SS-ZG548: ADVANCED DATA MINING

09 AR Mining On Streams

Dr. Kamlesh Tiwari
Assistant Professor, Department of CSIS,
BITS Pilani, Pilani Campus, Rajasthan-333031 INDIA

Sept 18, 2021 ONLINE (WILP @ BITS-Pilani July-Dec 2021)

http://ktiwari.in/adm

Recap: Data streams

Stream of data arrive in rapid succession, no Re-scan is, storage space is insufficient to accommodate all data points.

Without storing all the data one wish to estimate

- Set of frequent items
- Number of distinct items
- Frequent itemsets
- *etc*

[Constraints are on memory and processing power]

Frequent pattern mining over data streams

- Applications involves retail market data analysis, network monitoring, web usage mining, and stock market prediction.
- Using sliding window
- Efficiently remove the obsolete, old stream data
- Compact Pattern Stream tree (CPS-tree) [1]
- Highly compact frequency-descending tree structure at runtime
- Efficient in terms of memory and time complexity
- Pane and window
- Insertion and restructuring

[1] Tanbeer, Syed Khairuzzaman and Ahmed, Chowdhury Farhan and Jeong, Byeong-Soo and Lee, Young-Koo, "Sliding window-based frequent pattern mining over data streams", in Information sciences, 179(22) pages 3843–3865, Elsevier, 2009

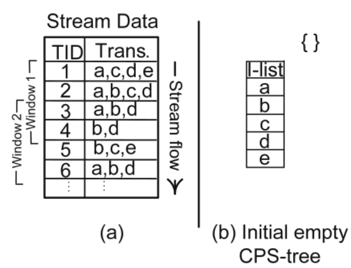
CPS-tree construction

```
Algorithm 1 (Construction of a CPS-tree for a data stream)
Input: Stream_data, Pane_size, Window_size, Initial_Sort_Order
Output: T_sort: a CPS-tree for the current window
Method:
  Begin
    w ← ϕ;
    T ← a prefix-tree with null initialization;
    Current_Sort_Order ← Initial_Sort_Order;
  //For the first Window
      While (w ≠ Window_size) do
          Call Insert_Pane(T);                                        // Insertion Phase
          Current_Sort_Order ← Frequency-descending sort order;       // Restructuring Phase
          Restructure T;
          w = w+1;
      End While
  //At each slide of Window
      Repeat
          Delete the oldest pane information from T;                  // Extracting the old pane
          Call Insert_Pane(T);                                        // Insertion Phase
          Current_Sort_Order ← Frequency-descending sort order;       // Restructuring Phase
          Restructure T;
      End
  End

Insert_Pane(Tr)
  Begin
    p ← ϕ;
    While (p ≠ Pane_size) do
        Scan transaction from the current location in Stream_data;
        Insert the scanned transaction into Tr according to Current_Sort_Order;
        p = p+1;
    End While
  End
```

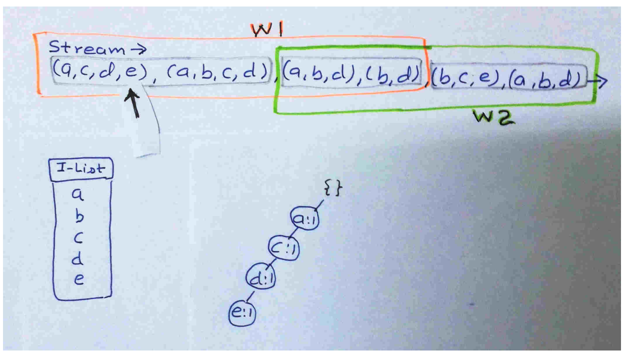
Fig. 3. The CPS-tree construction algorithm.

CPS-tree construction

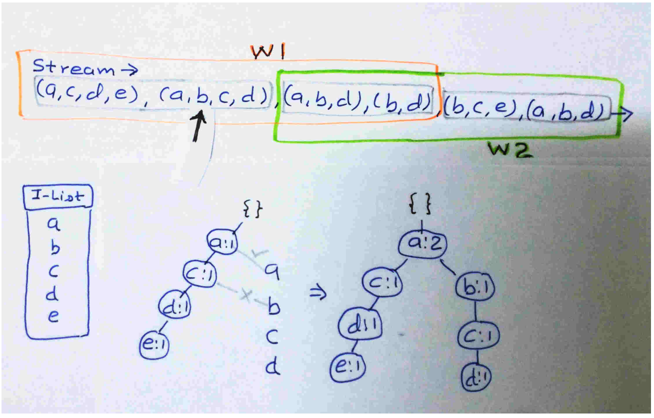


(a) (b) Initial empty CPS-tree

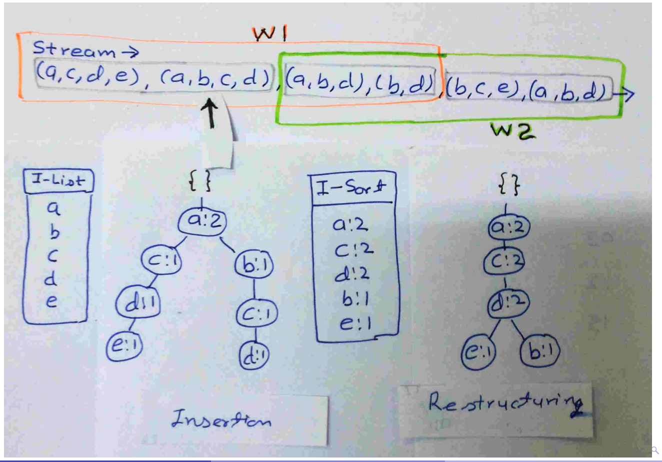
CPS-tree construction



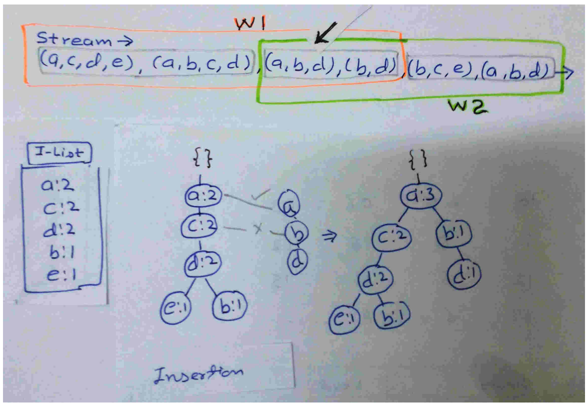
CPS-tree construction



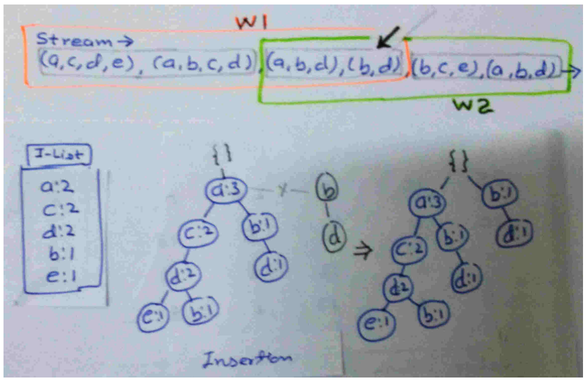
CPS-tree construction



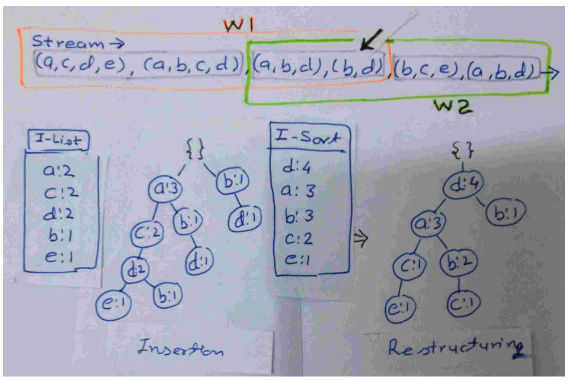
CPS-tree construction



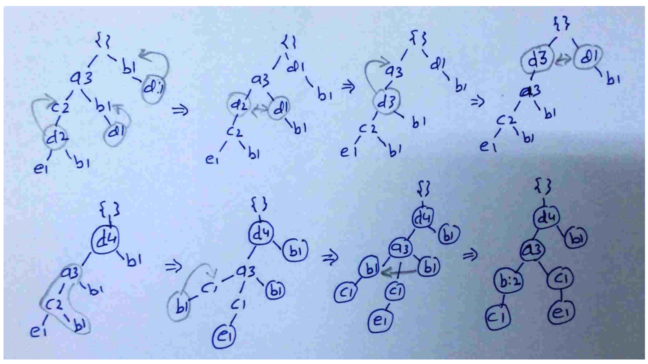
CPS-tree construction



CPS-tree construction

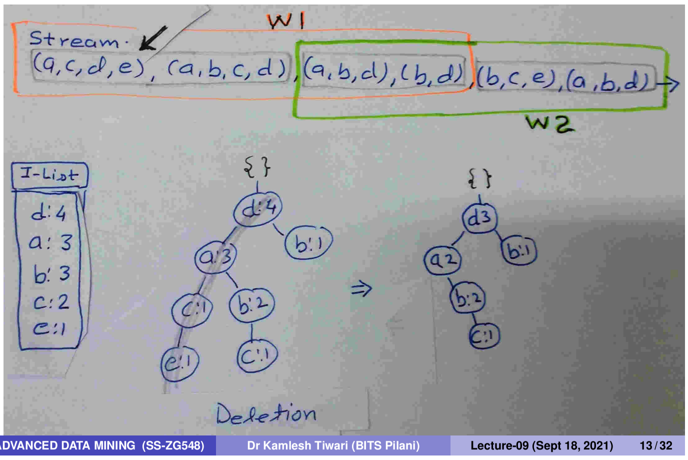


CPS-tree construction

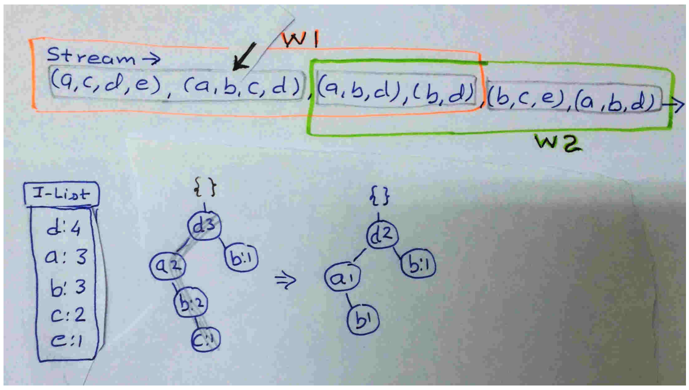


I-Sorted: d,a,b,c,e

CPS-tree construction

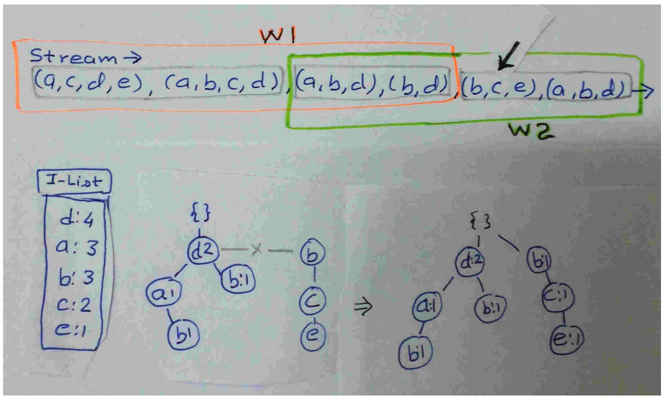


CPS-tree construction



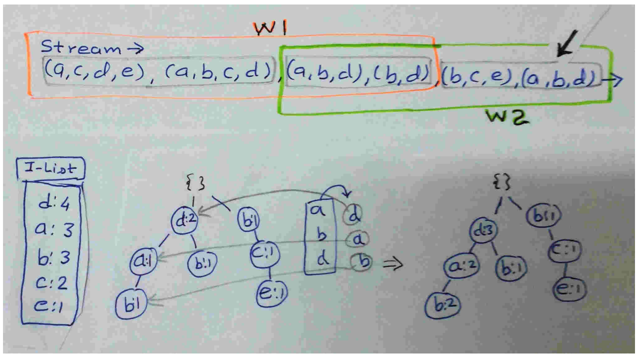
I-Sorted: d,a,b,c,e

CPS-tree construction



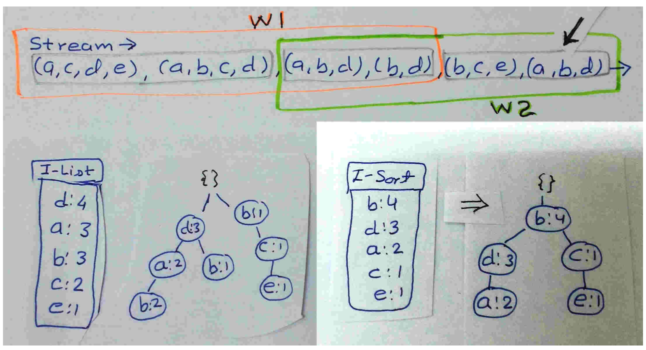
I-Sorted: d,a,b,c,e

CPS-tree construction



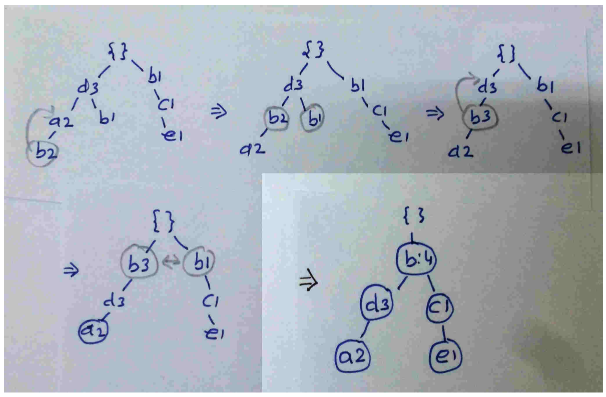
I-Sorted: d,a,b,c,e

CPS-tree construction



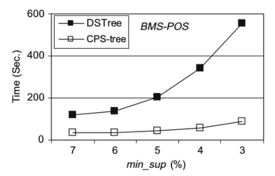
I-Sorted: b, d, a, c, e

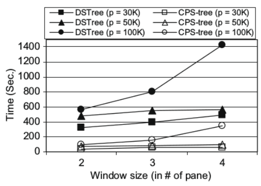
CPS-tree construction



I-Sorted: b, d, a, c, e

CPS-tree Performance





Clustering over Evolving Data Stream (DenStream)

Clustering over Evolving Data Stream

With limited memory and one-pass constraint one want to determine arbitrary number of clusters of arbitrary shape by efficiently handling outliers.

Use DenStream [2]

- Damped window model: Weights with time t are $f(t) = 2^{-\lambda . t}$. (other models landmark and sliding window)
- core-micro-cluster, potential-micro-cluster and outlier-micro-cluster structures
- Guarantees the precision of the weights of the micro-clusters

[2] Cao, Feng and Ester, Martin and Qian, Weining and Zhou, Aoying, "Density-Based Clustering over an Evolving Data Stream with Noise.", in International Conference on Data Mining, pages 328–339, vol 6, SIAM, 2006

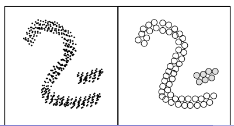
Clustering over Evolving Data Stream

- Beside limited memory and one-pass constraints, we require
  - No assumption on the number of clusters,
  - Discovery of clusters with arbitrary shape and
  - Ability to handle outliers
- DenStream [3], is a new approach for discovering clusters in an evolving data stream.
- Uses core-micro-cluster to summarize the clusters
- Along with potential core-micro-cluster and outlier micro-cluster structures
- Designed a pruning strategy that guarantees the precision of the weights of the micro-clusters
- User damped window model. Weights with time t are $f(t) = 2^{-\lambda . t}$. (other models landmark and sliding window)

[3] Cao, Feng and Ester, Martin and Qian, Weining and Zhou, Aoying, "Density-Based Clustering over an Evolving Data Stream with Noise.", in International Conference on Data Mining, pages 328–339, vol 6, SIAM, 2006

Clustering over Evolving Data Stream

- **Definition (core object)** It is an object in whose ϵ neighborhood the overall weight of data points is at least μ
- **Definition (density-area)** A density area is defined as the union of the ϵ neighborhoods of core objects
- **Definition (core-micro-cluster)** At time t it is defined as CMC(w, c, r) for a group of close points $p_{i_1}, p_{i_2}, ..., p_{i_n}$ with time stamp $T_{i_1}, T_{i_2}, ..., T_{i_n}$. $w = \sum_{j=1}^{n} f(t - T_{i_j})$, $w \geq \mu$ is the weight. $c = \frac{\sum_{j=1}^{n} f(t-T_{i_j})p_{i_j}}{w}$ is the center, $r = \frac{\sum_{j=1}^{n} f(t-T_{i_j})dist(p_{i_j}, c)}{w}$, $r \leq \epsilon$ is the radius, where $dist(p_{i_j}, c)$ denotes the Euclidean distance between point p_{i_j} and center c
- When a clustering request arrives, each c-micro-cluster will be labeled to get the final result.



Clustering over Evolving Data Stream

Definition (potential-micro-cluster) A potential-micro-cluster (or p-micro-cluster) at time t for a group of close points $p_{i_1}, p_{i_2}, ..., p_{i_n}$ with time stamp $T_{i_1}, T_{i_2}, ..., T_{i_n}$ is defined as $\{\overline{CF^1}, \overline{CF^2}, w\}$. $w = \sum_{j=1}^{n} f(t - T_{i_j})$, $w \geq \beta\mu$ is the weight. β, $0 < \beta \leq 1$, is the parameter to determine the threshold of outlier relative to c-micro-clusters. $\overline{CF^1} = \sum_{j=1}^{n} f(t - T_{i_j})p_{i_j}$ is the weighted linear sum of the points. $\overline{CF^2} = \sum_{j=1}^{n} f(t - T_{i_j})p_{i_j}^2$ is the weighted squired linear sum of the points. Centre of p-micro-cluster is $c = \frac{\overline{CF^1}}{w}$ and the radius of a p-micro-cluster $r = \sqrt{\frac{\overline{CF^2}}{w} + (\frac{\overline{CF^1}}{w})^2}$, $r \leq \epsilon$

Clustering over Evolving Data Stream

Definition (outlier-micro-cluster) An outlier-micro-cluster (or o-micro-cluster) at time t for a group of close points $p_{i_1}, p_{i_2}, ..., p_{i_n}$ with time stamp $T_{i_1}, T_{i_2}, ..., T_{i_n}$ is defined as $\{\overline{CF^1}, \overline{CF^2}, w, t_0\}$. The definition of $w, \overline{CF^1}, \overline{CF^2}$ center and radius are the same as p-micro-cluster. $t_0 = T_{i_1}$, denotes the creation time of o-micro-cluster which is used to define the life span of o-micro-cluster. However $w < \beta\mu$.

Note: p-micro-cluster and o-micro-cluster can be maintained incremently.

Clustering Algorithm has two parts:

- Online part of micro-cluster maintenance
- Offline part of generation of final clusters, on demand of user

Merging of P

p-micro-clusters and o-micro-clusters are maintained in an online way.

Algorithm 1 Merging (p)

Try to merge p into its nearest p-micro-cluster c_p;
if r_p (the new radius of c_p) $\leq \epsilon$ **then**
  Merge p into c_p;
else
  Try to merge p into its nearest o-micro-cluster c_o;
  if r_o (the new radius of c_o) $\leq \epsilon$ **then**
    Merge p into c_o;
    if w (the new weight of c_o) $> \beta\mu$ **then**
      Remove c_o from outlier-buffer and create a new p-micro-cluster by c_o;
    end if
  else
    Create a new o-micro-cluster by p and insert it into the outlier-buffer;
  end if
end if

DenStream Algorithm

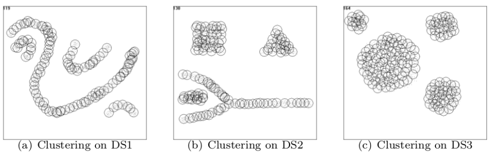
Algorithm 2 DenStream $(DS, \epsilon, \beta, \mu, \lambda)$

$T_p = \lceil \frac{1}{\lambda} \log(\frac{\beta\mu}{\beta\mu - 1}) \rceil$;
Get the next point p at current time t from data stream DS;
Merging(p);
if (t mod T_p)=0 **then**
  for each p-micro-cluster c_p **do**
    if w_p(the weight of c_p)$< \beta\mu$ **then**
      Delete c_p;
    end if
  end for
  for each o-micro-cluster c_o **do**
    $\xi = \frac{2^{-\lambda(t - t_o + T_p)} - 1}{2^{-\lambda T_p} - 1}$;
    if w_o(the weight of c_o)$< \xi$ **then**
      Delete c_o;
    end if
  end for
end if
if a clustering request arrives **then**
  Generating clusters;
end if

Performance

(a) Data set DS1 (b) Data set DS2 (c) Data set DS3

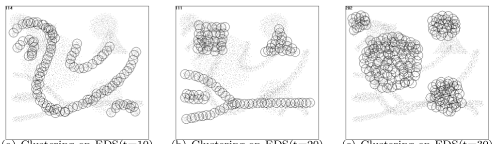
Figure 4: Synthetic data sets



(a) Clustering on DS1 (b) Clustering on DS2 (c) Clustering on DS3

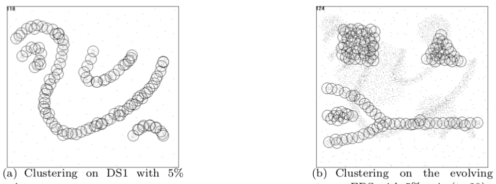
Figure 5: Clustering on DS1, DS2 and DS3

Performance



(a) Clustering on EDS(t=10) (b) Clustering on EDS(t=20) (c) Clustering on EDS(t=30)

Figure 6: Clustering on the evolving data stream EDS



(a) Clustering on DS1 with 5% noise (b) Clustering on the evolving stream EDS with 5% noise(t=20)

Figure 7: Clustering on data streams with noise

Performance

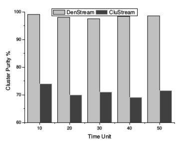


Figure 8: Clustering quality(EDS data stream, horizon=2, stream speed=2000)

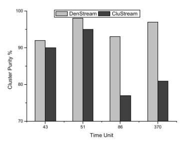


Figure 10: Clustering quality(Network Intrusion da set, horizon=1, stream speed=1000)

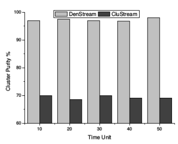


Figure 9: Clustering quality(EDS data stream, horizon=10, stream speed=1000)

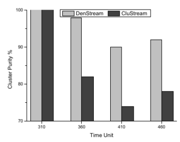


Figure 11: Clustering quality(Network Intrusion da set, horizon=5, stream speed=1000)

Performance

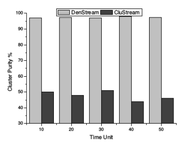


Figure 12: Clustering quality(EDS data stream with 1% noise, horizon=2, stream speed=2000)

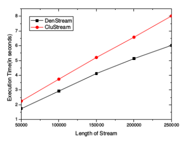


Figure 14: Execution time vs. length of stream(Netw Intrusion data set)

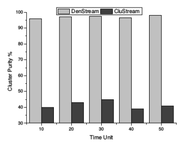


Figure 13: Clustering quality(EDS data stream with 5% noise, horizon=10, stream speed=1000)

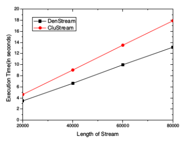


Figure 15: Execution time vs. length stream(Charitable Donation data set)

Performance

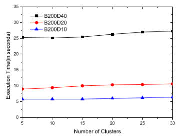


Figure 16: Execution time vs. number of clusters

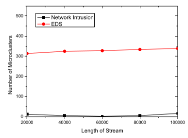


Figure 18: Memory usage

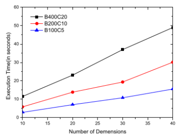


Figure 17: Execution time vs. dimensionality

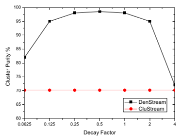


Figure 19: Clustering quality vs. decay factor λ

Thank You!

Thank you very much for your attention!

Queries ?