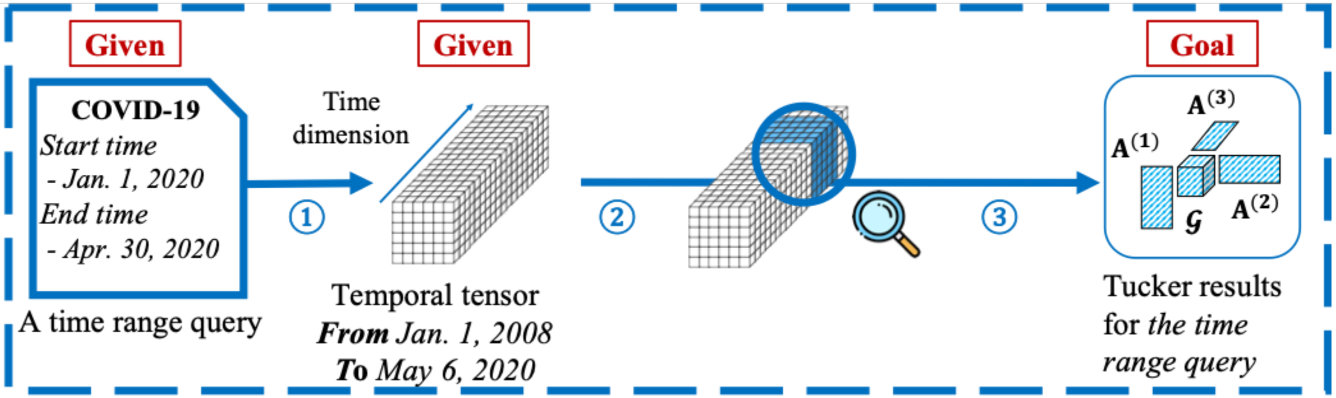
Fast and Memory-Efficient Tucker Decomposition for Answering Diverse Time Range Queries

Jun-Gi Jang and U Kang
Data Mining Lab
Dept. of CSE
Seoul National University

Overview

- **Q.** Given a temporal dense tensor and a time range (e.g., January -March 2019), how can we efficiently analyze the tensor in the given time range?
- **A. Zoom-Tucker** enables us to analyze the tensor in the given range, quickly and memory-efficiently

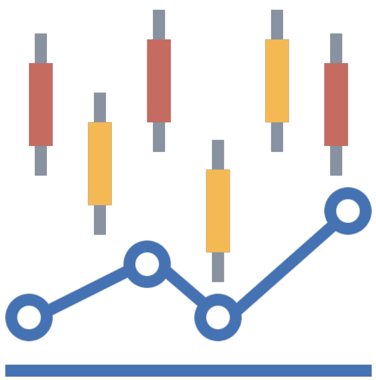


Outline

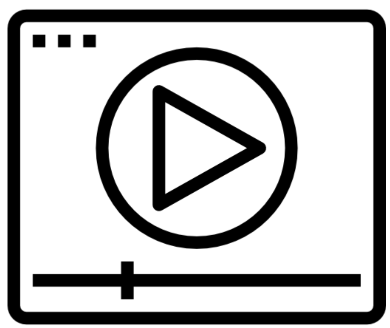
- **Introduction**
- Proposed Method
- Experiments
- Conclusion

Temporal Dense Tensors

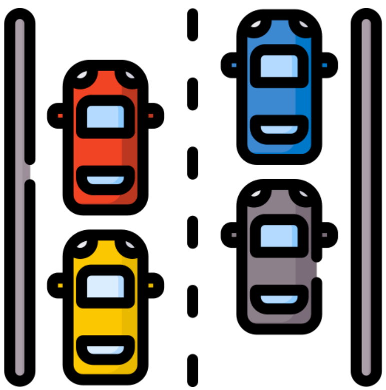
- Several real-world data are represented as **temporal dense tensors**
  - One dimension corresponds to time
  - Most entries of a tensor are measured



Stocks
3-way tensor
Index: (stock, feature, time)
Value: measurement



Video
3-way tensor
Index: (width, height, time)
Value: measurement

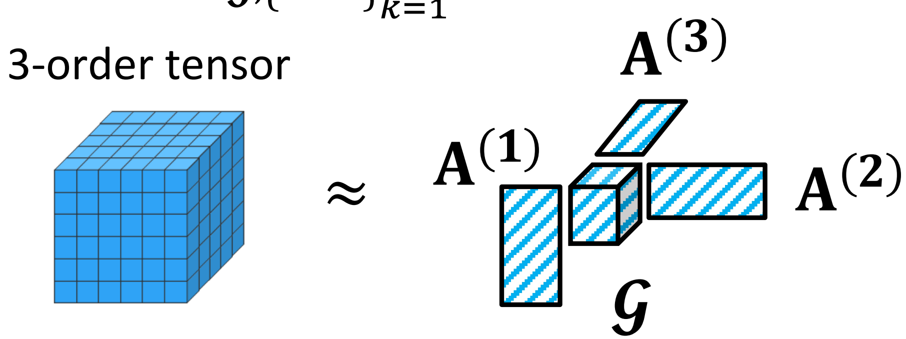


Traffic Volume
3-way tensor
Index: (sensor, frequency, time)
Value: measurement

Tucker Decomposition

- **Given** an N-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, rank $J_1, \ldots, J_N$
- **Obtain** factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J}$ for $n = 1, \ldots, N$ and core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$
- **Objective function**

$$\min_{\boldsymbol{\mathcal{G}}, \{\mathbf{A}^{(k)}\}_{k=1}^{N}} \left\| \mathbf{X}_{(n)} - \mathbf{A}^{(n)} \mathbf{G}_{(n)} \left(\bigotimes_{k \neq n}^{N} \mathbf{A}^{(k)T} \right) \right\|_F^2$$

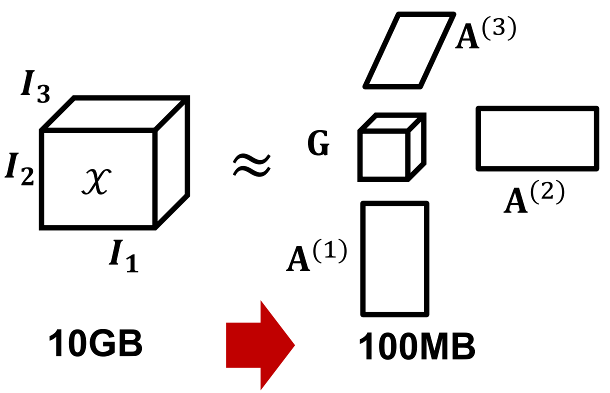


$\otimes$: Kronecker product
$\left(\bigotimes_{k \neq n}^{N} \mathbf{A}^{(k)T} \right)$: the entire Kronecker product of $\mathbf{A}^{(k)T}$ in descending order for $k = N, \ldots, n+1, n-1, \ldots, 1$
$\mathbf{X}_{(n)}$: mode-n matricization of $\boldsymbol{\mathcal{X}}$
$\mathbf{G}_{(n)}$: mode-n matricization of $\boldsymbol{\mathcal{G}}$

- **ALS (Alternating Least Square) approach**
  - Iteratively updates a factor matrix of a mode while fixing all factor matrices of other modes

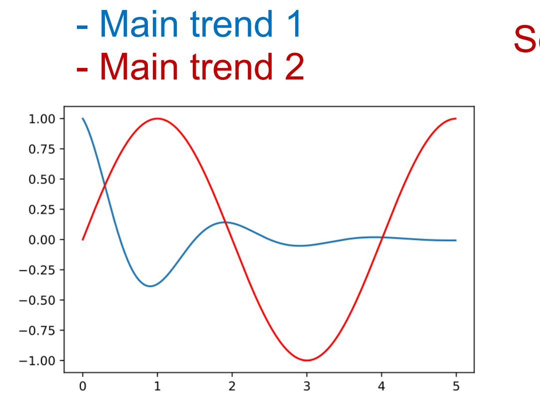
Applications

- **Several applications for Tucker decomposition**
  - Dimensionality reduction, concept discovery, trend analysis, anomaly detection, and clustering



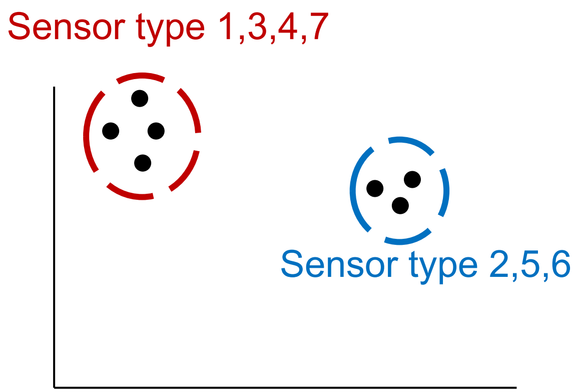
Dimensionality Reduction

Lossy compression for a temporal dense tensor



Trend Analysis

Find main trends using latent factors of the time dimension

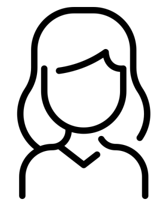


Clustering

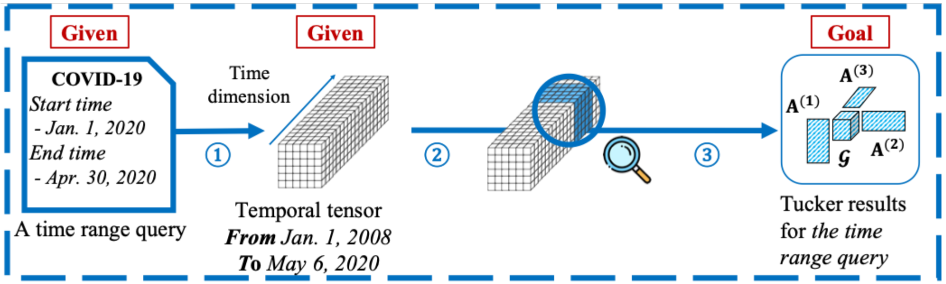
Find similar objects using latent factors

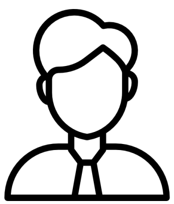
Time Range Query

- Several users are interested in investigating patterns of **diverse** time ranges using Tucker decomposition



Find similar stocks in **COVID-19**





Compare **monthly** patterns in a stock market

Analyze patterns for **the entire time range**

⋮

Q. How can we answer diverse time range queries **quickly** and **memory-efficiently**?

Problem Definition

Problem: time range query problem on temporal dense tensor

- **Given**
  - A temporal dense tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times I_N}$
    - Assume the last dimension is the time dimension
  - A time range query $[t_s, t_e]$
- **Find**
  - The Tucker results of the sub-tensor of $\boldsymbol{\mathcal{X}}$ included in the given range $[t_s, t_e]$
    - The Tucker results include factor matrices $\tilde{\mathbf{A}}^{(1)}, \ldots, \tilde{\mathbf{A}}^{(N)}$, and core tensor $\tilde{\boldsymbol{\mathcal{G}}}$

Challenges

- **(Limitation)** Previous works are tailored for performing Tucker decomposition of only the whole tensor once
  - For each time range, performing Tucker decomposition from scratch requires **high time and space costs**
  - A few methods with a preprocessing phase are still **unsatisfactory in terms of time, space, and accuracy** on the problem
    - Before time range is given, they preprocess a given tensor, and perform Tucker decomposition with the preprocessed tensor for each time range query
- Need to address the following challenges
  1. Deal with diverse time ranges
  2. Minimize computational costs
  3. Avoid huge intermediate data

Outline

- Introduction
- **Proposed Method**
- Experiments
- Conclusion

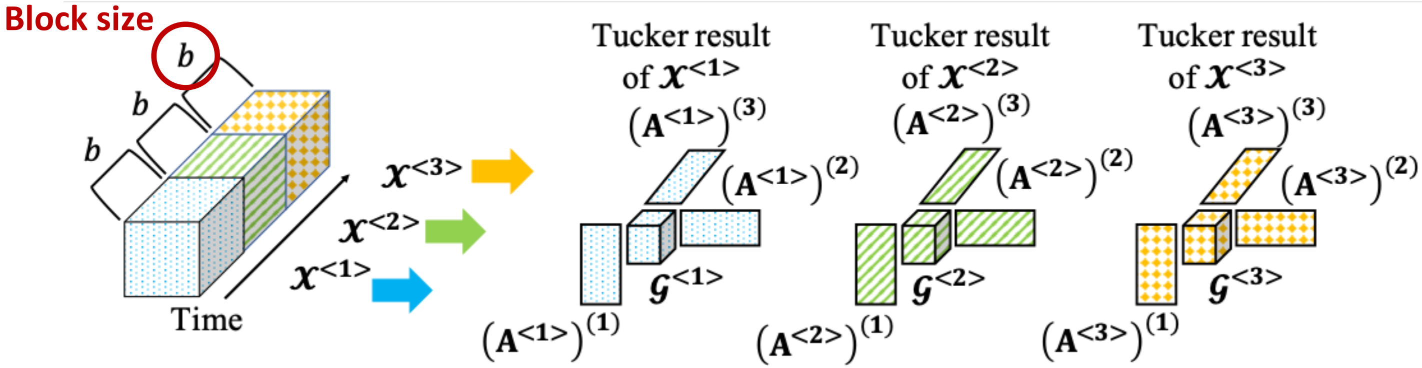
Proposed Method - Overview

- We propose Zoom-Tucker (Zoomable Tucker decomposition)
  - A **fast** and **memory-efficient** Tucker decomposition method for diverse time range queries

- Zoom-Tucker consists of the two phases
  - **Preprocessing phase**
    - Compresses a given temporal tensor block by block before time range queries are given
  - **Query phase**
    - Answers a given time range query by exploiting compression results obtained in the preprocessing phase

Preprocessing Phase

- Before time range queries are given, Zoom-Tucker compresses a given temporal tensor block by block along the time dimension
  - Split a given temporal tensor into sub-tensors along the time dimension
  - For each block tensor, we perform Tucker decomposition

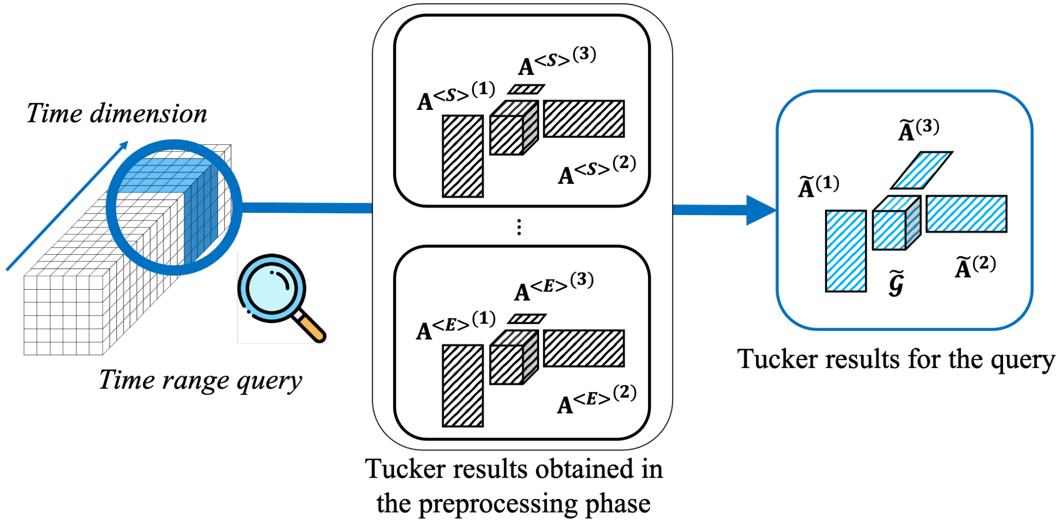


$\mathcal{X}^{<i>}$: i-th block tensor
$(\mathbf{A}^{<i>})^{(n)}$: the n-th factor matrix of i-th block tensor
$\mathcal{G}^{<i>}$: the core tensor of i-th block tensor

- The advantages of the preprocessing phase
  1. Generate **small results** compared to an input tensor (support high efficiency of the query phase)
  2. Capture **local temporal information** (reduce error increase)

Query Phase

- **Goal**
  - Given a time range query, efficiently perform Tucker decomposition of the sub-tensor included in the time range



- A naïve approach would reconstruct the preprocessed results in a given range, and then compute Tucker decomposition
  - Unsatisfactory in terms of time and space costs

Query Phase

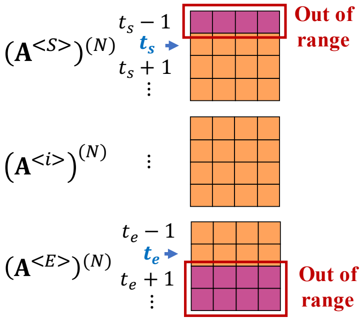
$\widetilde{\mathbf{X}}_{(n)}$: the mode-$n$ matricized version of a sub-tensor for a time range
$\widetilde{\mathbf{A}}^{(n)}$: the n-th factor matrix
$\widetilde{\mathbf{G}}_{(n)}$: the core tensor

- **Optimization problem**

$$\min_{\widetilde{\mathcal{G}},\{\widetilde{\mathbf{A}}^{(k)}\}_{k=1}^{N}} \left\| \widetilde{\mathbf{X}}_{(n)} - \widetilde{\mathbf{A}}^{(n)} \widetilde{\mathbf{G}}_{(n)} \left(\otimes_{k \neq n}^{N} \widetilde{\mathbf{A}}^{(k)T} \right) \right\|_F^2$$

- **Procedure**

  1. Given a time range $[t_s, t_e]$, load Tucker results included in the time range
  2. Adjust the first and the last blocks to fit the range
  3. **Alternatively update factor matrices and core tensor using the Tucker results**
  4. Repeatedly performs step 3 until convergence



- **Update rule**

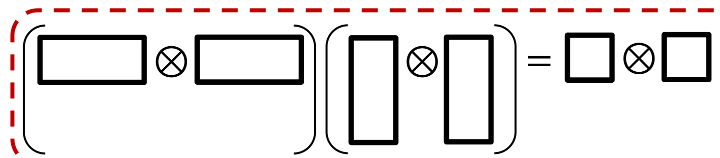
$$\widetilde{\mathbf{A}}^{(n)} \leftarrow \widetilde{\mathbf{X}}_{(n)} \left(\otimes_{k \neq n}^{N} \widetilde{\mathbf{A}}^{(k)} \right) \widetilde{\mathbf{G}}_{(n)}^{T} \left(\widetilde{\mathbf{G}}_{(n)} \left(\otimes_{k \neq n}^{N} \widetilde{\mathbf{A}}^{(k)T} \widetilde{\mathbf{A}}^{(k)} \right) \widetilde{\mathbf{G}}_{(n)}^{T} \right)^{-1}$$

$$= \boxed{\widetilde{\mathbf{X}}_{(n)} \left(\otimes_{k \neq n}^{N} \widetilde{\mathbf{A}}^{(k)} \right) \widetilde{\mathbf{G}}_{(n)}^{T}} \left(\mathbf{C}^{(n)} \right)^{-1}$$

Efficiently compute this term

Main Ideas

- Main ideas for the efficient query phase
  1. In our update rule, replace a sub-tensor $\tilde{\mathcal{X}}$ of a time range query with **preprocessed block Tucker results**
  2. **Decouple** block Tucker results in a time range query, obtained in the preprocessing phase
  3. Carefully determining the **order** of computations
- Exploit the following key techniques



Contain

$$[\mathbf{A}^T \quad \mathbf{B}^T]\begin{bmatrix}\mathbf{C}\\ \mathbf{D}\end{bmatrix} = \mathbf{A}^T\mathbf{C} + \mathbf{B}^T\mathbf{D}$$

Matrix multiplication of block matrices

Contain

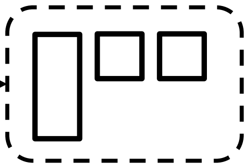
$$(\mathbf{E}^T \otimes \mathbf{F}^T)(\mathbf{G} \otimes \mathbf{H}) = \mathbf{E}^T\mathbf{G} \otimes \mathbf{F}^T\mathbf{H}$$

The mixed product property

- With the above two techniques, we **significantly reduce** the intermediate data and computational costs for a given query

Non-temporal Mode

- By exploiting the two techniques, we decouple block results along the time dimension
- Compute each block and then sum up the results



$$\tilde{\mathbf{A}}^{(n)} \leftarrow \boxed{\sum_{i=S}^{E}} \left(\mathbf{A}^{<i>}\right)^{(n)} \boxed{\left(\mathbf{B}^{<i>}\right)^{(n)}} \left(\mathbf{C}^{(n)}\right)^{-1}$$

Using matrix multiplication of block matrices

Efficiently compute it using the mixed product property

- In $\left(\mathbf{B}^{<i>}\right)^{(n)} \in \mathbb{R}^{J \times J}$ and $\mathbf{C}^{(n)} \in \mathbb{R}^{J \times J}$, there are the following computations (low time and space costs)
  - $\mathbf{U}^T\mathbf{V} \in \mathbb{R}^{J \times J}$ $(\mathbf{U}, \mathbf{V} \in \mathbb{R}^{I \times J})$
  - n-mode products between a core tensor and $\mathbf{U}^T\mathbf{V}$
  - Matrix multiplication between mode-n matricization of two core tensors

Temporal Mode & Core Tensor

- Decouple block results along the time dimension
- Compute each block

$$\widetilde{\mathbf{A}}^{(N)} \leftarrow \begin{bmatrix} (\mathbf{A}^{<S>})^{(N)}(\mathbf{B}^{<S>})^{(N)} \\ \vdots \\ (\mathbf{A}^{<E>})^{(N)}(\mathbf{B}^{<E>})^{(N)} \end{bmatrix} \left(\mathbf{C}^{(N)}\right)^{-1}$$

- $\left(\mathbf{B}^{<i>}\right)^{(N)} \in \mathbb{R}^{J\times J}$ and $\mathbf{C}^{(N)} \in \mathbb{R}^{J\times J}$ are also computed with low computational costs
- At the end of each iteration, the core tensor is also updated by exploiting the main ideas

With the main ideas, the dominant cost to update a factor matrix is mainly proportional to **the size I** of dimension and **the number B** of blocks $\Rightarrow$ **fast** and **memory-efficient**

Outline

- Introduction
- Proposed Method
- **Experiments**
- Conclusion

Experimental Questions

- **Answer the following questions:**
  - **Q1. Performance trade-off.** Does Zoom-Tucker provide the best trade-off between query time and reconstruction error?
  - **Q2. Space cost.** What is the space cost of Zoom-Tucker and competitors for preprocessed results?
  - **Q3. Effects of the block size b.** How does a block size b affect query time and reconstruction error of Zoom-Tucker?
  - **Q4. Discovery.** What pattern does Zoom-Tucker discover in different time ranges?

Real-world Datasets

- Use 6 real-world datasets

Dataset	Dimensionality	Length $l_{[t_s,t_e]}$ of Time Range	Summary
Boats[1] [37]	$320 \times 240 \times 7000$	(128, 2048)	Video
Walking Video [25]	$1080 \times 1980 \times 2400$	(128, 2048)	Video
Stock[3]	$3028 \times 54 \times 3050$	(128, 2048)	Time series
Traffic[4] [30]	$1084 \times 96 \times 2000$	(64, 1024)	Traffic volume
FMA[5] [8]	$7994 \times 1025 \times 700$	(32, 512)	Music
Absorb[6]	$192 \times 288 \times 30 \times 1200$	(64, 1024)	Climate

- The last dimension is the time dimension
- Length of time range
  - For each dataset, we use two kinds of time range queries: narrow and wide time ranges

Experimental Setting

- Machine
  - A workstation with a single CPU (Intel Xeon E5-2630 v4 @ 2.2GHz), and 512GB memory
- Target Rank
  - 10
- Block size b
  - 50
- Reconstruction error

$$\frac{\left\|\mathcal{X} - \hat{\mathcal{X}}\right\|_F^2}{\|\mathcal{X}\|_F^2}$$

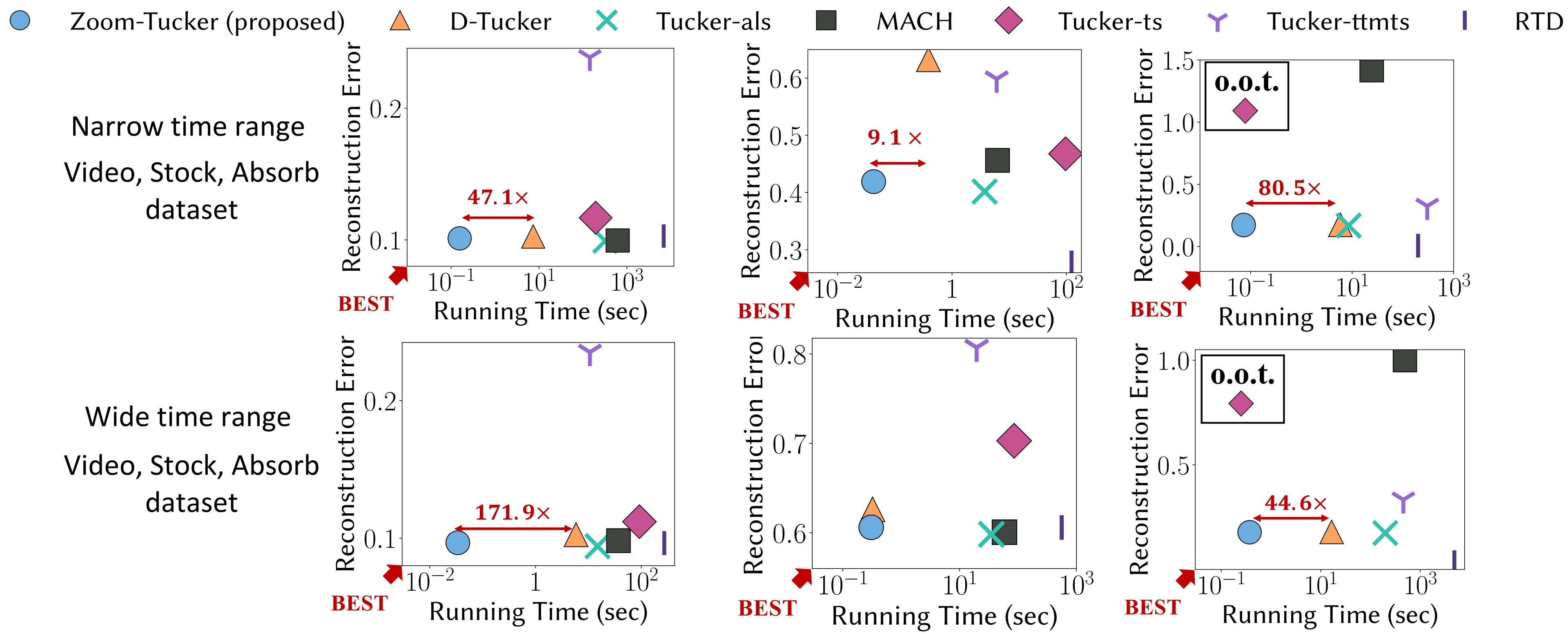
  - $\mathcal{X}$ is an input tensor
  - $\hat{\mathcal{X}}$ is the tensor reconstructed from factor matrices and core tensor

Competitor

- Compare Zoom-Tucker with the following Tucker decomposition approaches
  - **D-Tucker**
    - A SVD based Tucker decomposition method
  - **Tucker-ts and Tucker-ttmts**
    - A sketching based Tucker decomposition method
  - **MACH**
    - A sampling based Tucker decomposition method
  - **Tucker-ALS**
    - An implementation of ALS algorithm in Tensor Toolbox
  - **RTD**
    - A Tucker decomposition method with a randomized algorithm

Q1. Performance Trade-off

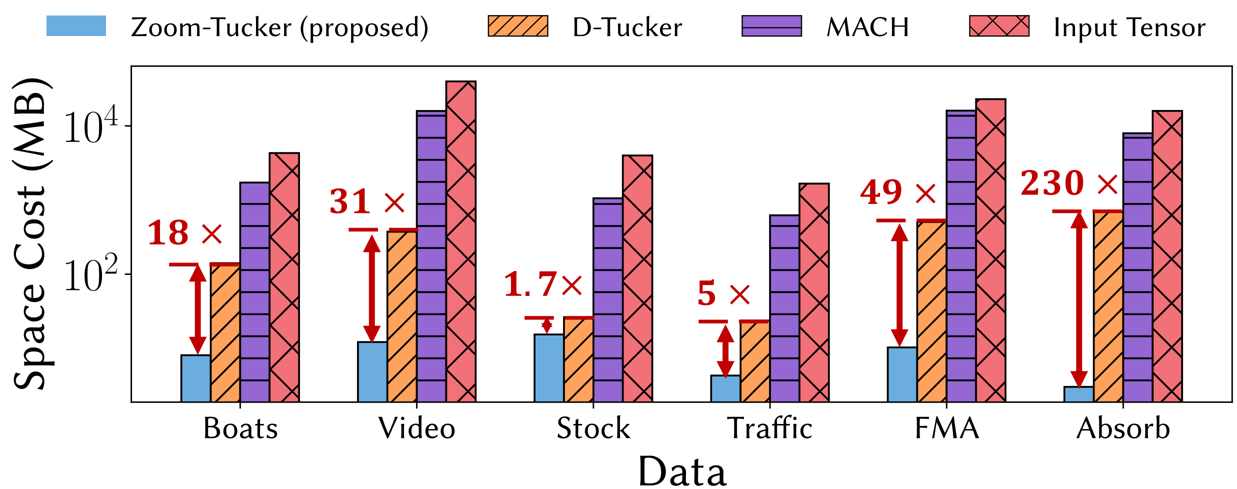
- The lower-left region indicates better performance



Zoom-Tucker **outperforms** the competitors based on Tucker-ALS for both narrow and wide time ranges while having comparable errors

Q2. Space Cost

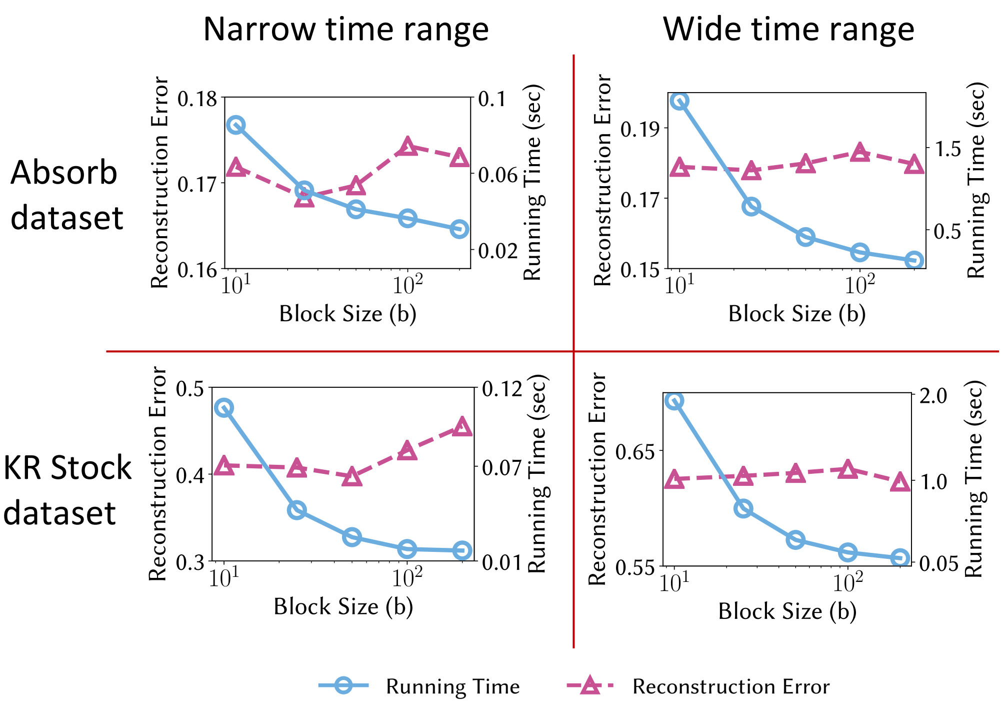
- Compare space cost for storing preprocessed results



  - *Input Tensor* corresponds to the space cost of Tucker-ALS, Tuckerts, Tucker-ttmts, and RTD
- Zoom-Tucker requires up to **230× less space** than competitors

Q3. Effect of block size

- Measure the running time and error with respect to block size b
- **Narrow time range queries**
  - **Trade-off relationships**
    - $b \uparrow$
      - Running time $\downarrow$
      - Error $\uparrow$
- **Wide time range queries**
  - The running time is **inversely proportional** to b
  - The reconstruction error is **not sensitive** to b



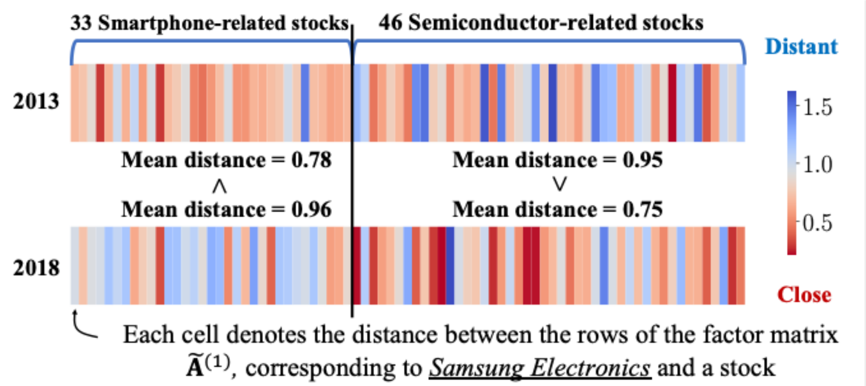
Q4. Discovery

- Given a Korean stock dataset in the form of (stock, features, time), analyze the trend change by comparing results of two time range queries
  - Analyze the change of yearly trend of *Samsung Electronics* in the years **2013 (Query 1)** and **2018 (Query 2)**

1. Perform Zoom-Tucker for the two time range queries and get the factor matrix $\tilde{\mathbf{A}}^{(1)}$ each of whose rows contain the latent features of a stock
2. Manually pick 33 smartphone-related stocks and 46 semiconductor-related stocks
3. For each query, compare the cosine distance between the latent feature vectors of each stock and *Samsung Electronics*

Q4. Discovery

- Analyze trend change by comparing results of the two time range queries



- A clear change of the distances between 2013 and 2018
  - Samsung Electronics is more close to smartphone-related stocks in 2013
  - but to semiconductor-related stocks in 2018
- This result exactly reflects the sales trend of Samsung Electronics
  - In 2013, the annual sales of its smartphone division ↑
  - In 2018, those of its semiconductor division ↑

Zoom-Tucker enables us to efficiently explore diverse time ranges

Outline

- Introduction
- Proposed Method
- Experiments
- **Conclusion**

Conclusion

- Zoom-Tucker answers diverse time range queries on a dense temporal tensor **quickly** and **memory-efficiently**
  - Compress a given temporal tensor block by block along the time dimension
  - Perform Tucker decomposition by elaborately using compression results, every time a time range is given
- Zoom-Tucker **outperforms** the previous Tucker decomposition methods based on ALS
- Zoom-Tucker provides opportunities to extract **unknown and interesting patterns** in diverse time ranges

Thank you !

https://datalab.snu.ac.kr/zoomtucker