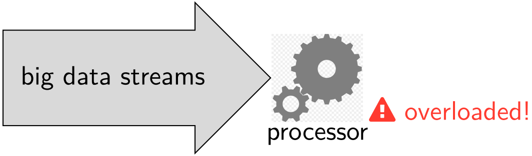
# Continuously Tracking Core Items in Data Streams with Probabilistic Decays

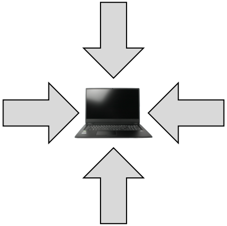
Junzhou Zhao Pinghui Wang Jing Tao Shuo Zhang John C.S. Lui
Xi'an Jiaotong University
The Chinese University of Hong Kong

# Expensive to Process Big Data Streams

- Data streams are ubiquitous:
  - email stream, tweets stream, news stream, etc
  - geo-location stream generated by taxis, IoT devices, LBSNs, etc
  - user consuming record stream from Amazon, Taobao, etc
- Applications:
  - real-time trending topic detection
  - network security monitoring
  - online collaborative filtering
- However, the high speed and large volume cause troubles.



# Two Ways for Handling Big Data Streams



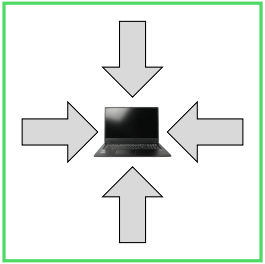
- scale up computation power

- reduce data complexity
  - cheap and green
  - rely on clever algorithms

# Two Ways for Handling Big Data Streams

- scale up computation power

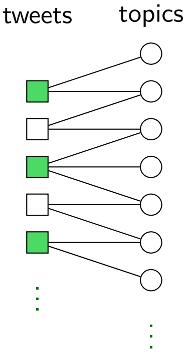
this work

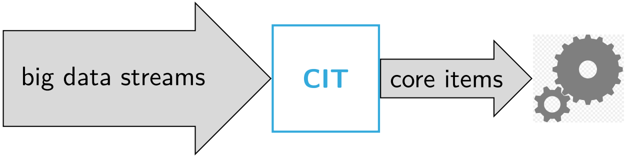


- reduce data complexity
  - cheap and green
  - rely on clever algorithms

# Need for Reducing Data Complexity

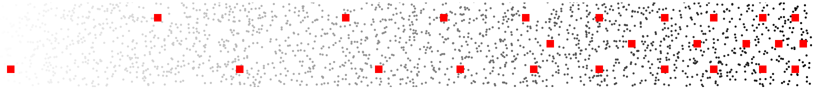
- too much redundant and noisy data
  - E.g., reading just a few tweets (or news articles) is enough to know the majority of the topics in the stream.
- **Core Items:** informative or representative items in a data stream.
- **Core Items Tracking (CIT):** a streaming algorithm that can continuously track core items in a data stream in real-time.



# The Right to be Forgotten

- We want the CIT algorithm to be able to gradually forget historical data in the stream.

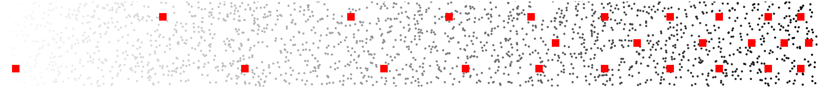


- CIT over insertion-only streams [KDD'14]:

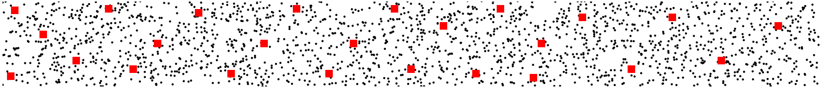
- CIT over sliding-window streams [WWW'17]:

# The Right to be Forgotten

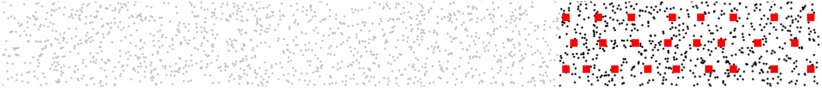
- We want the CIT algorithm to be able to gradually forget historical data in the stream.



- CIT over insertion-only streams [KDD'14]:



- CIT over sliding-window streams [WWW'17]:



# Outline

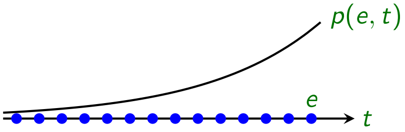
- Motivation ✓
- Problem formulation
- Algorithms
- Experiments
- Conclusion

# Measuring Informativeness of a Set of Items

- **Utility Function:** $f : 2^V \mapsto \mathbb{R}_{\geq 0}$ where $f(S)$ could measure the informativeness [JMLR'12, SIGIR'15], representativeness [KDD'14, CIKM'16, ICDE'18], diversity [WSDM'09], or coverage [PODS'14, KDD'15] of set $S \subseteq V$.
- $f(S)$ commonly satisfies the monotone submodular property, i.e.,
$$f(S \cup \{e\}) - f(S) \geq f(T \cup \{e\}) - f(T)$$
for all $S \subseteq T \subseteq V$ and $e \in V$.
  - captures the **dimension return** property [Nemhauser et al. 1978]

# Probabilistic-Decaying Stream (PDS) Model

- At time $t$, we let an item $e$ arrived at time $t_e \leq t$ participate in the analysis with a probability $p(e,t) = h_e(t - t_e)$.
- $h_e \colon \mathbb{Z}_{\geq 0} \mapsto [0,1]$ is an item-specific decaying function.
- $h_e(age)$ decreases as $age$ increases, e.g., $h_e(age) = p_e^{age}$, $p_e \in (0,1)$.

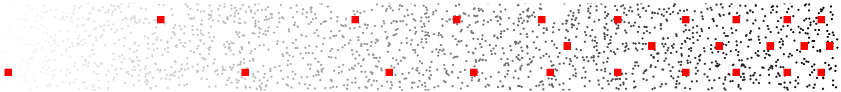


# The Core Items Tracking (CIT) Problem

- **Given** a monotone submodular utility function $f$, a PDS with item-specific decaying function $h_e$, and a budget $k > 0$
- **Want** to find a subset $S_t^* \subseteq V$ at any query time $t$, s.t.

$$S_t^* = \operatorname*{arg\,max}_{S \subseteq V \wedge |S| \leq k} \mathbb{E}_{h_e}[f(S)|\mathcal{D}_t]$$

where $\mathcal{D}_t \triangleq \{e \colon t_e \leq t\}$ denotes the items arrived before $t$.



prefer to choose more recent data items as core items

# Outline

- Motivation ✓
- Problem formulation ✓
- Algorithms
- Experiments
- Conclusion

# Expensive to Exactly Calculate $\mathbb{E}\left[f(S)|\mathcal{D}_t\right]$

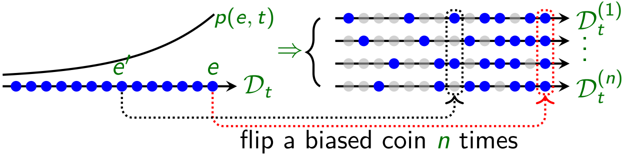
- To calculate $\mathbb{E}\left[f(S)|\mathcal{D}_t\right]$, we have to consider the participation possibility of each item in $S$, e.g.,

$$\mathbb{E}\left[f(\{a,b\})|\mathcal{D}_t\right] = \underbrace{p(a,t)p(b,t)f(\{a,b\})}_{\text{both } a \text{ and } b \text{ participate in the analysis}} + \underbrace{p(a,t)(1-p(b,t))f(\{a\})}_{\text{only } a \text{ participates in the analysis}} + \underbrace{(1-p(a,t))p(b,t)f(\{b\})}_{\text{only } b \text{ participates in the analysis}}.$$

- Exactly calculating $\mathbb{E}\left[f(S)|\mathcal{D}_t\right]$ requires $O(2^{|S|})$ oracle calls.

# Monte-Carlo Approximation

- Generate $n$ samples of the PDS, and estimate $\mathbb{E}[f(S)|\mathcal{D}_t]$.

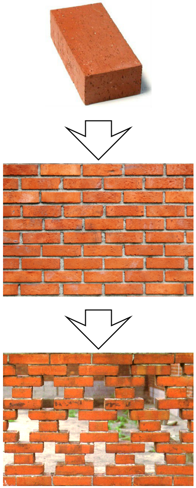



- By Monte-Carlo approximation, we have

$$F(S) \triangleq \frac{1}{n}\sum_{i=1}^{n} f(S \cap \mathcal{D}_t^{(i)}) \xrightarrow{a.s.} \mathbb{E}[f(S)|\mathcal{D}_t], \quad n \to \infty.$$

- The number of oracle calls reduces from $O(2^{|S|})$ to $O(n)$.
- $F(S)$ is still monotone and submodular.

# Overview

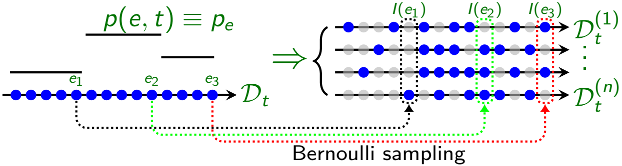


- **PNDCIT** can efficiently solve the CIT problem over a special kind of probabilistic non-decaying case.
- **PDCIT** uses PNDCIT as a building block to solve the CIT problem over general PDS.
- **PDCIT+** is designed to improve the efficiency of PDCIT.

| algorithm | #oracle calls | memory | approx. ratio |
|---|---|---|---|
| PNDCIT | $O(n\epsilon^{-1}\log k)$ | $O(nk\epsilon^{-1}\log k)$ | $1/2-\epsilon$ |
| PDCIT | $O(Ln\epsilon^{-1}\log k)$ | $O(Lnk\epsilon^{-1}\log k)$ | $1/2-\epsilon$ |
| PDCIT+ | $O(n\epsilon^{-2}\log^2 k)$ | $O(nk\epsilon^{-2}\log^2 k)$ | $1/4-\epsilon$ |

# PNDCIT: Probabilistic Non-Decaying Case

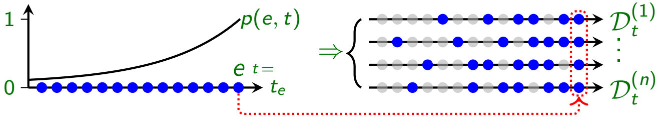
- Probabilistic non-decaying case: $p(e, t) \equiv p_e$
- The PDS can be converted to an insertion-only stream.

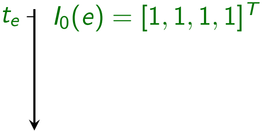



$$I(e_1) = [0, 0, 0, 1]^T$$
$$I(e_2) = [0, 1, 1, 1]^T$$
$$I(e_3) = [1, 0, 1, 0]^T$$

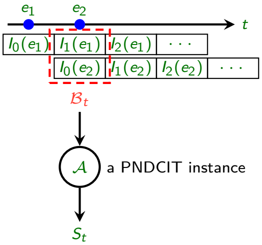
- $\{(e_1, p_{e_1}), (e_2, p_{e_2}), \ldots\} \Rightarrow \{I(e_1), I(e_2), \ldots\}$ where $I(e) \in \{0, 1\}^n$
- submodular optimization over insertion-only streams has been extensively studied [SDM'08, DAM'12, SPAA'13, KDD'14].

# PDCIT: Probabilistic Decaying Case

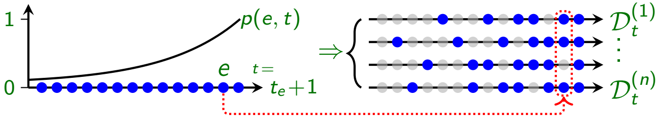


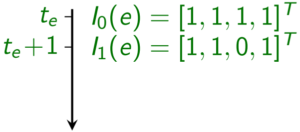




- At time $t$, denote those non-zero sample vectors by $\mathcal{B}_t$.
- Ideally, if we can feed $\mathcal{B}_t$ to a PNDCIT instance, we will get a quality guaranteed solution at time $t$.
- How to process $\mathcal{B}_t$ in a streaming fashion?



# PDCIT: Probabilistic Decaying Case

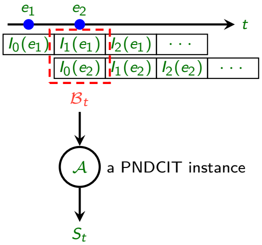


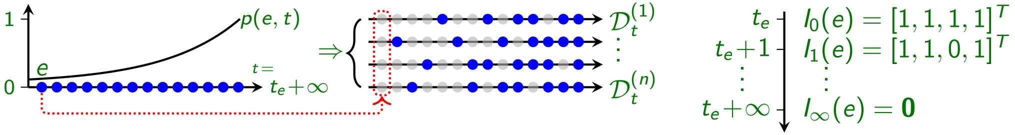

- At time $t$, denote those non-zero sample vectors by $\mathcal{B}_t$.

- Ideally, if we can feed $\mathcal{B}_t$ to a PNDCIT instance, we will get a quality guaranteed solution at time $t$.

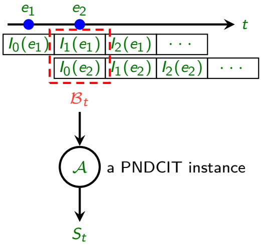
- How to process $\mathcal{B}_t$ in a streaming fashion?



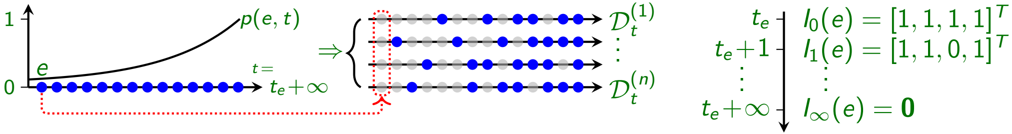
# PDCIT: Probabilistic Decaying Case



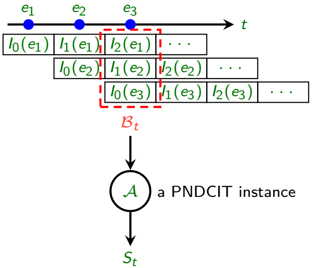

- At time $t$, denote those non-zero sample vectors by $\mathcal{B}_t$.
- Ideally, if we can feed $\mathcal{B}_t$ to a PNDCIT instance, we will get a quality guaranteed solution at time $t$.
- How to process $\mathcal{B}_t$ in a streaming fashion?



# PDCIT: Probabilistic Decaying Case



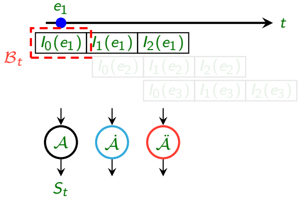

- At time $t$, denote those non-zero sample vectors by $\mathcal{B}_t$.
- Ideally, if we can feed $\mathcal{B}_t$ to a PNDCIT instance, we will get a quality guaranteed solution at time $t$.
- How to process $\mathcal{B}_t$ in a streaming fashion?



# PDCIT: Probabilistic Decaying Case

- Assume $I_l(e) = \mathbf{0}$ for $l \geq L, \forall e$. Thus an item has at most $L$ non-zero sample vectors $\{I_l(e)\}_{0 \leq l < L}$.
- PDCIT runs $L$ PNDCIT instances, and processes each arrived item's sample vectors in parallel.
- At next time step, because each $I_l(e)$ evolves to $I_{l+1}(e)$, which has been processed by PNDCIT instances on the right side, thus we shift these $L$ PNDCIT instances one unit to the left.

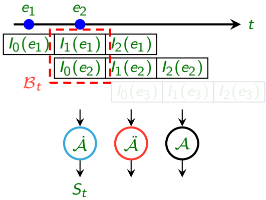
$L = 3$



# PDCIT: Probabilistic Decaying Case

- Assume $I_l(e) = \mathbf{0}$ for $l \geq L, \forall e$. Thus an item has at most $L$ non-zero sample vectors $\{I_l(e)\}_{0 \leq l < L}$.
- PDCIT runs $L$ PNDCIT instances, and processes each arrived item's sample vectors in parallel.
- At next time step, because each $I_l(e)$ evolves to $I_{l+1}(e)$, which has been processed by PNDCIT instances on the right side, thus we shift these $L$ PNDCIT instances one unit to the left.

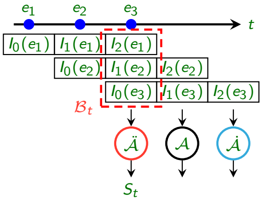
$L = 3$



# PDCIT: Probabilistic Decaying Case

- Assume $I_l(e) = \mathbf{0}$ for $l \geq L, \forall e$. Thus an item has at most $L$ non-zero sample vectors $\{I_l(e)\}_{0 \leq l < L}$.
- PDCIT runs $L$ PNDCIT instances, and processes each arrived item's sample vectors in parallel.
- At next time step, because each $I_l(e)$ evolves to $I_{l+1}(e)$, which has been processed by PNDCIT instances on the right side, thus we shift these $L$ PNDCIT instances one unit to the left.

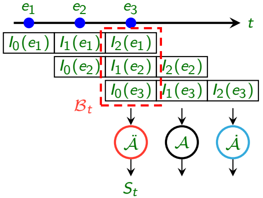
$L = 3$

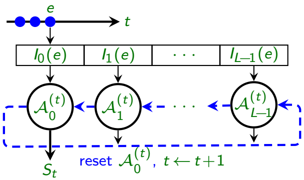


# PDCIT: Probabilistic Decaying Case

- Assume $I_l(e) = \mathbf{0}$ for $l \geq L, \forall e$. Thus an item has at most $L$ non-zero sample vectors $\{I_l(e)\}_{0 \leq l < L}$.
- PDCIT runs $L$ PNDCIT instances, and processes each arrived item's sample vectors in parallel.
- At next time step, because each $I_l(e)$ evolves to $I_{l+1}(e)$, which has been processed by PNDCIT instances on the right side, thus we shift these $L$ PNDCIT instances one unit to the left.

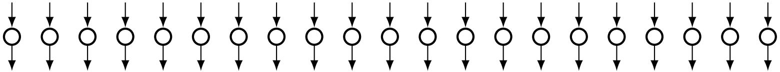
$L = 3$



# PDCIT+: Improve Efficiency

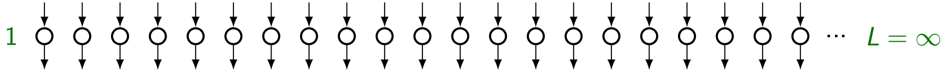
- PDCIT needs to maintain $L$ PNDCIT instances. What if $L$ is extremely large?

1  $\cdots$ $L = \infty$

- **Idea**: selectively maintain just a few PNDCIT instances, that can well approximate the rest
- Similar to using a histogram to approximate a curve

# PDCIT+: Improve Efficiency

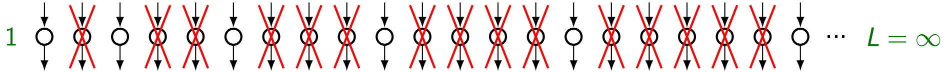
- PDCIT needs to maintain $L$ PNDCIT instances. What if $L$ is extremely large?




- **Idea**: selectively maintain just a few PNDCIT instances, that can well approximate the rest
- Similar to using a histogram to approximate a curve

# PDCIT+: Improve Efficiency

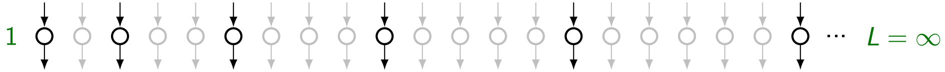
- PDCIT needs to maintain $L$ PNDCIT instances. What if $L$ is extremely large?




- **Idea**: selectively maintain just a few PNDCIT instances, that can well approximate the rest
- Similar to using a histogram to approximate a curve

# PDCIT+: Improve Efficiency

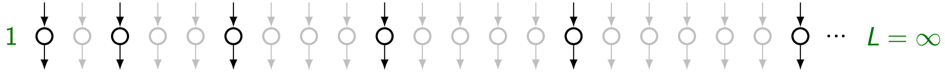
- PDCIT needs to maintain $L$ PNDCIT instances. What if $L$ is extremely large?



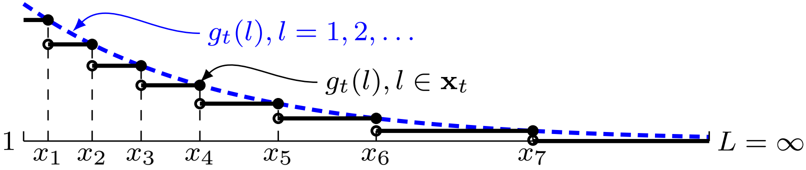

- **Idea**: selectively maintain just a few PNDCIT instances, that can well approximate the rest
- Similar to using a histogram to approximate a curve

# PDCIT+: Improve Efficiency

- PDCIT needs to maintain $L$ PNDCIT instances. What if $L$ is extremely large?




- **Idea**: selectively maintain just a few PNDCIT instances, that can well approximate the rest
- Similar to using a histogram to approximate a curve




checkout our paper for more details

# Outline

- Motivation ✓
- Problem formulation ✓
- Algorithms ✓
- Experiments
- Conclusion

# Data

- DBLP author stream: an item is a set of conferences that the author attended before
- MemeTracker article stream: an item is a set of memes that the article contains
- math.StackExchange question stream: an item is a set of tags that the question contains
- StackOverflow question stream: an item is a set of tags that the question contains

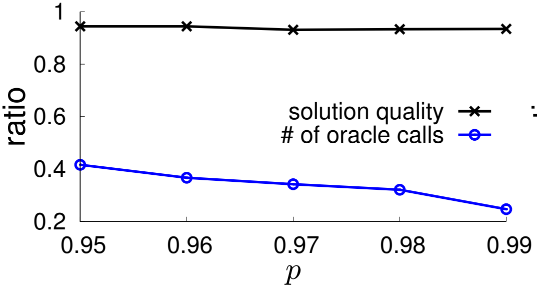
| data stream | item | length | time period |
|---|---|---|---|
| DBLP | author | $371,690$ | 1936 - 2018 |
| MemeTracker | article | $714,072$ | $1/2009$ (one month) |
| math.StackExchange | question | $955,284$ | $7/2010$ - $6/2018$ |
| StackOverflow | question | $2,904,450$ | $1/2015$ - $3/2016$ |

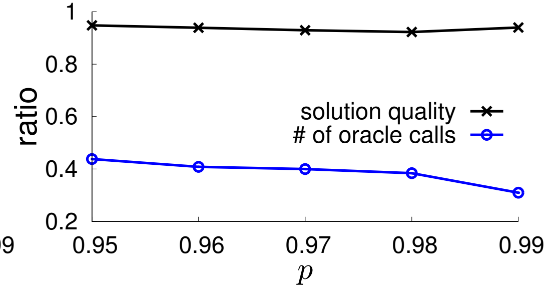
# Settings

- **Goal:** maintain $k$ most representative items that jointly have the maximum coverage, i.e., $f(S) = |\cup_{e \in S} e|$.
- **Decaying Function:** $h_e(x) = p^x, p \in (0, 1)$.
- **Baselines:**
  - GREEDY will serve as an upper bound
  - Unbiased Reservoir Sampling
  - Biased Reservoir Sampling

# PDCIT vs PDCIT+

- How close is their solution quality?
- How significant can PDCIT+ reduce the number of oracle calls?



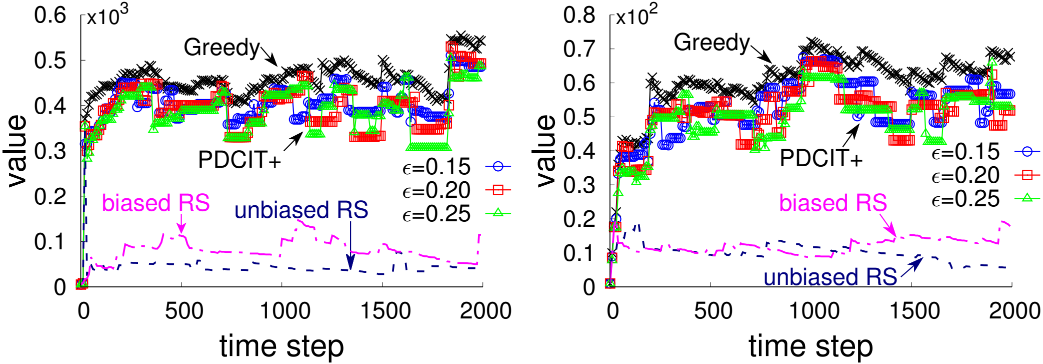




left: DBLP, right: MemeTracker, $\epsilon = 0.2$, $k = 10$, $n = 20$, $L = 100$

**achieves similar solution quality, reduces more than a half of oracle calls**

# Solution Quality

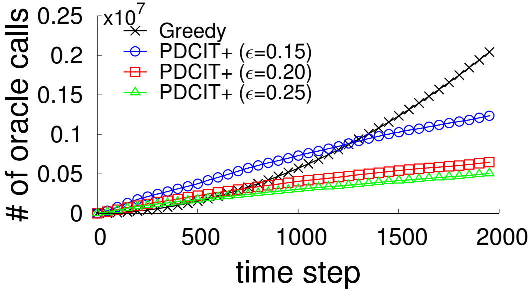
- Comparing solution quality of different methods (higher is better)

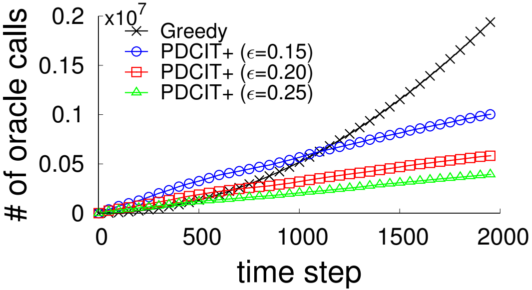



left: DBLP, right: MemeTracker, $p = 0.999$, $k = 10$, $n = 20$

# Scalability

- Comparing the number of oracle calls of different methods (lower is better)




left: DBLP, right: MemeTracker, $p = 0.999, k = 10, n = 20$

# Outline

- Motivation ✓
- Problem formulation ✓
- Algorithms ✓
- Experiments ✓
- Conclusion

# Conclusion

- The optimization problem behind the CIT problem is to solve the streaming submodular optimization problem over probabilistic-decaying streams.
- We designed two streaming algorithms, namely PDCIT and PDCIT+, to address this CIT problem. They use PNDCIT as a building block.
- These techniques are verified on several public available data streams. The results demonstrate the effectiveness of these methods.

# Thanks for listening!

junzhou.zhao@xjtu.edu.cn