Outlier Detection

Experiment

- setting

	w	s	r	k	size	dimension
ForestCover	10,000	500	525	50	581,012	55
TAO	10,000	500	1.9	50	575,648	3
STOCK	10,0000	5000	0.45	50	1,048,575	1
Gauss	10,0000	5000	0.028	50	1,000,000	1
HPC	10,0000	5000	6.5	50	1,000,000	16
EM	10,0000	5000	115	50	1,000,000	7

Experiment

- CPU time per window

CPU Time	TAO	ForestCover(w=10k)	ForestCover(w=20k)	Gauss(WN=10)	HPC(WN=10)	EM(WN=10)	Stock(WN=10)
`exactStorm`	0.53	0.22	0.41	77.00	169.60	57.22	90.69
`approximateStorm`	0.32	0.21	0.40	74.37	80.53	124.51	104.67
`abstractC`	0.58	0.34	0.74	70.53	171.36	65.03	114.12
`lue`	0.73	0.41	0.76	84.08	——	72.40	118.60
`due`	0.57	0.37	0.71	85.54	138.36	54.53	71.30
`microCluster`	0.02	0.59	0.53	2.17	2.65	4.15	0.38
`microCluster_new`	0.017	0.69	0.52	1.21	0.92	3.42	0.13
`mesi`	0.03	0.10	0.084	73.86	0.90	0.69	0.50
`mesiWithHash`	0.46	0.28	0.29	20.27	53.23	5.16	24.03
NETS	0.004	0.05	0.05	0.011	0.016	1.82	0.00537

MCOD

- **Idea**: Micro-cluster based method, points inside the clusters are inliners. Points in a window are divided into PD and clusters.

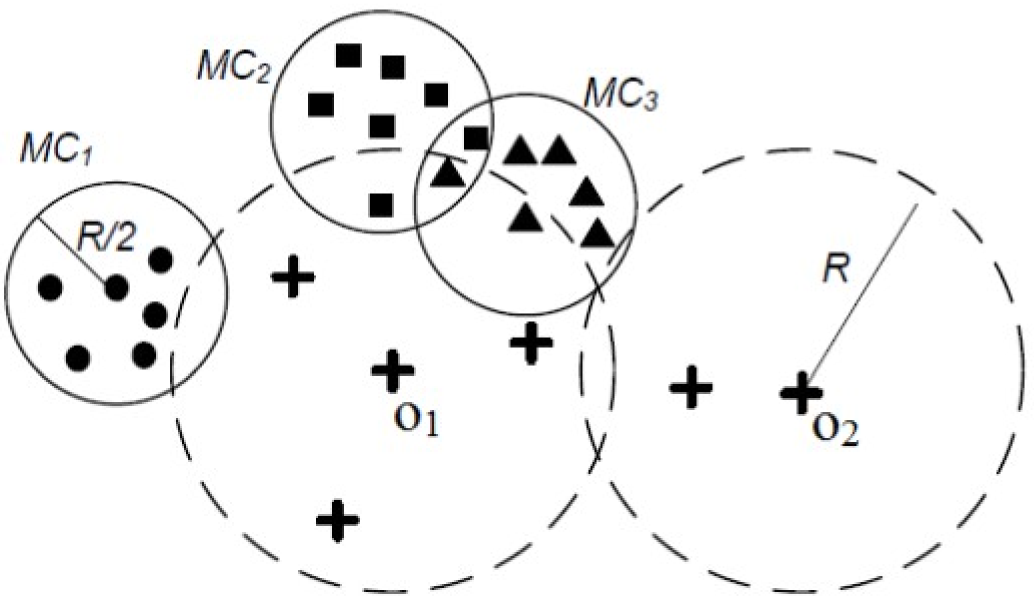


Figure 3: Example micro-clusters with $k = 4$ [10]

MCOD

- Data structures used:
  - **Mtree**: only data center stored there
  - **pd**
  - **micro_clusters**: center → all points
  - **event_queue**(earliest expired neighbor)
  - **Rmc:** list of cluster centers which is less than 1.5 R from p, which may include p's neighbor

MCOD

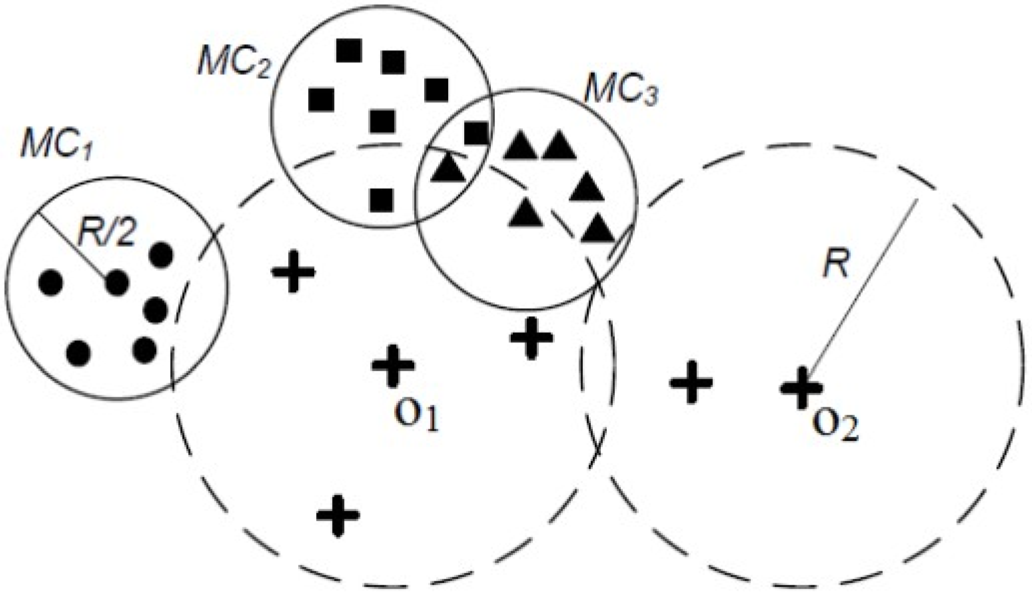


Figure 3: Example micro-clusters with $k = 4$ [10]

- O1.rmc contains MC1,MC2,MC3
- O2.rmc contains MC3

MCOD

- Algorithms:
  - **New slide processing:**
    - Find nearest center, if nearest center is less than 0.5r from p, **add p into the cluster c**, update c.associated_objects
    - Else, find all neighbors in pd, if can form a new cluster c, and update c.associated_objects
    - Else add it to pd and event queue if necessary

MCOD

- Algorithms:
  - **Expired slide processing:**
    - **If the point is in pd**, remove it from pd and outliers if necessary, remove expired preceding neighbor for point in outliers
    - **If the point is in cluster**, check if the cluster breaks.
      - If yes, remove cluster center, and treat each point as new point
    - **Update event queue**
      - Remove expired preceding neighbor for point in event queue and add points to outliers if necessary

Thresh LEAP

- **Idea:**
  - Minimal probing principle
  - Each slide has a smaller index

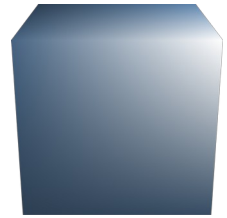
Thresh LEAP

- Data structures used:
  - o.ns: succeeding neighbor number
  - o.pre: previous slides -> neighbor number in each slide
  - Slide.triggerlist: the points which is affected by the slide’s expiration

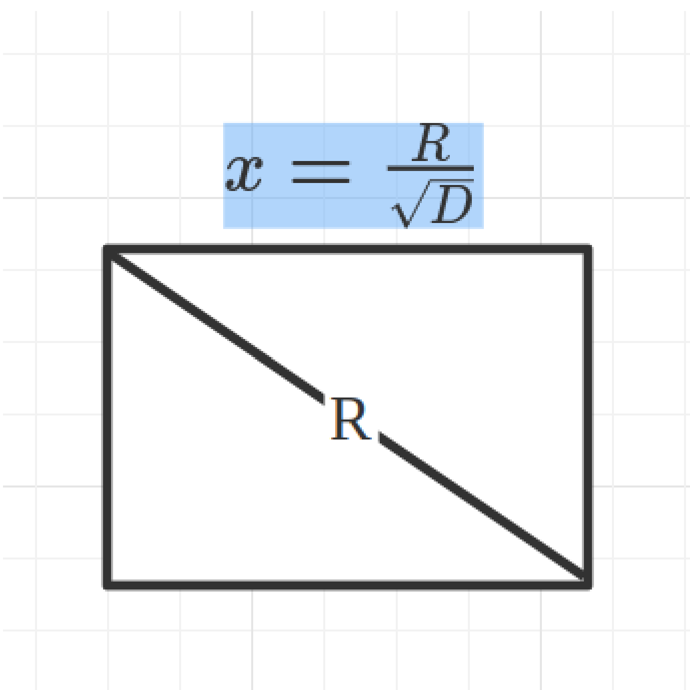
Thresh LEAP

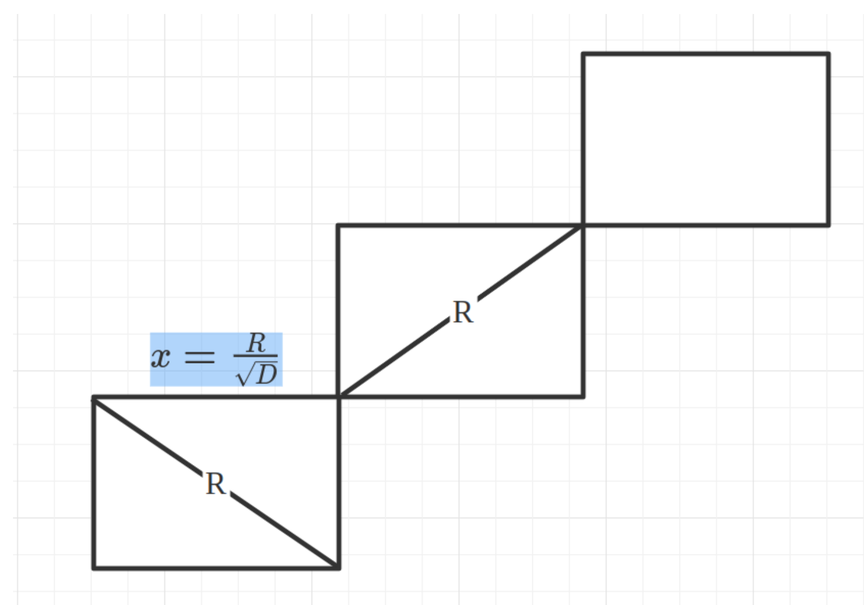
- Algorithms:
  - **New slide processing**
    - Find the number of neighbors in current slide
      - If n>=k, a safe point
      - If n<k, continue to search neighbor in previous slide and add point into the trigger list of those slide
  - **Expired slide processing**
    - For each point in trigger list, continue to find neighbor in succeeding slide, and check whether it is a outlier

NETS



- **Idea**:
  - Grid index
  - Net change
  - Dimensional filtering(sub-cell and full-cell)





NETS

- Data structures used
  - Slide: Cell Id → Cell instance
  - Slide-Delta: Changed Cell Id→ Change tuple count

NETS

- Algorithm
  - Find the optimal sub dimensions based on estimated cost and concentration ratio and sort data dimensions
  - **When a new slide comes and an old slide leaves**
    - **Calculate net Change:** update tuple count in sub-cell and full-cell in current window, get altered cells.
    - **Find outliers**:
      - Find all influenced cell exclude those inlier cells

NETS

- Algorithm
  - **Find outliers**:
    - Check if it is outlier cell in advance (less than k points in two-level neighbor)
    - If not, then for each non-determined tuple, find current tuple count inside the same cell and remove outdated preceding neighbor, if still more than k neighbor, recognize as inliner
    - Else, explore the rest slide to explore more neighbors