A PROOFS

A.1 Proof of Proposition 4.3

Proof. For a point p in bucket $B[u]$, its r-neighbors must locate in $[p.v-r, p.v+r]$. Referring to Section 3.1, the points in bucket $B[u]$ must fall into interval $[u\gamma, (u+1)\gamma)$, i.e., $u\gamma \le p.v < (u+1)\gamma$. Since $(\lambda - \frac{1}{2})\gamma < r \le \lambda\gamma$, the following inequality can be derived.

$$(u-\lambda)\gamma \le p.v - r < (u-\lambda+\frac{3}{2})\gamma$$

$$(u+\lambda-\frac{1}{2})\gamma < p.v + r < (u+\lambda+1)\gamma$$

That is, $N(p,r) \subseteq [(u-\lambda)\gamma, (u+\lambda+1)\gamma)$, which leads to $|N(p,r)| \le \sum_{i=u-\lambda}^{u+\lambda} |B[i]|$. Since $|B[i]| \le |\hat{B}[i]| = \sum_{h=1}^{\rho} |B^{(h)}[i]|$ referring to Proposition 4.1, there has

$$|N(p,r)| \le \sum_{i=u-\lambda}^{u+\lambda} \sum_{h=1}^{\rho} |B^{(h)}[i]|.$$

□

A.2 Proof of Proposition 4.4

Proof. First, we have $v_{\min} + (u-1)\gamma \le p.v < v_{\min} + u\gamma$. Since $(\lambda-1)\gamma < r \le (\lambda - \frac{1}{2})\gamma$, it follows

$$v_{\min} + (u-\lambda-\frac{1}{2})\gamma \le p.v - r < v_{\min} + (u-\lambda+1)\gamma.$$

Intuitively, with the constraint $(2\lambda-2)\gamma < 2r \le (2\lambda-1)\gamma$, $N(p,r)$ at most overlaps with 2λ consecutive buckets.

Thus, for the case with $v_{\min} + (u-\lambda-\frac{1}{2})\gamma \le p.v - r < v_{\min} + (u-\lambda)\gamma$, the r-neighbor of p overlaps with 2λ buckets $B[u-\lambda], B[u-\lambda+1] \ldots, B[u+\lambda-1]$, leading to $|N(p,r)| \le \sum_{i=u-\lambda}^{u+\lambda-1} |B[i]|$.

Otherwise, for the case with $v_{\min} + (u-\lambda)\gamma \le p.v - r < v_{\min} + (u-\lambda+1)\gamma$, the r-neighbor of p overlaps with 2λ buckets $B[u-\lambda+1], B[u-\lambda+2] \ldots, B[u+\lambda]$, leading to $|N(p,r)| \le \sum_{i=u-\lambda+1}^{u+\lambda} |B[i]|$.

Combining the above cases, the upper bound can be obtained as the maximum, i.e.,

$$|N(p,r)| \le \max\left(\sum_{i=u-\lambda}^{u+\lambda-1} |B[i]|, \sum_{i=u-\lambda+1}^{u+\lambda} |B[i]| \right).$$

For the case with multiple files, since $|B[i]| \le |\hat{B}[i]| = \sum_{h=1}^{\rho} |B^{(h)}[i]|$ referring to Proposition 4.1, there has

$$|N(p,r)| \le \max\left(\sum_{i=u-\lambda}^{u+\lambda-1} \sum_{h=1}^{\rho} |B^{(h)}[i]|, \sum_{i=u-\lambda+1}^{u+\lambda} \sum_{h=1}^{\rho} |B^{(h)}[i]| \right).$$

□

A.3 Proof of Proposition 4.6

Proof. For a point p in bucket $B[u]$, its r-neighbors must locate in $[p.v-r, p.v+r]$. Referring to Definition 3.1, the points in bucket $B[u]$ must fall into $[v_{\min} + (u-1)\gamma, v_{\min} + u\gamma)$, and we thus have $v_{\min} + (u-1)\gamma \le p.v < v_{\min} + u\gamma$. Since $\ell\gamma \le r < (\ell+1)\gamma$, we can derive

$$v_{\min} + (u-\ell-2)\gamma < p.v - r < v_{\min} + (u-\ell)\gamma,$$
$$v_{\min} + (u+\ell-1)\gamma \le p.v + r < v_{\min} + (u+\ell+1)\gamma.$$

That is,

$$[p.v-r, p.v+r] \supseteq [v_{\min} + (u-\ell)\gamma, v_{\min} + (u+\ell-1)\gamma),$$

which leads to $|N(p,r)| \ge \sum_{i=u-\ell+1}^{u+\ell-1} |B[i]|$. Since we have $|B[u]| \ge |B^{(\rho)}[u]|$ in Proposition 4.1, we can prove that

$$|N(p,r)| \ge \sum_{i=u-\ell+1}^{u+\ell-1} |B^{(\rho)}[i]|.$$

□

B SINGLE FILE QUERY ALGORITHM

Now, we present the pseudo-code of querying outliers in a file via sliding window in Algorithm 2. Following the convention of query processing in data streams [19], we also consider the current sliding window W, with a number of points expired and some others new compared to the previous window. Thereby, Lines 7 and 9 update the window statistics for each bucket $B[u]$, referring to the aggregation property in Proposition 3.3 in Section 3.2.

The pruning of points in bucket $B[u]$ is then applied, i.e., cases 1 and 2 in Section 4.3. Specifically, we prune the entire bucket $B[u]$ as inliers in Line 10, according to the lower bound in Proposition 3.7 in Section 3.3.2. Likewise, we directly output the points in bucket $B[u]$ as outliers in Lines 12 and 14, referring to Propositions 3.5 and 3.6, in Section 3.3.1.

Otherwise, we need to load at most 4 additional buckets to determine the r-neighbors of points p in bucket $B[u]$, i.e., case 3 in Section 4.3. Lines 19 and 22 aggregate the r-neighbor count according to Proposition 4.8, to determine the outlier.

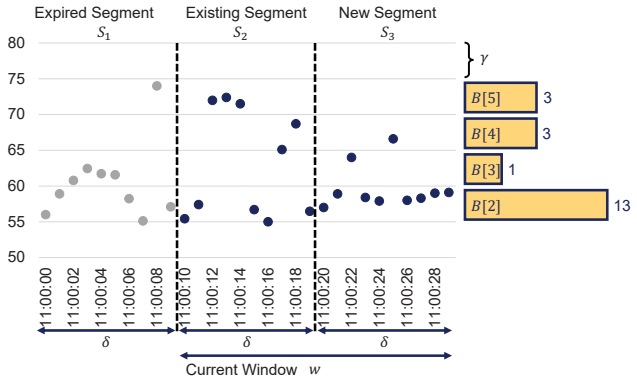

Figure 16: Query in sliding window with bucket statistics

Example B.1 (Example 4.9 continued). Figure 16 shows the next sliding window of Figure 4, where a segment S_1 is expired and a new segment S_3 comes. The corresponding bucket statistics are updated for the new window $W_{11:00:10}$ starting from time 11:00:10. For instance, we have $|B[2]| = |B[2]| - |B_1[2]| + |B_3[2]| = 10 - 5 + 8 = 13$, where $|B_1[2]|$ and $|B_3[2]|$ are the 2nd bucket sizes in segments S_1 and S_3, respectively. Once all the bucket statistics are updated, similar to the previous window in Example 4.9, it checks whether the pruning is applicable. If not, i.e., case 3 in Section 4.3, the algorithm loads the additional buckets for evaluating r-neighbors.

Complexity Analysis. We have β buckets as introduced in Definition 3.1. Consider all the ω segments in a window as in Definition 3.3. There are at most ω segments expired and ω segments new in the sliding window. The update of bucket statistics in Lines 7 and 9 thus takes $O(\beta\omega)$ time. In the worst case, all the n points in the window may be output as outliers, referring to the upper bounds in Lines 12 and 14, in $O(n)$ time. Otherwise, we need to load point p and check its r-neighbors. Luckily, we only need to load a constant number (up to 4) of additional buckets in Line 17. Referring to the average number of points in a bucket $\frac{n}{\beta}$, it takes $O(\frac{n^2}{\beta})$ time. To sum up, Algorithm 2 runs in $O(\beta\omega + \frac{n^2}{\beta})$ time, and $O(\beta\zeta)$ extra space, where β is the number of buckets, ω is the number of segments in a window, n is the number of points in a window and ζ is the number of segments in a file.

Algorithm 2 Outlier Detection Algorithm on Single File

Input: A window W_t at time t with size w and slide s, neighbor distance threshold r and count threshold k
Output: outlier set O_t of current window W_t

1: $\lambda := \lceil r/\gamma \rceil$
2: $\ell := \lfloor r/\gamma \rfloor$
3: initialize outlier set $O_t := \emptyset$
4: initialize buckets $\mathcal{B}$ if algorithm runs for the first window
5: **for** each bucket $B[u]$, $u := 1$ to β **do**
6: **for** each expired segment S_g **do**
7: $|B[u]| := |B[u]| - |B_g[u]|$
8: **for** each new segment S_g **do**
9: $|B[u]| := |B[u]| + |B_g[u]|$
10: **if** $\sum_{i=u-\ell+1}^{u+\ell-1} |B[i]| \geq k$ **then**
11: continue
12: **else if** $\lceil r/\gamma \rceil - r/\gamma < \frac{1}{2}$ **and** $\sum_{i=u-\lambda}^{u+\lambda} |B[i]| < k$ **then**
13: $O_t := O_t \cup B[u]$
14: **else if** $\lceil r/\gamma \rceil - r/\gamma \geq \frac{1}{2}$ **and** $\max\left(\sum_{i=u-\lambda}^{u+\lambda-1} |B[i]|, \sum_{i=u-\lambda+1}^{u+\lambda} |B[i]|\right) < k$ **then**
15: $O_t := O_t \cup B[u]$
16: **else**
17: load additional buckets $B[u \pm \lambda]$, $B[u \pm \ell]$
18: **for** each point p in $B[u]$ **do**
19: $cnt := \sum_{i=u-\ell+1}^{u+\ell-1} |B[i]|$
20: **for** each point p' in $B[u \pm \lambda] \cup B[u \pm \ell]$ **do**
21: **if** $dist(p, p') \leq r$ **then**
22: $cnt := cnt + 1$
23: **if** $cnt < k$ **then**
24: $O_t := O_t \cup \{p\}$
25: **return** O_t

C EXTRA EVALUATION

C.1 Scalability in Data Sizes

Figure 17 reports the query time costs for different numbers of data points. It is not surprising that the time costs of all methods increase with the data sizes. Our proposed LSMOD consistently shows 1 order of magnitude improvement compared to the baselines. The time cost of LSMOD grows much slower than the baselines,

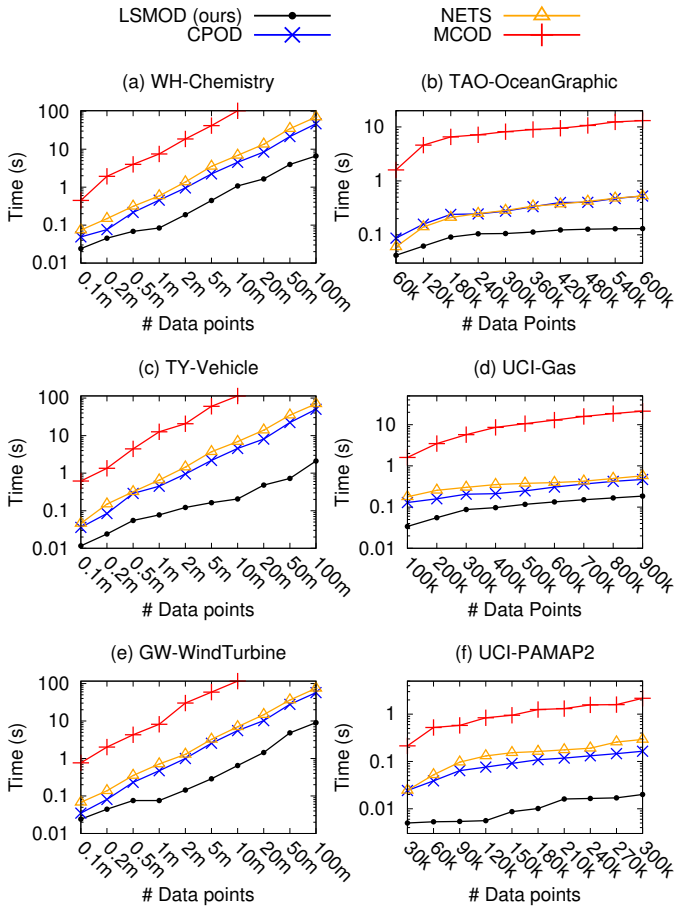

Figure 17: Query time costs under different data sizes

especially on large scale datasets (a) WH-Chemistry, (c) TY-Vehicle, and (e) GW-WindTurbine with 100 million data points. NETS and CPOD have similar time costs in Figure 17, which is consistent with the previous evaluation [20]. Unfortunately, MCOD cannot handle datasets with more than 20 million points in a limit of 200 seconds. The results verify the effectiveness of our pruning by bucket statistics, rather than loading and merging all the data as baselines.

C.2 Supporting Various Query Parameters

As introduced in Section 1, outliers are often queried with different parameter settings for various user interests and application demands. Same as the existing methods, our LSMOD also supports queries with different (1) neighbor distance threshold r, (2) neighbor count threshold k, (3) window size w, and (4) slide size s. Note that it is not the focus of this study to determine proper parameters for various applications, such as neighbor threshold r in miles for destination prediction application or in feed for trajectory analysis as discussed in Section 1. Instead, we concentrate on efficiently processing the outlier queries posed by different domain experts with various parameter settings. Thereby, we compare our proposal

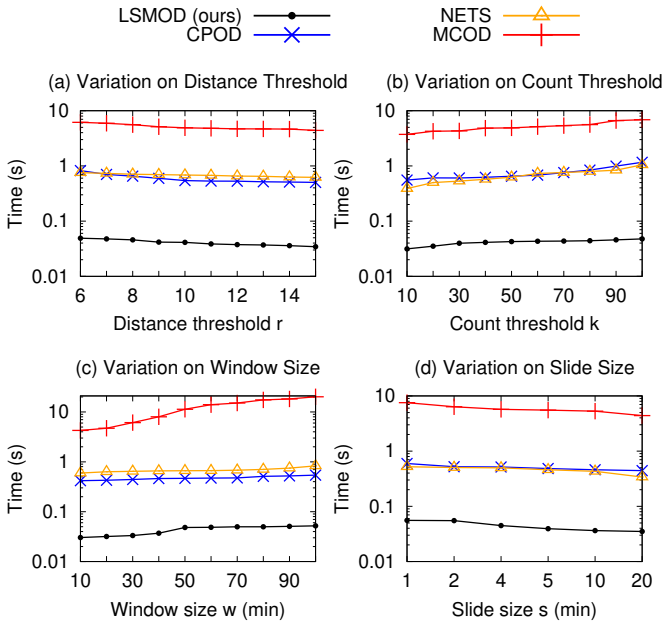

Figure 18: Varying query parameters on WH-Chemistry

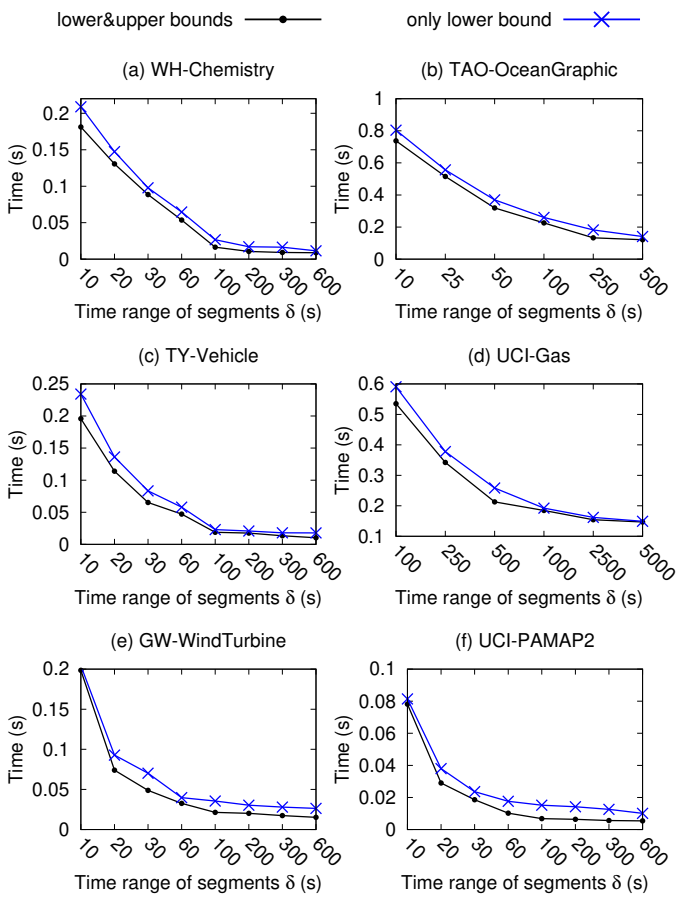

Figure 19: Varying the range of segments on time δ

with other baselines under different query parameters, and present the results on the dataset WH-Chemistry in Figure 18.

When neighbor distance threshold r increases in Figure 18(a), each data point has more neighbors in a window, and more inliers can be directly pruned by the index structures of NETS, CPOD and MCOD. Therefore, the time costs of all the baselines decrease. Similarly, for LSMOD, it may aggregate more buckets with the increase of r, and the lower bounds for the neighbor count of each point become larger. Thus, more inliers can be pruned by comparing the lower bounds with the fixed k, also leading to the decrease of time cost.

When varying k in Figure 18(b), the baselines need to check more points to determine the point status no matter by which index structure, resulting in a slight increasing tendency. For our LSMOD, when k is small, inliers are easy to be determined by Proposition 3.7. However, fewer inliers can be pruned with the increase of k, and the time cost thus increases as well.

In Figure 18(c), with window size w increasing, the time cost of MCOD increases rapidly, since the points and micro-clusters in a window become more time-costly to maintain. For others, their time costs slightly increase due to the more costly initialization phases, while our LSMOD remains 1 order of magnitude faster than other baselines.

When varying slide size s in Figure 18(d), NETS, CPOD and our LSMOD have lower time costs under large slide sizes. This is because a larger slide size s means fewer windows to check, for a time series with fixed length. However, MCOD takes more time to maintain its micro-clusters, with more points expired and arrived for each slide.

C.3 Varying the range of segments on time δ

Figure 19 varies the time range δ for splitting time series into segments in Definition 2.5. Under the same data sizes, a larger time range δ leads to fewer segments, and thus the processing time cost decreases. Recall that to support query processing in sliding window, the window size w and slide s are usually the integral multiples of δ, as discussed in Section 2.2. A larger δ thus means less supported queries.