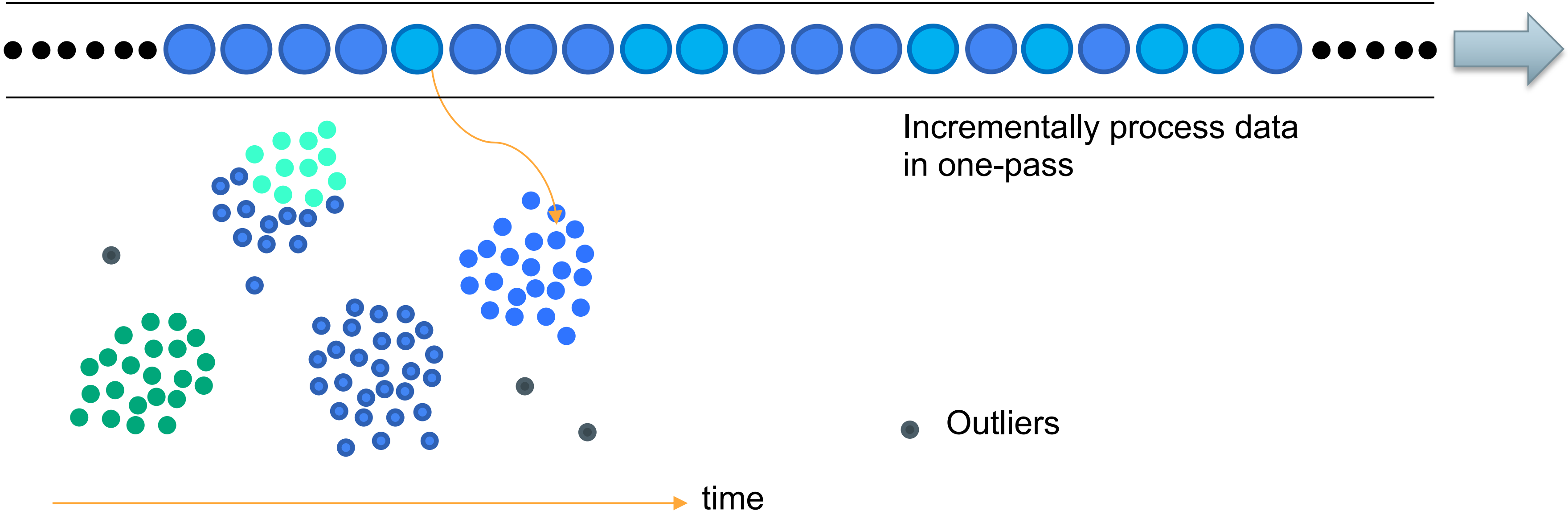
Data Stream Clustering: An In-depth Empirical Study

Shuhao Zhang (SUTD)

Collaborating with HUST and Sichuan U

Background: Data Stream Clustering (DSC)

- **Definition:** Partitioning streaming data into clusters in real time.

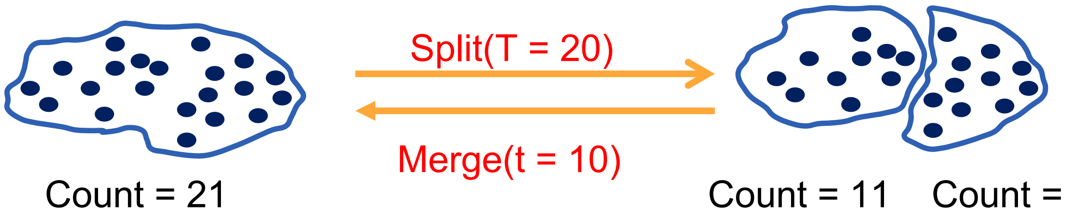


Data distribution evolves with time

Background: Data Stream Clustering (DSC)

- **Definition:** Partitioning streaming data into clusters in real time.
- **Challenges:**
  - Memory limitation ⟷ Unbounded data streams
  - Fast response time ⟷ Usually expecting fast responses
  - Handle evolving activities
    - Cluster Evolution
    - Outlier Evolution

    ⟷ Shifting nature of data distributions and the emergence of new outliers over time



Background: Data Stream Clustering (DSC)

- **Definition:** Partitioning streaming data into clusters in real time.
- **Challenges:**
  - Memory Limitation ⟷ Unbounded data streams
  - Fast Response Time ⟷ Usually expecting fast responses
  - Handle evolving activities
    - Cluster Evolution
    - Outlier Evolution

    ⟷ Shifting nature of data distributions and the emergence of new outliers over time
- **Algorithms:** CluStream [VLDB'03], D-Stream [KDD'07], DBStream [TKDE'16], EDMStream [VLDB'17], SL-Kmeans [NIPS'20] …

Background: Literature Gaps and Contributions

- **Coarse-grained comparison:**
  - Ignore analysing the impact of individual design aspect of algorithms.
- **Problematic benchmark settings:**
  - Not implement algorithms in a unified framework (e.g., varying programming languages, compilers)
  - Lack evaluating processing efficiency.

(×) It is not interesting to compare Huawei Mate60 against Iphone14 entirely.

(√) It is more insightful to go into the details: compare their CPU, GPU, memory, etc.

(×) It is less meaningful to compare two algorithms that are implemented in python and C++, respectively.

(√) It is far more fair to implement all algorithms in one language, e.g., C++, before the comparison.

(×) Existing studies only evaluate accuracy.

(√) Both accuracy and efficiency shall be evaluated carefully.

Background: Our Contributions

Our Contributions:

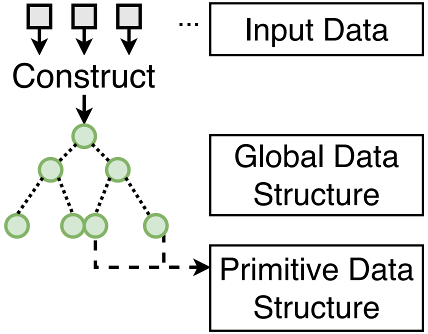
- Implement algorithms and designs in an open-sourced platform named ***Sesame*** (supporting 22 different DSC algorithms, ~13,000 lines of code in C++)
- Evaluate both accuracy and efficiency impact on every single design aspect (our study is like using a “microscope” to observe and analysis DSC algorithms)
- Propose a new algorithm ***Benne*** through combining flexible design choices achieving either SoTA accuracy and efficiency (this is an unexpected gift).

Published (SIGMOD’23)

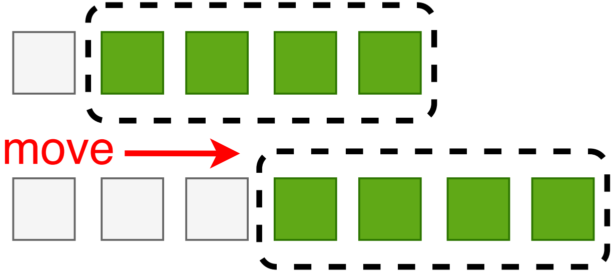
So, how does a DSC algorithm look like actually?

?

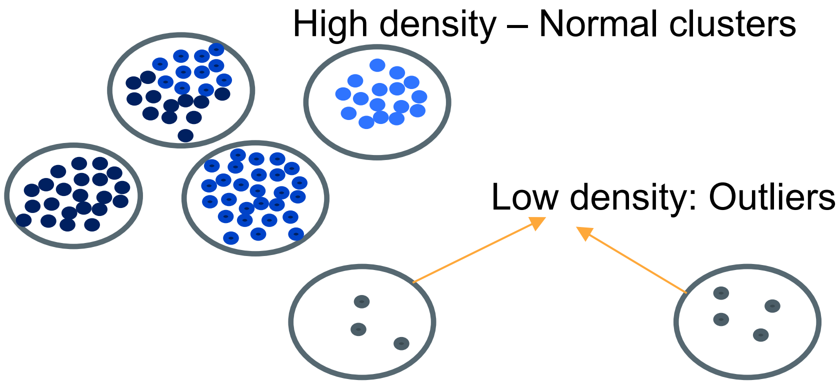
Design Aspects:



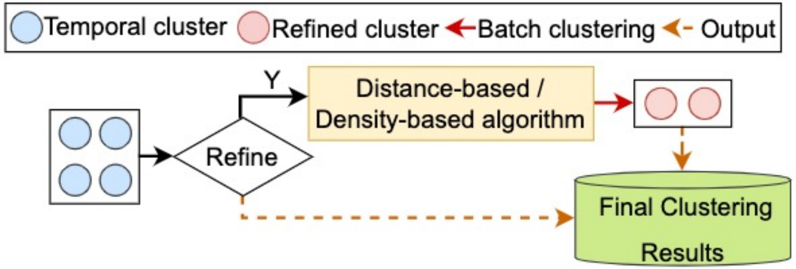
(a) Summarizing Data Structure



(b) Window Model



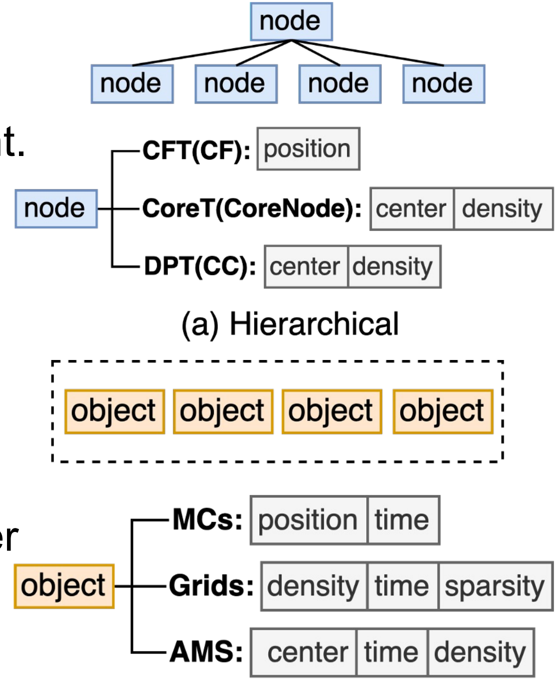
(c) Outlier Detection



(d) Refinement Strategy

Design Aspects: Summarizing Data Structure

- **Hierarchical:** Organize temporal clusters into a tree.
  - **Clustering Feature Tree (*CFT*):** flexibly make adjustment.
  - **Coreset Tree (*CoreT*):** lazily rebuild the whole structure.
  - **Dependency Tree (*DPT*):** cluster based on the evolving cluster density.
- **Partitional:** Organize temporal clusters into a list.
  - **Micro Clusters (*MCs*):** similar structure as *CFT* but under different catalog.
  - **Grids (*Grids*):** free from frequent distance calculation.
  - **Augmented Meyerson Sketch (*AMS*):** Frequently reconstruct temporal clusters to keep its total number fixed.



(b) Partitional

Figure: Two Catalogs of Summarizing Data Structures.

Design Aspects: Window Model

- **Landmark (*LandWM*):** Cluster data between two landmarks into a window.
- **Sliding (*SlidingWM*):** Cluster data whose timestamp falls within current window range.
- **Damped (*DampedWM*):** Associate data with weights decaying over time.

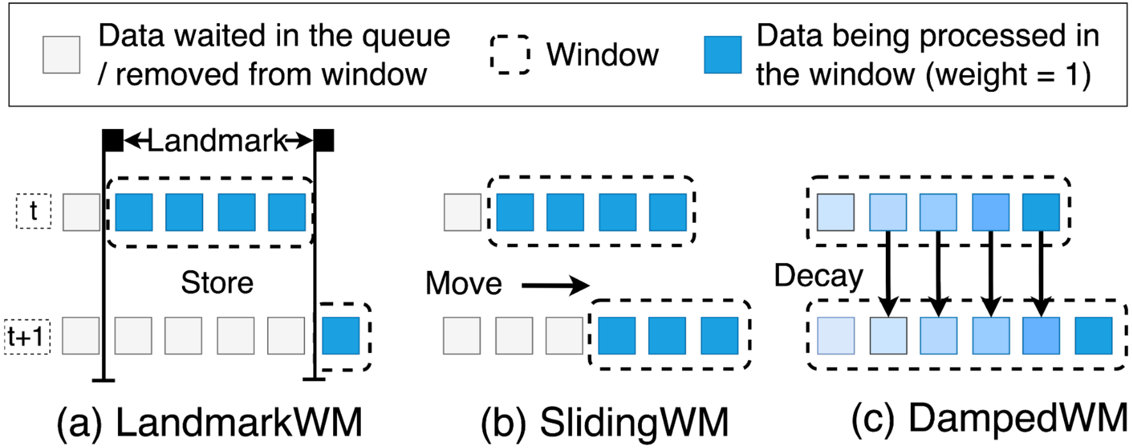


Figure: Three Types of Window Models.

Design Aspects: Outlier Detection Mechanism

- **No Outlier Detection (*NoOutlierD*)**
- **Outlier Detection (*OutlierD*):** periodically discard sparse clusters
- **Outlier Detection with buffer (*OutlierD-B*):** store the sparse clusters into a buffer rather than discarding.
- **Outlier Detection with Timer (*Outlier-T*):** additionally check the activity of the discarded clusters
- **Outlier Detection with Buffer and Timer (*OutlierD-BT*)**

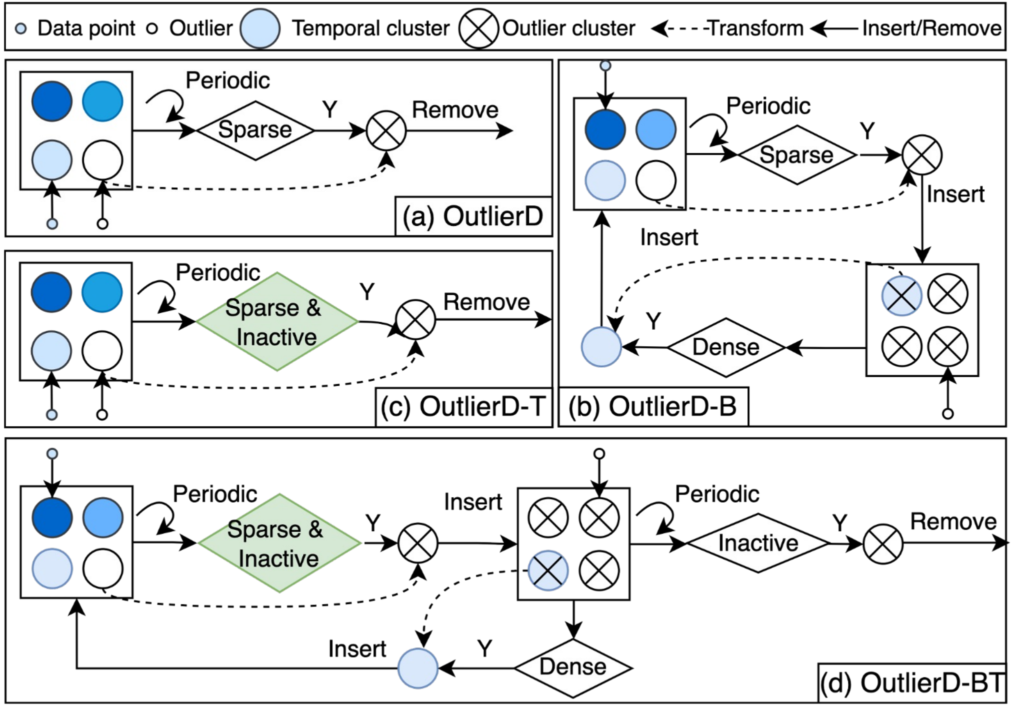


Figure: Four Types of Outlier Detection Mechanism

Design Aspects: Refinement Strategy

- **With refinement (Refine):** Apply batch clustering algorithms such as KMeans or DBSCAN to further refine the online results before output.
- **Without refinement (NoRefine):** directly output the online temporal clusters as the final clustering results.

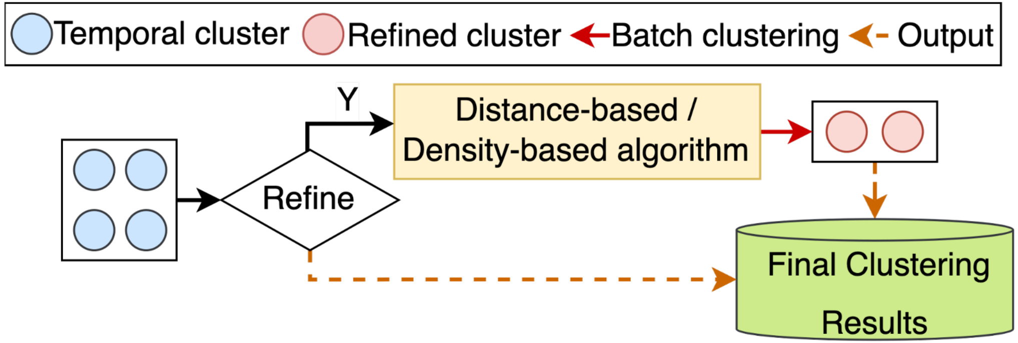


Figure: General workflow of refinement strategy

How do different design options really matter?

Methodology

- **Benchmark Testbed: *Sesame***

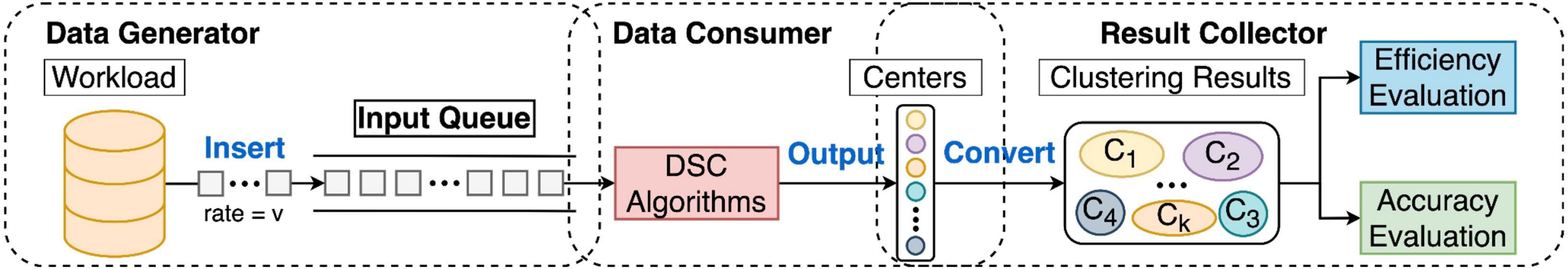


Figure: Sesame Workflow

Methodology

- **Algorithm Selection:**
  - Cover a wide range of design decisions of all four design aspects.
  - Either representative or recently covering a long history in this field.

Algorithm	Year	Summarizing Data Structure		Window Model	Outlier Detection	Offline Refinement
		Name	Catalog			
BIRCH	1996	*CFT*	*Hierarchical*	*LandmarkWM*	*OutlierD*	*NoRefine*
CluStream	2003	*MCs*	*Partitional*	*LandmarkWM*	*OutlierD-T*	*Refine*
DenStream	2006	*MCs*	*Partitional*	*DampedWM*	*OutlierD-BT*	*Refine*
DStream	2007	*Grids*	*Partitional*	*DampedWM*	*OutlierD-T*	*Refine*
StreamKM++	2012	*CoreT*	*Hierarchical*	*LandmarkWM*	*NoOutlierD*	*Refine*
DBStream	2016	*MCs*	*Partitional*	*DampedWM*	*OutlierD-T*	*Refine*
EDMStream	2017	*DPT*	*Hierarchical*	*DampedWM*	*OutlierD-BT*	*NoRefine*
SL-KMeans	2020	*AMS*	*Partitional*	*SlidingWM*	*NoOutlierD*	*NoRefine*

Table: Selected Algorithm Summary

Methodology

- **Workload Selection:**

Table: Workload Summary

Workload	Length	Dimension	Cluster Number	Outliers	Evolving Frequency
FCT	581012	54	7	False	Low
KDD99	4898431	41	23	True	Low
Insects	905145	33	24	False	Low
Sensor	2219803	5	55	False	High
EDS	245270	2	363	False	Varying
ODS	100000	2	90	Varying	High

- **Evaluation Metrics:**
    - **Accuracy:** We use purity to measure the general clustering quality and also use CMM to test the design aspects' ability to handle cluster evolution.
    - **Efficiency:** We use throughput for efficiency comparison.

Experimental Analysis: Findings and Obseravations

- **Key Finding 1: For each design aspect, none of the design choices can always guarantee good performance under varying workload characteristics and/or optimization targets.**

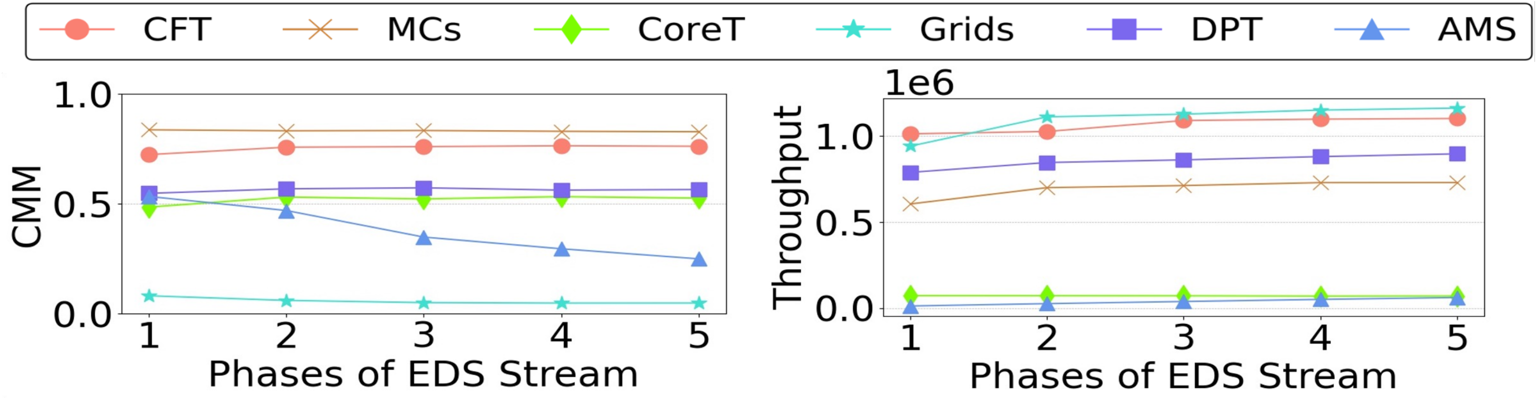


Figure: Comparison of the Ability for Summarizing Data Structures to Handle Cluster Evolution.

Observation 1: *MCs* and *CFT* guarantee high accuracy while *CFT* and *Grids* guarantee high efficiency for handling cluster evolution than other types of data structure.

Experimental Analysis: Findings and Obseravations

- **Key Finding 1: For each design aspect, none of the design choices can always guarantee good performance under varying workload characteristics and/or optimization targets.**

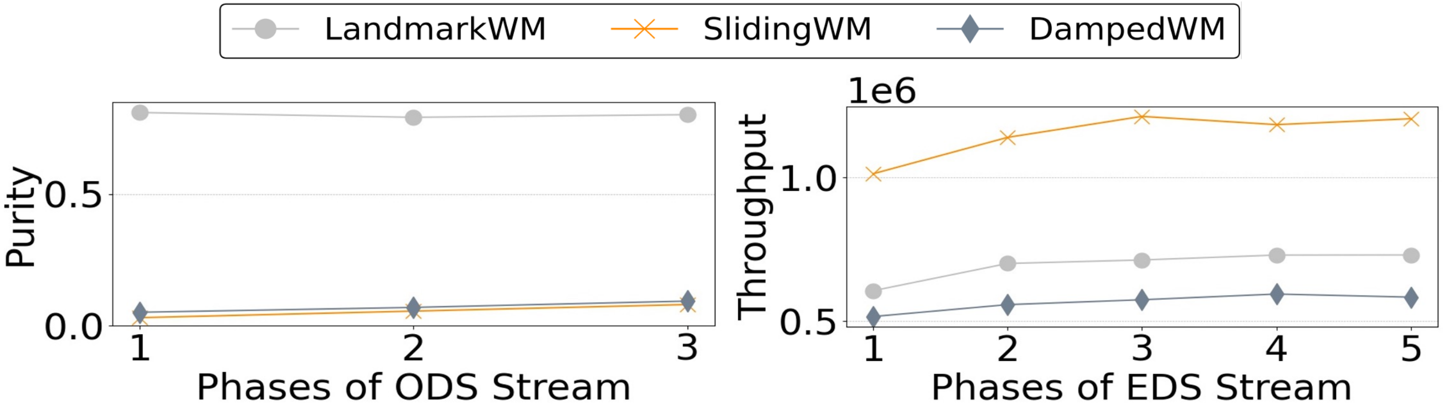


Figure: Comparison of the Ability for Window Models to Handle Outlier / Cluster Evolution.

Observation 2: The efficiency of *LandmarkWM* and *DampedWM* becomes worse with the increase of cluster evolution frequency.

Experimental Analysis: Findings and Obseravations

- **Key Finding 1: For each design aspect, none of the design choices can always guarantee good performance under varying workload characteristics and/or optimization targets.**

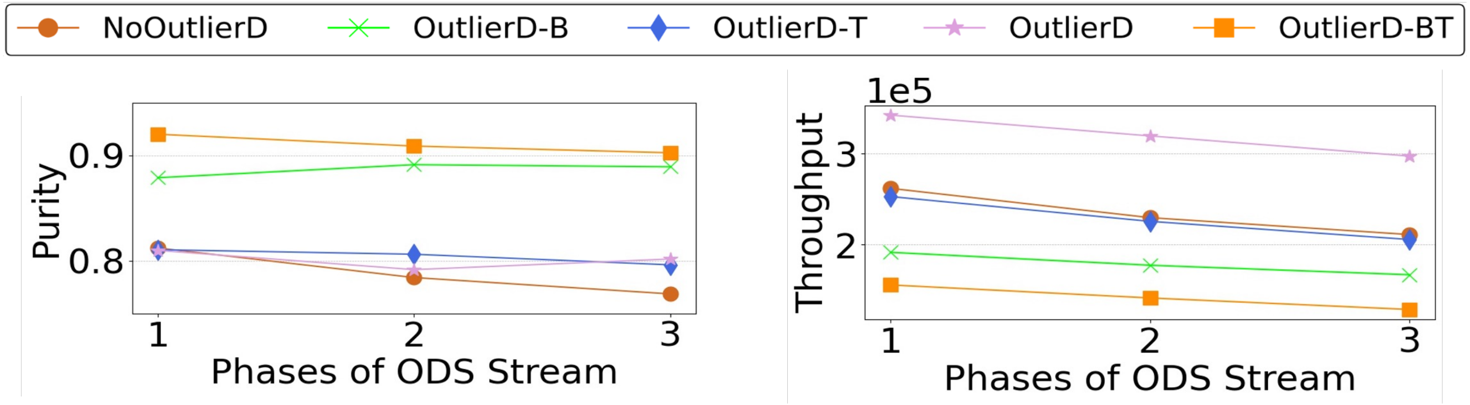
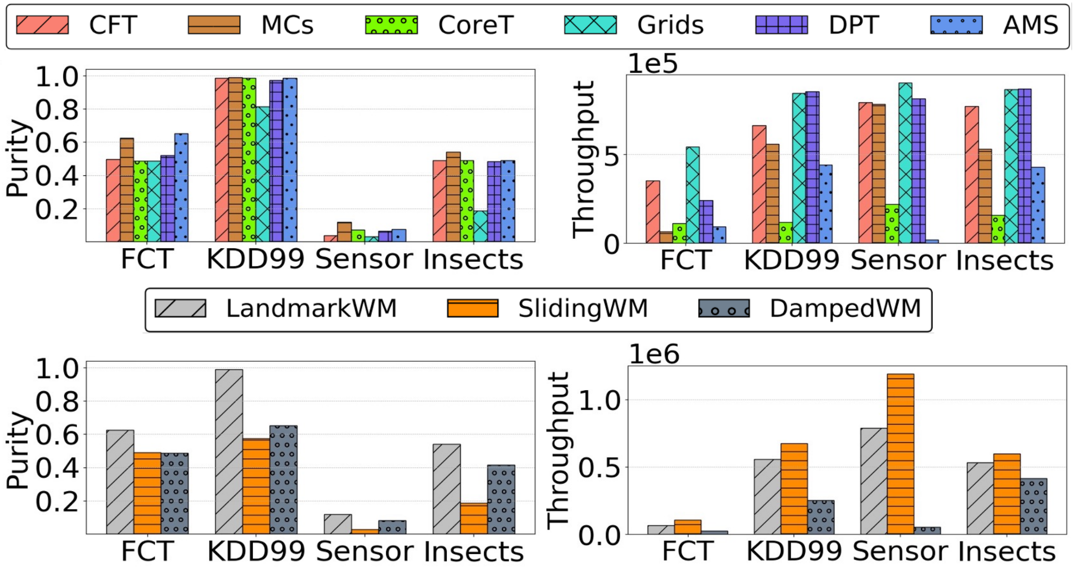


Figure: Comparison of the Ability for Outlier Detection Mechanisms to Handle Outlier Evolution.

Observation 3: Utilizing a *timer* in outlier detection can greatly improve the clustering accuracy and even increase the efficiency under outlier evolution.

Experimental Analysis: Findings and Obseravations

- **Key Finding 2: Each combined selection of design choices from four design aspects has its own strength and limitation and none can achieve the highest accuracy and efficiency at the same time.**



Observation 3:

{*Grids* summarizing data structure}
+ {*SlidingWM* window mode}
lead to high clustering efficiency but low accuracy.

Figure: General Comparison of Summarizing Data Structures / Window Models

Experimental Analysis: Findings and Obseravations

- **Key Finding 2: Each combined selection of design choices from four design aspects has its own strength and limitation and none can achieve the highest accuracy and efficiency at the same time.**

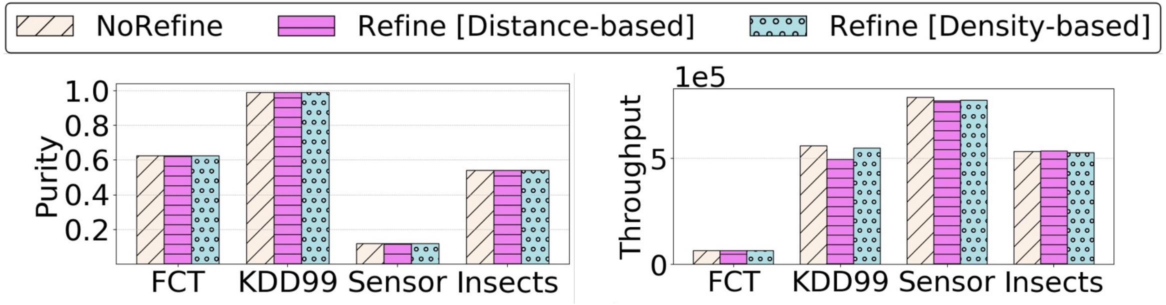


Figure: General Comparison of Offline Refinement Strategy.

Observation 4: Applying an *offline refinement strategy* has little impact on both clustering accuracy and efficiency.

Observation 5: Composing suitable design choices from each design aspect, we obtain a novel DSC algorithm (i.e., *Benne*) that can be reconfigured to achieve either the highest accuracy or highest efficiency, but not at the same time. (Show later)

Experimental Analysis: Findings and Obseravations

- **Key Finding 3:** **Algorithm configuration and correlations among design aspects bring further complex influence on the clustering behaviour.**

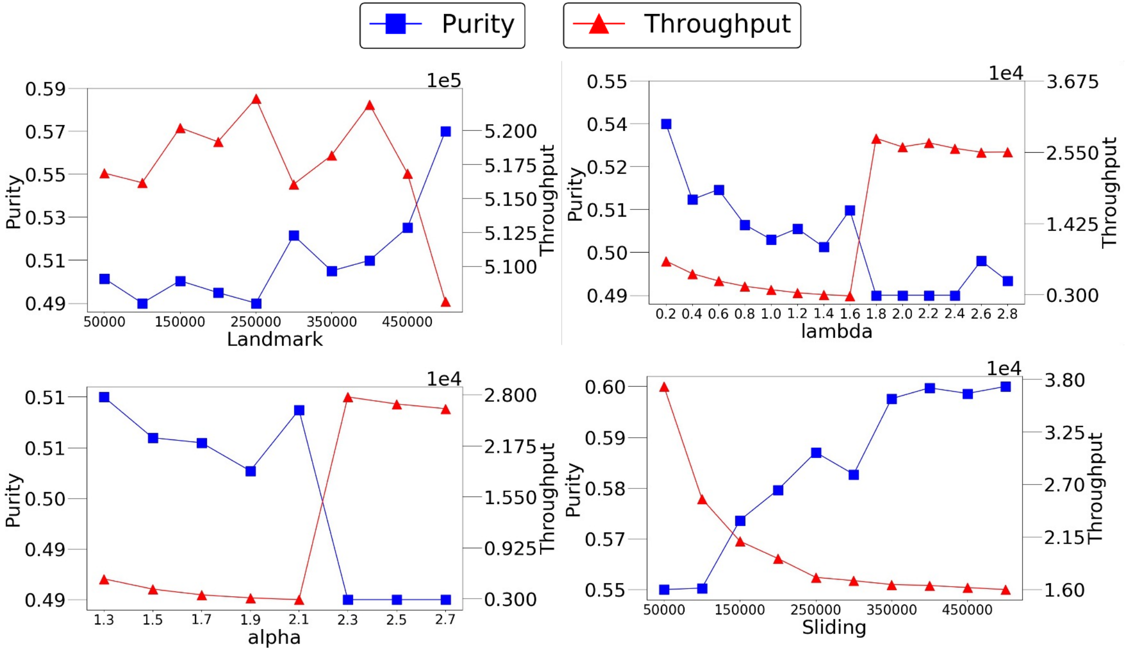


Figure: Configuration Analysis of Window Models.

Observation 6: Different algorithm configurations bring *non-trivial trade-offs* in terms of accuracy and efficiency.

Observation 7: There is an unsuitable summarizing data structure *dominantly leads to poor performance* of DSC algorithms regardless of the selection of other design choices.

How does the empirical analysis useful?

Experimental Analysis: *Benne* Algorithm

Algorithm 1: Execution flow of *Benne*.

```
Data: p // Input point
Data: s // Summarizing data structure
Input: struc. // Selected type of summarizing data structure
Input: win. // Selected type of window model
Input: out. // Selected type of outlier detection mechanism
Input: ref. // Selected type of refinement strategy
// Online Phase
while !stop processing of input streams do
    Window Fun. (...);
    if out. != NoOutlierD then
        b ← Outlier Fun. (...);
        if b = false then
            Insert Fun. (...)// Insert p to s and update s
    else
        Insert Fun. (...)// Insert p to s and update s
// Offline Phase
if ref. != NoRefine then
    Refine Fun. (ref.);
```

- **Benne (Accuracy):**

MCs + LandmarkWM + OutlierD-B + NoRefine

- **Benne (Efficiency):**

CFT + LandmarkWM + OutlierD-T + NoRefine

Experimental Analysis: Overall Comparison

- **Benne (Accuracy)** achieves the best purity.
- **Benne (Efficiency)** achieves the highest throughput.

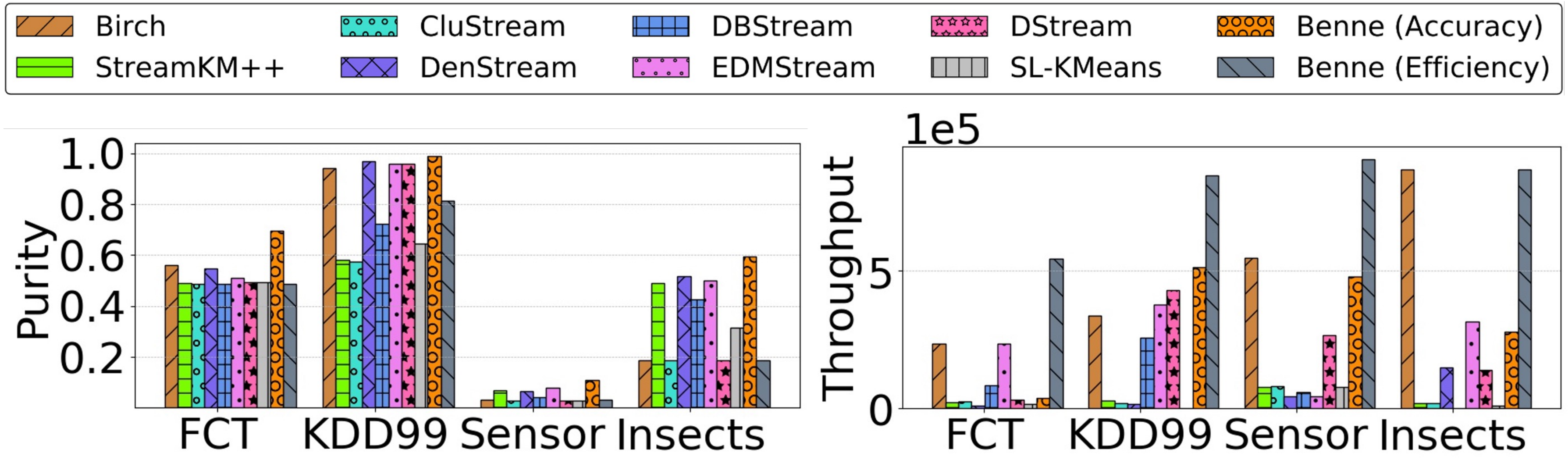


Figure: General Comparison of Existing DSC algorithms and *Benne*.

Open-Source

- Code, data and scripts are available at https://github.com/intellistream/Sesame
- It is purely written in modern C++.
- One can easily reproduce all of our experimental results by “one-click” of our scripts in your machine.
- For easier access by the ML community, we have additionally built a Python API to our framework.

```python
import benne

# Create an instance of Parameters
params = benne.Parameters()

# Get and set the algorithm
algorithm = params.algo

print("Current algorithm:", algorithm)
params.algo = benne.AlgoType.BIRCH
params.input_file = "/home/shaun/Sesame/benchmark/datasets/CoverType.txt"
print("Updated algorithm:", params.algo)

# ... Continue getting and setting other parameters
# Accessing docstring
print(benne.Parameters.algo.__doc__)

print("Input file:", params.input_file)

# Run the SESAME algorithm
benne.run()
```

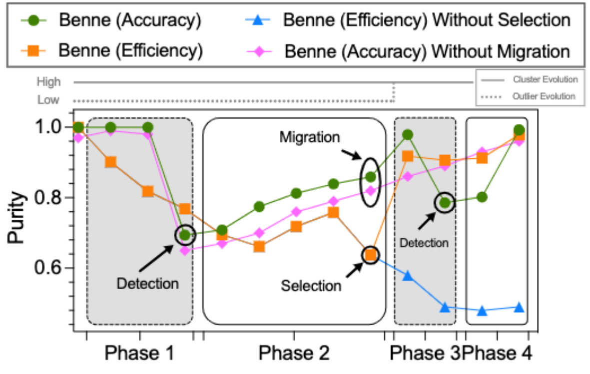
And then?

Enhancements

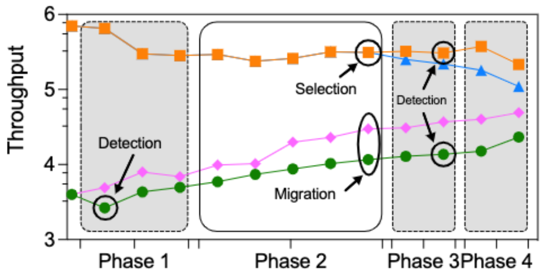
- **Key Enhancement 1:** Regular Stream Characteristics Detection
- **Key Enhancement 2:** Automatic Design Choice Selection
- **Key Enhancement 3:** Flexible Algorithm Migration

Under Review (TKDE)

Enhancements



(a) Accuracy



(b) Efficiency

1) Both Benne (Accuracy) and Benne (Efficiency) swiftly recover from workload changes.

2) Automatic design choice selection is a critical component to ensure the adaptivity.

3) Algorithm migration improves accuracy at the expense of clustering speed.

Figure: Detailed performance analysis on KDD99 workload

Enhancements

1) For Benne (Accuracy), the time allocation for both detection and migration is relatively minimal in comparison to the primary clustering task.
2) For Benne (Efficiency), the proportion of time spent on the detection appears to be larger. However, it's crucial to note that Benne (Efficiency) omits the migration procedure altogether.

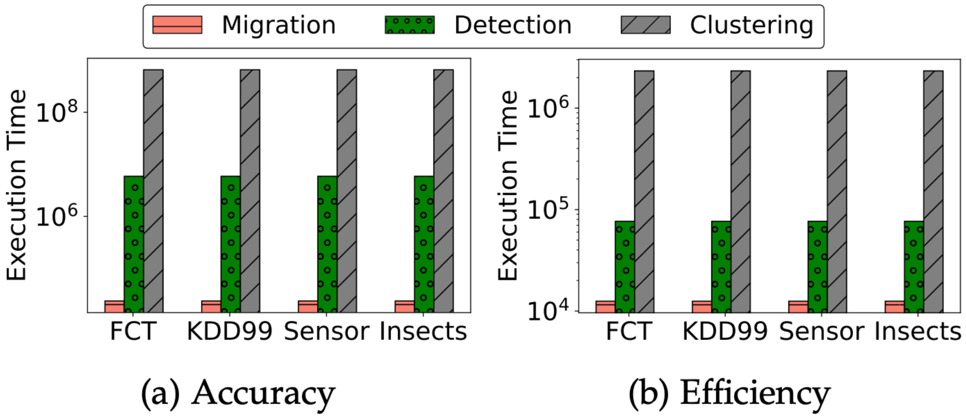


Figure: Execution Time Break Down Analysis

Future Work

?

Future work

- High performance scalable data stream clustering algorithms (e.g., GPU acceleration)
- High-dimensional data stream clustering (e.g., VectorDB, trajectory data stream analytics)
- Online continual learning (e.g., fast coreset selection)

Conclusion

!

Conclusion

- It is still challenging to balance the trade-off between accuracy and efficiency of DSC algorithms.
- Each design choice has its own pros and cons and should be dynamically adjusted and carefully combined to obtain the best DSC algorithm under different workload characteristics.
- A dynamic algorithm configuration strategy is required for the stream setting.

- Thank you for listening
- Email to ask follow-up questions: shuhao_zhang@sutd.edu.sg