages. Meanwhile, the latency is independent of the period length, which is consistent with our theoretical complexity of $O(1)$. Compared to OneShotSTL with complexity of $O(I)$, BacktrackSTL achieves about 15× improvement because it does not require iteration (maximum iterations $I = 8$ for OneShotSTL) and has a smaller constant complexity. Additionally, it is also significantly faster than OnlineSTL with update complexity $O(T)$, 5× when $T = 200$ and 400× when $T = 12800$.

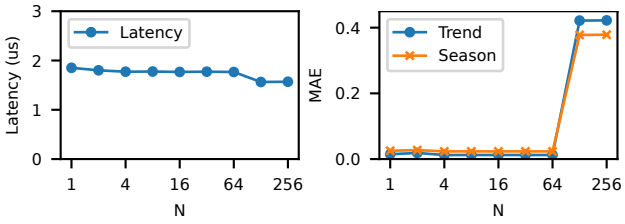
## 6.4 Evaluation on Robustness

The period length $T$ is an important parameter of BacktrackSTL. For real datasets, however, the value of $T$ discovered by algorithms may




(a) Trend MAE (b) Seasonality MAE

**Figure 9: Robustness on period length**




(a) Latency (b) MAE

**Figure 10: Influence of N-Sigma**

not always be true. Thus, we evaluate the tolerance of BacktrackSTL to errors in $T$. Specially, we add a period length error $\Delta T$ to $T$ and evaluate its MAE for the trend and seasonality. The values of $\Delta T$ are set to $\{0, 5, 10, 15, 20\}$.

Figure 9 shows the experimental results. Without extra processing, i.e., $H = 0$, the MAE of trend and seasonality increase with $\Delta T$ as shown by the blue line. Actually, the impact of an incorrect period length $T$ can be weakened by a proper neighborhood width $H$. If the width is large enough to cover the period error, e.g., $H = 20$, the MAE significantly decrease, as shown by the orange line.

## 6.5 Influence of N-Sigma Strategy

In outlier-resilient smoothing in Section 4.2, we employ the N-sigma strategy for outlier detection. Consequently, $N$ serves as a critical parameter in distinguishing noise from outliers. Thus, To explore the impact of varying $N$, we perform an evaluation of the average update latency and the trend/seasonality MAE with varying $K$. This evaluation is conducted on the SYNTHETIC dataset. The results are shown in Figure 10 with logarithmic-scaled horizontal axis.

As the parameter $N$ increases, a greater number of outliers are classified as noise, leading to a reduction in the number of detected jumps. This, in turn, results in a lesser number of backtracks and marginally reduces latency, as depicted in (a). Concerning the trend/seasonality MAE depicted in (b), we observe that both excessively small and large values of $N$ yield a detrimental effect. However, the performance retains robustness over a substantially wide range of $N$, suggesting that the parameter is not difficult to configure. In light of the commonly adopted N-sigma strategy and the infrequency of outliers, we set $N = 6$ uniformly in the proposal.

## 7 Related Work

STD has been the subject of extensive research for several decades. Among them, STL [16] is one of the most famous algorithms, which utilizes local regression (LOESS) smoothing to extract trend and seasonality, iterating until convergence. Meanwhile, SSA [22] folds the time series into a matrix based on the period length and employs singular value decomposition (SVD) to extract the seasonality. Moreover, TBATS [26] builds a state space model for time series and solves it with maximum likelihood estimation (MLE), thereby providing confidence intervals for the results. STR [17] combines trend and seasonality into a joint optimization function, while its variant Robust-STR [17] further enhances the algorithm's robustness by introducing the $l_1$-norm. After that, RobustSTL [35] is proposed, which utilizes $l_1$-norm optimization to extract trend in the presence of trend jumps and outliers, and employs non-local seasonal filtering to extract seasonality with shifts. Furthermore, Fast RobustSTL [37] extends RobustSTL to support multiple seasonality and speeds up it with ADMM algorithm.

In recent years, researchers have focused more on the online scenario. The most straightforward online strategy is to run the above offline algorithms within a sliding window. However, the time cost is extremely high. For instance, the time complexity of each decomposition is $O(IW^2)$ for RobustSTL on the sliding window.

To address the efficiency issue, several native online STD algorithms have been proposed. OnlineSTL [27] is the first online algorithm, which decomposes time series 100× faster than traditional STL. However, its complexity is still dependent with period length, making it less effective with long-period time series. On the other hand, OneShotSTL [23] calculates the trend and seasonality using a joint optimization function and utilizes linear systems to approximate the solution. However, this approximation still requires iterations to approach the optimal solution, leaving space for optimization in terms of time efficiency. Different from the above algorithms, BacktrackSTL combines period-insensitive steps such as outlier-resilient smoothing and non-local seasonal filtering, achieving much lower update latency without iterations.

## 8 Conclusion

In this paper, we introduce BacktrackSTL, a novel seasonal-trend decomposition algorithm with $O(1)$ time complexity. Our investigation highlights that the main bottleneck for RobustSTL in terms of time efficiency is the high-complexity $l_1$-norm optimization though it is robust to outliers and trend jumps. Therefore, we combine outlier-resilient smoothing and backtrack strategy to replace the optimization, and inherit non-local seasonal filtering, resulting in a significant improvement in time efficiency while still addressing trend jumps, seasonality shifts, and outliers. Our experimental results demonstrate that our algorithm BacktrackSTL decomposes a value within 1.6$\mu s$, which is 15× faster than state-of-the-art online algorithms.

## Acknowledgments

The authors would like to thank all reviewers and chairs for their helpful comments, and Prof. Shaoxu Song and Dr. Chengguang Fang for their encouragements and helps. Haoyu Wang (https://wanghy.pages.dev/) is the corresponding author.

## References


[1] 2023. Apache Flink. https://flink.apache.org/

[2] 2023. Elastic Compute Service (ECS). https://www.alibabacloud.com/help/en/ecs/

[3] 2023. OneShotSTL. https://github.com/xiao-he/OneShotSTL

[4] 2023. Realtime Compute for Apache Flink. https://www.alibabacloud.com/help/en/flink/

[5] 2023. Simple Log Service (SLS). https://www.alibabacloud.com/help/en/sls/

[6] 2023. SREWorks. https://github.com/alibaba/SREWorks/

[7] 2023. STL. https://github.com/ServiceNow/stl-decomp-4j

[8] 2024. Artifact of BacktrackSTL. https://github.com/543202718/BacktrackSTL

[9] 2024. Daily Minimum Temperatures in Melbourne. https://www.kaggle.com/datasets/samfaraday/daily-minimum-temperatures-in-me

[10] 2024. Daily total female births in California, 1959. https://www.kaggle.com/datasets/dougcresswell/daily-total-female-births-in-california-1959

[11] 2024. MaxCompute. https://www.aliyun.com/product/odps

[12] 2024. R package foreacst. https://cran.r-project.org/web/packages/forecast/index.html

[13] 2024. Sunspots. https://www.kaggle.com/datasets/robervalt/sunspots

[14] 2024. Tair. https://www.aliyun.com/product/apsaradb/kvstore/tair

[15] Paul Boniol, Michele Linardi, Federico Roncallo, Themis Palpanas, Mohammed Meftah, and Emmanuel Remy. 2021. Unsupervised and scalable subsequence anomaly detection in large data series. *VLDB J.* 30, 6 (2021), 909–931. https://doi.org/10.1007/s00778-021-00655-8

[16] Robert B Cleveland, William S Cleveland, Jean E McRae, and Irma Terpenning. 1990. STL: A seasonal-trend decomposition. *J. Off. Stat* 6, 1 (1990), 3–73.

[17] Alexander Dokumentov, Rob J Hyndman, et al. 2015. STR: A seasonal-trend decomposition procedure based on regression. *Monash econometrics and business statistics working papers* 13, 15 (2015), 2015–13.

[18] Hadi Fanaee-T. 2013. Bike Sharing Dataset. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5W894.

[19] Hadi Fanaee-T and Joao Gama. 2013. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence* (2013), 1–15. https://doi.org/10.1007/s13748-013-0040-3

[20] Valentin Flunkert, David Salinas, and Jan Gasthaus. 2017. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *CoRR* abs/1704.04110 (2017). arXiv:1704.04110 http://arxiv.org/abs/1704.04110

[21] Jingkun Gao, Xiaomin Song, Qingsong Wen, Pichao Wang, Liang Sun, and Huan Xu. 2020. RobustTAD: Robust Time Series Anomaly Detection via Decomposition and Convolutional Neural Networks. *CoRR* abs/2002.09545 (2020). arXiv:2002.09545 https://arxiv.org/abs/2002.09545

[22] Nina Golyandina and E Osipov. 2007. The "Caterpillar"-SSA method for analysis of time series with missing values. *Journal of Statistical planning and Inference* 137, 8 (2007), 2642–2653.

[23] Xiao He, Ye Li, Jian Tan, Bin Wu, and Feifei Li. 2023. OneShotSTL: One-Shot Seasonal-Trend Decomposition For Online Time Series Anomaly Detection And Forecasting. *Proc. VLDB Endow.* 16, 6 (2023), 1399–1412. https://www.vldb.org/pvldb/vol16/p1399-he.pdf

[24] Scott H Holan and Nalini Ravishanker. 2018. Time series clustering and classification via frequency domain methods. *Wiley Interdisciplinary Reviews: Computational Statistics* 10, 6 (2018), e1444.

[25] Davor Horvatic, H Eugene Stanley, and Boris Podobnik. 2011. Detrended cross-correlation analysis for non-stationary time series with periodic trends. *Europhysics Letters* 94, 1 (2011), 18007.

[26] Alysha M De Livera and Rob J Hyndman. 2011. Forecasting time series with complex seasonal patterns using exponential smoothing. *Monash Econometrics & Business Statistics Working Papers* 106, 496 (2011), 1513–1527.

[27] Abhinav Mishra, Ram Sriharsha, and Sichen Zhong. 2022. OnlineSTL: Scaling Time Series Decomposition by 100x. *Proc. VLDB Endow.* 15, 7 (2022), 1417–1425. https://www.vldb.org/pvldb/vol15/p1417-mishra.pdf

[28] Denise R. Osborn. 1995. Moving Average Detrending and the Analysis of Business Cycles. *Oxford Bulletin of Economics and Statistics* 57, 4 (1995), 547–558.

[29] Donald B Percival and Andrew T Walden. 2000. *Wavelet methods for time series analysis*. Vol. 4. Cambridge university press.

[30] Kira Rehfeld, Norbert Marwan, Jobst Heitzig, and Jürgen Kurths. 2011. Comparison of correlation analysis techniques for irregularly sampled time series. *Nonlinear Processes in Geophysics* 18, 3 (2011), 389–404.

[31] Manel Rhif, Ali Ben Abbes, Imed Riadh Farah, Beatriz Martínez, and Yanfang Sang. 2019. Wavelet transform application for/in non-stationary time-series analysis: A review. *Applied Sciences* 9, 7 (2019), 1345.

[32] Shaoxu Song, Aoqian Zhang, Jianmin Wang, and Philip S. Yu. 2015. SCREEN: Stream Data Cleaning under Speed Constraints. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives (Eds.). ACM, 827–841. https://doi.org/10.1145/2723372.2723730

[33] Haoyu Wang and Shaoxu Song. 2022. Frequency Domain Data Encoding in Apache IoTDB. *Proc. VLDB Endow.* 16, 2 (2022), 282–290. https://doi.org/10.14778/3565816.3565829

[34] Haoyu Wang, Aoqian Zhang, Shaoxu Song, and Jianmin Wang. 2023. Streaming data cleaning based on speed change. *VLDB J.* (2023). https://doi.org/10.1007/s00778-023-00796-y

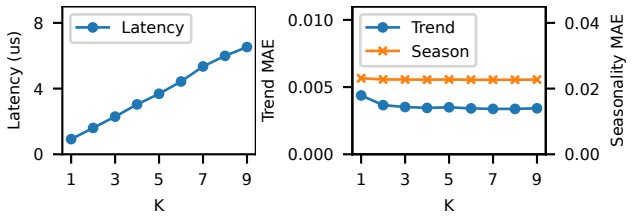
[35] Qingsong Wen, Jingkun Gao, Xiaomin Song, Liang Sun, Huan Xu, and Shenghuo Zhu. 2019. RobustSTL: A Robust Seasonal-Trend Decomposition Algorithm for Long Time Series. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 5409–5416. https://doi.org/10.1609/aaai.v33i01.33015409

[36] Qingsong Wen, Kai He, Liang Sun, Yingying Zhang, Min Ke, and Huan Xu. 2021. RobustPeriod: Robust Time-Frequency Mining for Multiple Periodicity Detection. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2328–2337. https://doi.org/10.1145/3448016.3452779

[37] Qingsong Wen, Zhe Zhang, Yan Li, and Liang Sun. 2020. Fast RobustSTL: Efficient and Robust Seasonal-Trend Decomposition for Time Series with Complex Patterns. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 2203–2213. https://doi.org/10.1145/3394486.3403271

[38] Jiehui Xu, Haixu Wu, Jianmin Wang, and Mingsheng Long. 2022. Anomaly Transformer: Time Series Anomaly Detection with Association Discrepancy. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net. https://openreview.net/forum?id=LzQQ89U1qm_

[39] Aoqian Zhang, Shaoxu Song, Jianmin Wang, and Philip S. Yu. 2017. Time Series Data Cleaning: From Anomaly Detection to Anomaly Repairing. *Proc. VLDB Endow.* 10, 10 (2017), 1046–1057. https://doi.org/10.14778/3115404.3115410




(a) Latency (b) MAE

**Figure 11: Influence of window size $W = (k+1)T$**

## A BacktrackSTL Initialization

We use a simple offline algorithm in initialization with the first $W = (K+1)T$ values. The first step is jump detection. We calculate a new sequence $d_{T+1\ldots W-T+1}$ as follows:

$$d_i = \frac{1}{T}\Big(\sum_{j=i}^{i+T-1} y_j - \sum_{j=i-T}^{i-1} y_j\Big), \quad T+1 \le i \le W-T+1 \tag{14}$$

Subsequently, we identify the local maximum/minimum values in the sequence as candidate jump points. For a candidate point $d_i$, if the difference before and after it exceeds the N-Sigma constraint, it is classified as a jump point. Formally, a jump point is a candidate which satisfies the following formula:

$$|d_i| > n * \max\{std(y_{i\ldots i+T-1}), std(y_{i-T\ldots i-1})\} \tag{15}$$

Suppose we have $m$ jump points, donated as $P_1, P_2, \ldots, P_m$. Let $P_0 = 0$ and $P_{m+1} = W+1$, we divide the sequence into $m+1$ segments based on the jump points. The $k$-th segment starts from index $P_{k-1}$ and ends at index $P_k - 1$, formulated as $y_{P_{k-1}\ldots P_k-1}$.

After that, we calculate the moving average of length $T$ as the trend for each segment. Specifically, if the segment is shorter than $T$, i.e., $P_k - P_{k-1} < T$, the estimated trend term for any point $t$ within the segment is given by

$$\tau_t = mean(y_{P_{k-1}\ldots P_k-1}) \tag{16}$$

Otherwise, the trend is as follows:

$$\tau_t = \begin{cases} mean(y_{t\ldots t+T-1}) & if\ t+T \le P_k \\ mean(y_{P_k-T\ldots P_k-1}) & if\ t+T > P_k \end{cases} \tag{17}$$

Next, we employ the non-local seasonal filtering in Section 4.3 to calculate the seasonality component. If the neighborhood $\Omega$ is empty, we directly regard the de-trend value $y'_t$ as the seasonal component. Finally, we obtain the residual by subtracting the trend and seasonal component from the original values.

## B Additional Experiments

### B.1 Influence of Window Size

For online STD algorithms, the choice of window size $W$ is a pivotal factor that influences both the accuracy and time efficiency of the approach. Within the BacktrackSTL framework, $W$ is defined as $(K+1)T$, where $T$ represents the data-driven period length and $K$ is a user-adjustable parameter. Thus, similar to the experiments in Section 6.5, we perform an evaluation to explore the impact of varying window sizes. It is conducted on the synthetic dataset ($T = 200$) described in Section 6.3, with results illustrated in Figure 11.

As illustrated in Figure 11(a), the update latency exhibits a linear increase with the increment of $K$, which aligns with our theoretical analysis provided in Proposition 4.1 Observations from Figure 11(b) indicate that the seasonality MAE remains relatively stable across the evaluated range. Meanwhile, the trend MAE decreases initially and then stabilizes for values of $K \ge 2$, a phenomenon largely attributed to inadequate smoothing at smaller window sizes. In light of the analysis presented, and to effectively balance time efficiency with decomposition accuracy, $K = 2$ is selected for all experiments in this proposal.

### B.2 Visual Decomposition Results

To demonstrate the generalizability of BacktrackSTL, we conduct experiments on an extended range of datasets. The following datasets are utilized for this purpose:

- TEMPERATURE [9, 19]: The daily minimum temperature in Melbourne, Australia from 1981 to 1990. The series length is 3650 with $T = 365$.
- SUNSPOT [13]: The monthly count of observed sunspots from 1749 to 1983. The series length is 2820 with $T = 120$.
- BIKE [18]: The daily bike sharing rental totals in Capital bike-share system from 2011 to 2012. The series length is 731 with $T = 7$.
- BIRTH [10]: The daily number of female births in California in 1959. The series length is 365 with $T = 7$.
- CPU1: The CPU utilization of a certain virtual machine in Alibaba Cloud over a week. The sampling interval is one minute. The series length is 10080 with $T = 1440$.
- CPU2: The CPU utilization of another virtual machine in Alibaba Cloud. The series length is also 10080 with $T = 1440$.

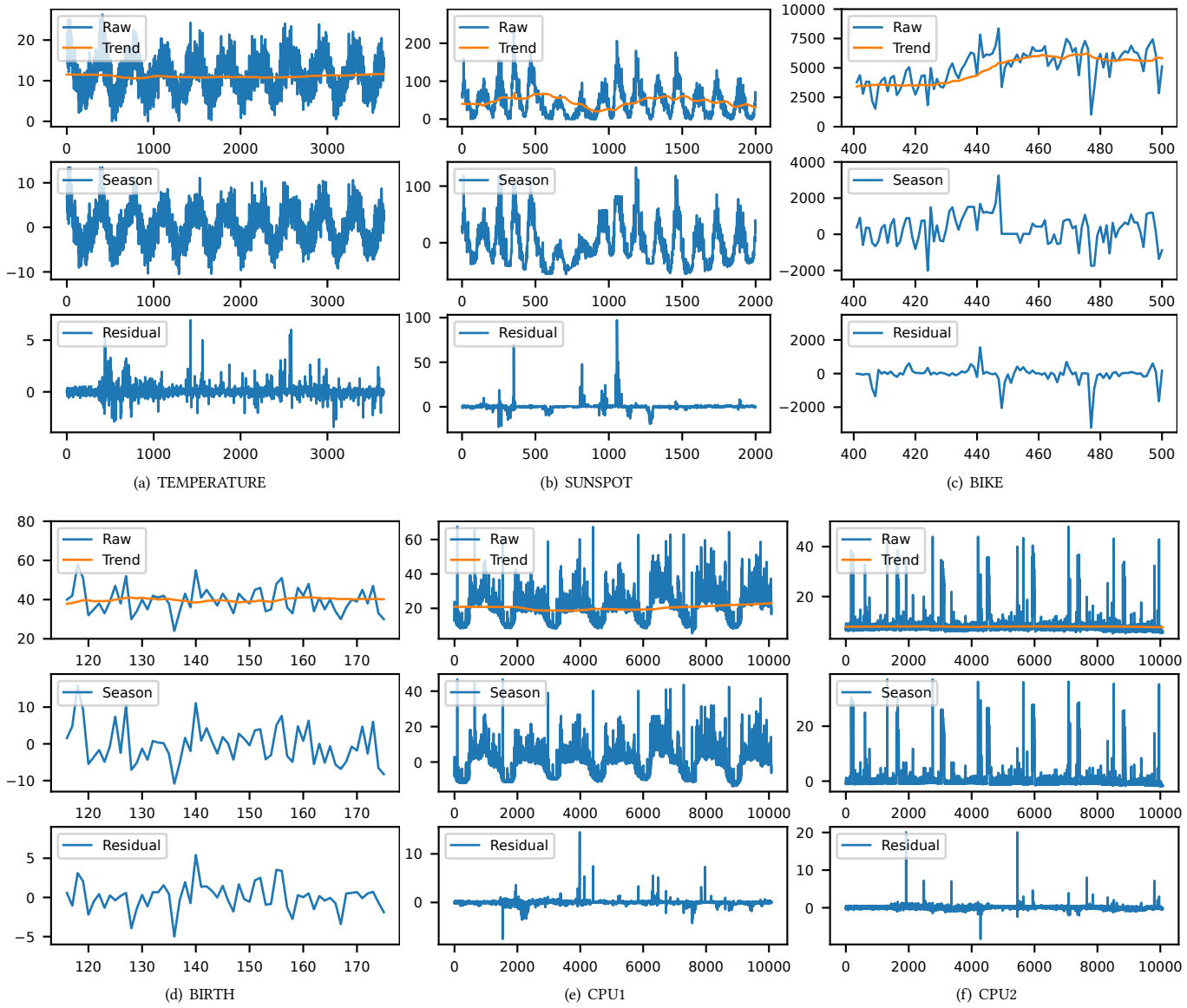
Figure 12 shows the visual decomposition results of BacktrackSTL on above six datasets. Since there are too many periods in BIKE and BIRTH datasets, we only display part of the series for clarity.

As demonstrated in (a), it is not surprising that TEMPERATURE dataset, characterized by its distinct periodic patterns, is effectively decomposed by BacktrackSTL. In the case of SUNSPOT dataset presented in (b), despite the noticeable variation in the amplitude of seasonal components, BacktrackSTL maintains its robustness, attributed to the adaptability granted by the $\delta$ parameter.

For BIKE dataset in (c) and BIRTH dataset in (d), although their periodicities are not as strong as those in the other datasets, the decomposition still aligns well with the data characteristics. Especially, BacktrackSTL extracts the rising trend in (c) successfully.

For the two CPU datasets with large period length and long series in (e) and (f), BacktrackSTL also exhibits proficient decomposition capability. It accurately extracts the smooth trend and the seasonal components with shifts, leaving a minimal number of outliers in the residual components.

In summary, BacktrackSTL exhibits great performance across datasets from various domains, demonstrating its generalizability. The effectiveness of BacktrackSTL is unaffected by the period length or series length of the dataset. Moreover, given that the complexity of BacktrackSTL is $O(1)$, which is independent of the period length $T$, it is an appropriate solution for periodic time series decomposition.




(a) TEMPERATURE (b) SUNSPOT (c) BIKE

(d) BIRTH (e) CPU1 (f) CPU2

**Figure 12: Decomposition results on various datasets**