frontiers | Frontiers in Big Data

TYPE Original Research
PUBLISHED 23 November 2022
DOI 10.3389/fdata.2022.929511

OPEN ACCESS

EDITED BY
Shandian Zhe,
The University of Utah, United States

REVIEWED BY
Shikai Fang,
The University of Utah, United States
Syed Abbas Z. Naqvi,
University of Engineering and Technology, Lahore, Pakistan

*CORRESPONDENCE
Ravdeep S. Pasricha
rpasr001@ucr.edu

SPECIALTY SECTION
This article was submitted to Machine Learning and Artificial Intelligence, a section of the journal Frontiers in Big Data

RECEIVED 26 April 2022
ACCEPTED 05 September 2022
PUBLISHED 23 November 2022

CITATION
Pasricha RS, Gujral E and Papalexakis EE (2022) Adaptive granularity in tensors: A quest for interpretable structure. *Front. Big Data* 5:929511. doi: 10.3389/fdata.2022.929511

Adaptive granularity in tensors: A quest for interpretable structure

Ravdeep S. Pasricha*, Ekta Gujral and Evangelos E. Papalexakis

Department of Computer Science and Engineering, University of California, Riverside, Riverside, CA, United States

Data collected at very frequent intervals is usually extremely sparse and has no structure that is exploitable by modern tensor decomposition algorithms. Thus, the utility of such tensors is low, in terms of the amount of interpretable and exploitable structure that one can extract from them. In this paper, we introduce the problem of finding a tensor of adaptive aggregated granularity that can be decomposed to reveal meaningful latent concepts (structures) from datasets that, in their original form, are not amenable to tensor analysis. Such datasets fall under the broad category of sparse point processes that evolve over space and/or time. To the best of our knowledge, this is the first work that explores adaptive granularity aggregation in tensors. Furthermore, we formally define the problem and discuss different definitions of "good structure" that are in practice and show that the optimal solution is of prohibitive combinatorial complexity. Subsequently, we propose an efficient and effective greedy algorithm called IceBreaker, which follows a number of intuitive decision criteria that locally maximize the "goodness of structure," resulting in high-quality tensors. We evaluate our method on synthetic, semi-synthetic, and real datasets. In all the cases, our proposed method constructs tensors that have a very high structure quality.

KEYWORDS

tensor, unsupervised learning, temporal granularity, tensor decomposition, multi-aspect data

1. Introduction

In the age of big data, applications deal with data collected at very fine-grained time intervals. In many real-world applications, the data collected spans a long duration and can be extremely sparse. For instance, a time-evolving social network that records interactions of users every second results in a very sparse adjacency matrix per second if observed at that granularity. Similarly, in spatio-temporal data, if one considers GPS data over time, discretizing GPS coordinates based on the observed granularity can lead to very sparse data which may not contain any visible and useful structure. How can we find meaningful and actionable structures in these types of data? Plenty of such datasets are multi-aspect in nature and hence can be modeled using tensors. For instance, a three-mode tensor can represent a time-evolving graph capturing user-user interactions over a period of time, measuring crime incidents in a city community area over a period of time (Smith et al., 2017),

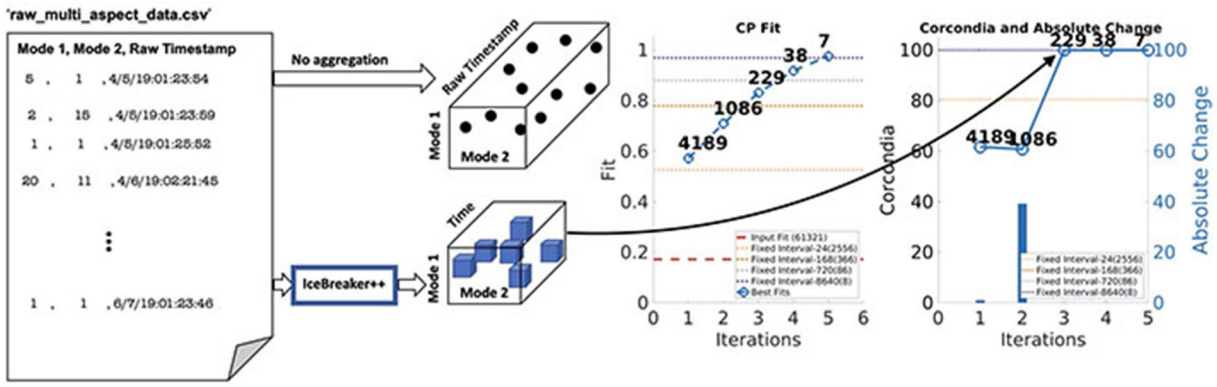

FIGURE 1
Starting from raw CSV files, IceBreaker++ discovers a tensor that has a good structure (under various measures of quality, including interpretability and predictive quality), outperforming traditional fixed aggregation heuristics. Furthermore, IceBreaker++ using various notions of locally optimal structure discovers different resolutions in the data.

or measuring traffic patterns (Zheng et al., 2014). Tensor decomposition has been used in order to extract hidden patterns from such multi-aspect data (Kolda and Bader, 2009; Papalexakis et al., 2016; Sidiropoulos et al., 2017). However, the degree of sparsity in the tensor, which is a function of the granularity in which the tensor is formed, significantly affects the ability of the decomposition to discover a "meaningful" structure in the data.

Consider a dataset that can be modeled as a three-mode tensor, where the third mode is temporal as shown in Figure 1. If the granularity of the temporal mode is too fine (in milliseconds or seconds), one might end up with a tensor that is extremely long on the time mode and where each instance of time has a very small number of entries. This results in an extremely sparse tensor, which typically is of very high rank, and usually has no underlying exploitable structure for widely popular and successful tensor decomposition algorithms (Kolda and Bader, 2009; Papalexakis et al., 2016; Sidiropoulos et al., 2017). However, as we aggregate data points over time, the exploitable structure starts to appear (where-by "exploitable" means the kind of low-rank structure that a tensor decomposition can successfully model and extract). In this paper, we set out to identify what is the best such data-driven aggregation of a tensor which leads to better, exploitable, and interpretable structure, and how this fares against the traditional alternative of selecting a fixed interval for aggregation.

As far as tackling the problem above, there is a considerable amount of work that focuses on a special case, that of aggregating edges of a time evolving graph into "mature" adjacency matrices based on certain graph properties (Sun et al., 2007; Sulo et al., 2010; Soundarajan et al., 2016). In our work, however, we address the problem in more general terms, where the underlying data can be any point process that is observed over time and/or space, and where the aggregation/discretization of the corresponding dimensions directly affects our ability to extract interpretable patterns *via* tensor decomposition. Effectively, as shown in Figure 1, we work toward automating the data aggregation starting from raw data into a well-structured tensor. This paper is based on the preliminary work which has appeared in arxiv (earlier version arXiv:1912.09009v1; Pasricha et al., 2019) and non-archival workshop (Pasricha et al., 2020).

Our contributions to this work are as follows:

- **Novel problem formulation**: We formally define the problem of optimally aggregating a tensor, which is formed from raw sparse data in their original level of aggregation, into a tensor with exploitable and interpretable structure. We further show that solving this problem optimally is computationally intractable. To the best of our knowledge, this paper is the first to tackle this problem in its general form, and we view our formulation as the first step toward automating the process of creating well-behaved tensor datasets.
- **Practical algorithm**: We propose a practical, efficient, and effective algorithm that is able to produce high-quality tensors from raw data without incurring the combinatorial cost of the optimal solution. Our proposed method follows a greedy approach, where at each step, we decide whether different "slices" of the tensor are aggregated based on a variety of intuitive functions that characterize the "goodness of structure" locally.
- **Experimental evaluation**: We extensively evaluate our proposed method on synthetic, semi-synthetic, and real data where we use popular heuristic measures of structure goodness to measure success. Furthermore, we conduct a data mining case study on a large real dataset of crime over time in Chicago, where we

identify interpretable hidden patterns in multiple time resolutions.

We make our implementation publicly available[1] in order to encourage reproducibility of our results.

2. Problem formulation

2.1. Tensor definition and notations

Tensors are multi-dimensional extensions of matrices, and tensor decompositions are a class of methods that extract latent structure from tensor datasets by extending techniques such as principal component analysis and singular value decomposition. The different "dimensions" of a tensor are usually referred to as "modes." In this paper, we focus on the CANDECOMP/PARAFAC (henceforth referred to as CP for brevity) decomposition (Carroll and Chang, 1970; Harshman, 1970), which is the "rank decomposition" of a tensor, i.e., the decomposition of an arbitrary tensor into a sum of R rank-one tensors. Mathematically, for a three-mode tensor $\underline{\mathbf{X}}$, the CP decomposition is $\underline{\mathbf{X}} \approx \sum_{r=1}^{R} \mathbf{A}(:,r) \circ \mathbf{B}(:,r) \circ \mathbf{C}(:,r)$, where $\circ$ is the generalized outer product. Matrices $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$ are called "factor matrices," and each column corresponds to a latent pattern, directly relating an entity of the corresponding mode to a value that can be roughly construed as a soft clustering coefficient (Papalexakis et al., 2012). CP has arguably been the most popular tensor decomposition model in applications where the interest is to extract interpretable patterns for exploratory analysis, and thus, we adopt this decomposition model as our standard in this work. In the interest of space, we refer the reader to a number of available surveys (Kolda and Bader, 2009; Papalexakis et al., 2016; Sidiropoulos et al., 2017). We denote tensors as $\underline{\mathbf{X}}$ and matrices as $\mathbf{X}$, and we adopt Matlab-like notation for indexing.

2.2. Tensor decomposition quality

Unsupervised tensor decomposition, albeit very popular, poses a significant challenge: how can we state whether a computed decomposition is of "high quality," and how can we go about defining "quality" in a meaningful way? Unfortunately, this happens to be a very hard problem to solve (Papalexakis, 2016) and defining a new measure of quality is beyond the scope of this paper. However, there has been a significant amount of work in that direction, which basically boils down to (1) model-based measures, where the quality is measured by how well a given decomposition represents the intrinsic hidden structure of the data and (2) extrinsic measures, where the quality is measured by how well the computed decomposition factors perform in a predictive task. However, extrinsic measures do not generalize, as they specialize to a particular labeled task, and in general, we cannot assume that labels will be available for the data at hand. Thus, in this study, we focus on model-based measures, which can provide a general solution.

In model-based measures, the most straightforward one is the fit, i.e., how well does the decomposition approximate the data under the chosen loss function, in a *low rank*. Low rank is key because the number of components (rank) has to be as small and compact as possible in order to lend itself to human evaluation and exploratory analysis. However, the fit is unstable and prone to errors especially in real and noisy data, thus the community has collectively turned its attention to more robust measures such as the Core Consistency Diagnostic (CORCONDIA for short) (Bro and Kiers, 2003), which measures how well the computed factors obey the CP model.

Both types of quality measure capture different elements of what an end-user would deem good in a set of decomposition factors. In this paper, we are going to use such popular measures of quality in order to characterize the quality of a given tensor dataset $\underline{\mathbf{X}}$. In order to do so, we assume that we have a function $\mathcal{Q}(\underline{\mathbf{X}})$, which optimizes the quality measure $q()$ for a given tensor over all possible decomposition ranks R[2], i.e.,

$$\mathcal{Q}(\underline{\mathbf{X}}) = \max_{R} q(\underline{\mathbf{X}}, \mathbf{A}, \mathbf{B}, \mathbf{C}) \quad (1)$$

where $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$ are the R-column factor matrices for $\underline{\mathbf{X}}$. Finally, a useful operation is the n-mode product, where a matrix $\mathbf{W}$ is multiplied by the n-th mode of a tensor (predicated on matching dimensions in the n-th mode of the tensor and the rows of the matrix), denoted as $\underline{\mathbf{X}} \times_n \mathbf{W}$. For instance, in an $I \times J \times K$ tensor where $n = 3$ and $\mathbf{W}$ of size $K \times K^*$, the product $\underline{\mathbf{X}} \times_n \mathbf{W}$ multiplies all third mode slices of $\underline{\mathbf{X}}$ with $\mathbf{W}$ and results in an $I \times J \times K^*$ tensor.

2.3. The *Trapped Under Ice* problem

To give the reader an intuition of the problem, consider an example of a time-evolving graph that captures social activity over the span of some time. This example can be modeled as a three-mode tensor $\underline{\mathbf{X}}$ of dimensions $I \times J \times K$ where "sender" and "receiver" are the first two modes, "time" being the third mode, and non-zero entry in the tensor represents communications between users at a particular time. If the time granularity is extremely fine-grained (milliseconds or seconds),

1 https://github.com/ravdeep003/adaptive-granularity-tensors

2 In practice, this is done over a small number of low ranks, since low-rank structure is desirable.

there might be only a handful of interactions/edges between nodes at a particular timestamp, resulting in an extremely sparse adjacency matrix for that timestamp, which, in turn, results in an extremely sparse tensor overall and to have a high tensor-rank as a result. In that case, $\underline{\mathbf{X}}$ might not have any interpretable low-rank structure that can be exploited by CP. In this example, we assume that the third mode (time mode) is too fine-grained but in reality, any mode (one or more) can be extremely fine-grained. For example, in spatio-temporal data, where the first two modes are latitude and longitude and the third mode is time, all three modes can suffer from the same problem.

Given tensor $\underline{\mathbf{X}}$ which is created using the "raw" granularities, how does one find a tensor (say $\underline{\mathbf{Y}}$) which has a better exploitable structure and hence can be decomposed into a meaningful latent structure. This is informally the *Trapped Under Ice* problem that we define here (which draws an analogy between the good structure that may exist within the data as being trapped under the ice and not visible by mere inspection). *Trapped Under Ice* has an inherent assumption that the mode in which we aggregate is ordered (e.g., representing time or space), thus permuting the third mode will lead to a different instance of the problem.

More formally we define our problem as follows:

Given a tensor $\underline{\mathbf{X}}$ of dimensions $I \times J \times K$ Find:
A tensor $\underline{\mathbf{Y}}$ of dimensions $I \times J \times K^*$ with $K^* \leq K$ such that

$$\max_{\mathbf{W}} \mathcal{Q}\left(\underline{\mathbf{X}} \times_3 \mathbf{W}\right)$$

where $\mathcal{Q}$ is a measure of goodness and $\mathbf{W}(i,j) = 1$ if slice i in tensor $\underline{\mathbf{X}}$ is aggregated into slice j in the resulting tensor, otherwise $\mathbf{W}(i,j) = 0$.

At first glance, *Trapped Under Ice* might look like a problem amenable to dynamic programming, since it exhibits the optimal substructure property. However, it lacks the overlapping subproblems property, which is across the set of different $\mathbf{W}$ matrices (e.g., two different matrices may have overlapping subproblems) but not within any single $\mathbf{W}$. Thus, we still have to iterate over 2^{K-1} $\mathbf{W}$'s, refer Section 2.4 for more details.

Structure of W: The matrix $\mathbf{W}$ has a special structure. For example, consider a three-mode tensor $\underline{\mathbf{X}}$ of dimensions $10 \times 10 \times 10$, with the third mode being the time mode. Suppose that the optimal level of aggregation for $\underline{\mathbf{Y}}$ is $K^* = 3$.

In this case, $\mathbf{W}$ is of size 3×10 and an example of such a matrix is

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

This $\mathbf{W}$ aggregates the first three slices of $\underline{\mathbf{X}}$ to form the first slice of $\underline{\mathbf{Y}}$, then next three to form the second slice, and last four to form the third slice. No two $\mathbf{W}$ matrices will produce the same aggregation. They can have the same K^* but the order of aggregation of slices will be different.

2.4. Solving *Trapped Under Ice* optimally is hard

Solving *Trapped Under Ice* optimally poses a number of hurdles. First and foremost, the hardness of the problem depends on the definition of the function $\mathcal{Q}$, and most reasonable and intuitive definitions are very hard to optimize since they are non-differentiable, non-continuous, and not concave. So far, in the literature, to the best of our knowledge, there are only heuristics for this quality function. Even so, those heuristic functions can only be evaluated on a single already fully-aggregated tensor, not a partially aggregated version thereof. Thus, *Trapped Under Ice* can only be solved optimally *via* enumerating all admissible solutions and choosing the best. In order to conduct this enumeration, we need to calculate the cardinality of the set of all $\mathbf{W}$ for a given instance of the problem.

Lemma 1. *For an instance of a problem with K initial slices, the cardinality of the set of all* $\mathbf{W}$ *is* 2^{K-1}

Proof. To get K^* aggregated slices, there are $\binom{K-1}{K^*-1}$ ways to choose each of them leading to a different $\mathbf{W}$. There are a number of ways that $K-1$ partition slots can be filled, partitioned by K^*-1 blocks. In order to get the final number, we need to sum up over all potential K^*:

$$\sum_{K^*=0}^{K-1} \binom{K-1}{K^*} = 2^{K-1}$$

□

The direct corollary of the above lemma is that solving optimally *Trapped Under Ice* requires calling the function $\mathcal{Q}$ $O\left(2^K\right)$ times, which is computationally intractable. There may be a small room for improvement by exploiting special structure in the set of all $\mathbf{W}$, however, given discontinuities in our objective function $\mathcal{Q}$, this is not a feasible alternative either. In this paper, we define proxy quality functions $\mathcal{Q}$ that lend themselves to partial evaluation on a partially aggregated solution, thus allowing for efficient algorithms Thus, in the next section, we propose a greedy approach that locally optimizes different criteria quality.

3. Proposed methods

In this section, we propose our efficient and effective greedy algorithm called IceBreaker which takes a tensor $\underline{\mathbf{X}}$ as an input, which has been created directly from raw data, and has no exploitable structure and returns a tensor $\underline{\mathbf{Y}}$, which maximizes the interpretable and exploitable structure. The basic idea behind IceBreaker is to make a linear pass on the mode for which the granularity is suboptimal and using a number of intuitive and locally optimal criteria for the goodness of structure (henceforth referred to as *utility functions*), we greedily decide whether a particular slice across that mode needs to be aggregated[3] into an existing slice or contains good-enough structure to stand on its own. IceBreaker can choose from a number of intuitive utility functions which are based on different definitions of good quality in matrices.

3.1. The IceBreaker algorithm

Algorithm 1 gives a high-level overview of IceBreaker. More specifically, the algorithm takes a three-mode tensor $\underline{\mathbf{X}}$ of dimension $I \times J \times K$ as an input and loops over all the K slices of tensor $\underline{\mathbf{X}}$. Two slices next to each other get aggregated into a single slice if a certain utility function *has stabilized*, i.e., if aggregating the two slices does not offer any additional utility (larger than a particular threshold), then the second slice should not be aggregated with the first and should mark the beginning of a new slice.

Consider a three-mode tensor $\underline{\mathbf{X}}$ with time as the third mode of dimension $I \times J \times K$ is run through IceBreaker with a particular utility function. Our algorithm iterates over the time mode (K slices) and aggregates slices as decided by the utility function. IceBreaker is agnostic to the utility function used. Let us consider a slice that has been aggregated into a single slice from indices i to $j-1$ called the previous slice and another aggregated slice from indices i to j called a candidate slice. Both previous and candidate slices are passed to the utility function separately to obtain a value each called previous and current values, respectively. These values are compared (line 5 in Algorithm 1) to decide whether j^{th} slice is absorbed(line 6 in Algorithm 1) into previous slice or previous slice has stabilized and entry is added in W to indicate which indices of tensor $\underline{\mathbf{X}}$ are aggregated together(line 8 – 9 in Algorithm 1). Now *jth* slice becomes the previous slice and aggregated slice of j and $j+1$ become the candidate slice, the whole process is repeated until all the slices are exhausted.

```
Input: Tensor X of dimension I × J × K
Output: Tensor Y of dimension I × J × K1 and matrix W
   of size K1 × K
1: i = 1; j = 2
2: previousValue = UtilityFunction(X(:, :, i))
3: while j ≤ K do
4:   currentValue = UtilityFunction(sum(X(:, :, i : j), 3)
5:   if previousValue ≶ currentValue then
6:     j = j+1 {Aggregate Slice}
7:   else
8:     {Create a New Slice}
       Add a row in W with value as 1 for indices i
       to j − 1.
       {Update indices for next candidate slice}
9:     i = j; j = j + 1;
10:     previousValue = UtilityFunction(X(:,:,i));
11:   end if
12: end while
13: Y = X ×3 W
14: return Y and W
```

Algorithm 1. IceBreaker.

Note that IceBreaker's complexity is *linear* in terms of the slices K of the original tensor, and its overall complexity depends on the specific utility function used (which is called $O(K)$ times).

3.1.1. Utility functions

In this subsection, we summarize a number of intuitive utility functions that we are using in this paper. This list is by no means exhaustive and can be augmented by different functions (or function combinations) that capture different elements of what is good structure and can be informed by domain-specific insights.

1. **Norm:** We use multiple norm types to find the adaptive granularity of a tensor. For a given threshold, if the rate of change of norm between the previous and candidate slice is less than the threshold, the candidate slice is not selected. Our assumption, in this case, is that no significant amount of information is being added to the previous slice and is considered to have been stabilized. Matrix $\mathbf{W}$ is updated accordingly with indices of the previous slice (aggregated slices in the previous slice). Otherwise, the candidate slice is selected and the process continues until all the slices are exhausted. Different norms demonstrated in this work are Frobenius, 2-norm, and Infinity norm.

3 For the purposes of our work, we use matrix addition as aggregation of slice but this might not be the case and would depend on the problem domain. Other aggregation functions that can be used are OR, min, max, depending on the application domain (e.g., binary data).

Input: Tensor $\underline{\mathbf{Y}}$ of dimension $I \times J \times K$
Output: One Tensor for each iteration
while $K \leq 1$ **do**
  for all Uitlity Functions **do**
    $[\underline{\mathbf{Z}}, \mathbf{W}]$ = IceBreaker $(\underline{\mathbf{Y}})$
  end for
  Select $\underline{\mathbf{Z}}$ with the best Realtive fit
  {Third mode dimension}
  $K_1 = size(\underline{\mathbf{X}}, 3)$
  if $K_1 == K$ **then**
    break;
  else
    $K = K1$
    $\underline{\mathbf{Y}} = \underline{\mathbf{Z}}$
  end if
end while
return one Tensor for each iteration

Algorithm 2. IceBreaker++.

2. **Matrix rank:** In the case of matrix rank, we focus on the 95% reconstruction rank, which is typically much lower than the full rank of the data, but captures the essence of the number of components within the slice. In this case, we consider the previous slice to be stabilized if the matrix-rank of the previous slice decreases by the addition of a new slice, no more slices are added and an entry in matrix **W** is added. We keep aggregating slices if the matrix-rank of the slice is increasing or remains constant.
3. **Missing value prediction:** If a piece of data has a good structure, when we hide a small random subset of the data, the remaining data can successfully reconstruct the hidden values, under a particular model that we have chosen. To this end, we employ a variant of matrix factorization-based collaborative filtering (Koren, 2009) as a utility function to see how good is the aggregated matrix in predicting a certain percent of missing values. This utility function takes the percent of missing values as a parameter and hides those percent of non zeros values in the matrix. Our implementation of matrix factorization with Stochastic Gradient Descent tries to minimize the loss function: $\min_{\mathbf{U},\mathbf{V}} \sum_{i,j \in \Omega} \mathcal{RMSE}\left(\mathbf{A_{ij}} - \mathbf{U_{i,:}} \cdot \mathbf{V_{:,j}}\right)$ where **A** is a given slice, **U** and **V** are factor matrices for a given rank (typically chosen using the same criterion as the matrix rank above), and Ω is the set of *observed* (i.e., non-missing) values. In order to create a balanced problem, since we are dealing with very sparse slices, we conduct *negative sampling* where we randomly sample as many zero entries as there are non-zeros in the slice, and this ends up being the Ω set of observed values.

3.2. The IceBreaker++ algorithm

IceBreaker algorithm returns a tensor $\underline{\mathbf{Y}}$ as an output that is considered to have an exploitable and better structure than the input tensor $\underline{\mathbf{X}}$. The idea behind IceBreaker++ is to recursively feed the output back to IceBreaker until the third mode is reduced to a single slice (matrix) or the dimension of the third mode does not change. IceBreaker algorithm returns a tensor associated with each utility function. Hence, if we used five utility functions, we would get 5 tensors associated with each of them. Now we select the tensor with the highest CP Fit (see Section 4.1), use that as input for IceBreaker, and we repeat this process until the stopping condition is met. The output of each iteration is a candidate tensor. In the end, we have multiple tensors (one for each iteration) which have different temporal resolutions, which can help us get a tensor with the optimal resolution based on the evaluation measures used. Algorithm 2 describes the process discussed in this section.

4. Experimental evaluation

In this section, we present a thorough evaluation of IceBreaker++ using variety of data, including synthetic, semi-synthetic, and real data. We empirically evaluate our analysis using a number of criteria described in detail below. We implement our method in Matlab using the tensor toolbox library (Bader et al., 2015).

4.1. Evaluation measures

When formulating the problem, we neither specify a quality function $\mathcal{Q}$ to be maximized nor did we use such a function in our proposed method. The reason for that is that we reserve the use of different quality functions as a form of evaluation. In particular, we use the two following notions of quality:

- **CP Fit:** To evaluate the effectiveness of our method, we compute the CP fit of the computed tensor for a particular rank with respect to the input tensor.

$$Relative\ Fit = 1 - \left(\frac{||\underline{\mathbf{X}}_{Input} - \underline{\mathbf{X}}_{computed}||_F}{||\underline{\mathbf{X}}_{Input}||_F}\right) \quad (2)$$

- **CORCONDIA**: We employ AutoTen (Papalexakis, 2016), which essentially searches for the maximal number of components which attains a high CORCONDIA (Bro and Kiers, 2003) score, within a user-defined search space. AutoTen returns that number of components (i.e., the low

rank) and the corresponding CORCONDIA score, which we use as our quality metric.

We should note at this point that the two quality measures above are far from continuous and monotonic functions, thus we do not expect that our method progressing the quality will monotonically increase. Thus, we calculate the quality for the final solution of IceBreaker++, and we reserve investigating whether monotonic and well-behaved quality functions exist for future work.

In our experiments, we used five utility functions (see Section 3.1.1) namely Frobenius norm, 2-norm, Infinity norm, Matrix Rank, and Missing Value Prediction. In the case of synthetic datasets, we ran all the utility functions once except for Missing Value Prediction which we ran 10 times. In case of both semi-synthetic and real datasets, in the interest of computational efficiency, we ran all the utility functions once.

4.2. Baseline methods

A naive way to find tensor $\underline{\mathbf{Y}}$ can be by aggregating time mode based on some fixed intervals. If time granularity was in milliseconds, then combining one thousand slices to form slices of seconds granularity reduces the third dimension of tensor $\underline{\mathbf{X}}$ from K to $K/1,000$. This can be applied incrementally from seconds to minutes and so on to find a tensor that has some exploitable structure. We compare the resulting tensor $\underline{\mathbf{Y}}$ determined by IceBreaker against tensors constructed with fixed aggregations. For fixed aggregation, we aggregate the temporal with a window size of 10, 100, and 1,000 for synthetic data. For semi-synthetic and real datasets, we use appropriate time windows accordingly.

4.3. Performance for synthetic data

4.3.1. Creating synthetic data

In order to create a synthetic dataset, we follow a two-step process:

1. We create a random sparse tensor of specific sparsity.
2. Subsequently, we randomly distribute (drawn from a uniform distribution) non zero entries in each slice over some fixed number of slice as explained in below example.

Example: Consider a three-mode tensor $\underline{\mathbf{X}}$ of dimension $I \times J \times K$, for purpose of this example, consider $K = 4$ as shown in Figure 2. Now for each slice of size $I \times J$, distribute randomly (drawn from Uniform distribution) all the non-zeros entries across W slices preserving the I and J indices, creating a tensor of the size $I \times J \times W$. Now append all the tensors in the same order as they appeared in the original tensor, we get a resulting tensor of size $I \times J \times 4W$, which is used as an input for IceBreaker. Thus, if the original tensor is of size $I \times J \times K$ and bucket size W, the resulting tensor is of the size $I \times J \times KW$ approximately[4]. Table 1 shows the synthetic data used for experiments.

4.3.2. Results for synthetic data

In order to evaluate the performance of IceBreaker++, we measure CORCONDIA and fit it on 10 synthetic datasets for both types of datasets as mentioned in Table 1. In interest of conserving space, we only show one set of results for both synthetic datasets. The leftmost part of Figures 3, 4 show the best fit at end of each iteration. The number on top of the dots represents the dimension of the third mode after each iteration. The dotted line in the plot shows the fit of the input tensor and fixed intervals tensor[5]. The rightmost part of Figures 3, 4 show the CORCONDIA computed at the end of each iteration and the absolute change of CORCONDIA. Absolute change of CORCONDIA is computed as shown below:

$$abs(corcondia(j+1) - corcondia(j))$$

The dotted line in the plot represents CORCONDIA value for the fixed intervals tensor. When there is a sudden drop in the value of CORCONDIA, we consider the iteration before as a suitable candidate for tensor analysis. In the case of SD1 that would be iteration number 2 and the resulting tensor of size $100 \times 100 \times 8$. In the case of SD2 that would also be iteration number 2 and the resulting tensor of size $100 \times 100 \times 57$.

4.4. Performance for semi-synthetic data

4.4.1. Creating semi-synthetic data

In this study, we used the Enron dataset (Priebe et al., 2006; Bader et al., 2007), which is a dataset of the number of email exchanges between employees spread over 44 months. Each month is represented by a matrix. To create the semi-synthetic data, we use step 2 as described in the generation of synthetic case. We take the non-zero elements and randomly distribute non zero entries in each slice over some fixed number of the slice. For this dataset, we converted the monthly data into weekly, daily, and hourly data. Non-zero entries in each slice were distributed over four different candidate slices for creating the weekly dataset (roughly approximating 4 weeks as a month). In the case of daily, each slice of monthly data was distributed over 30 different slices as mentioned in Table 2 and finally in

4 The number of slice can be less than KW, since slice for each non-zero value is selected randomly, there can be a case where a slice is not selected.

5 The number in the parenthesis represents the dimension of the third mode for that tensor.

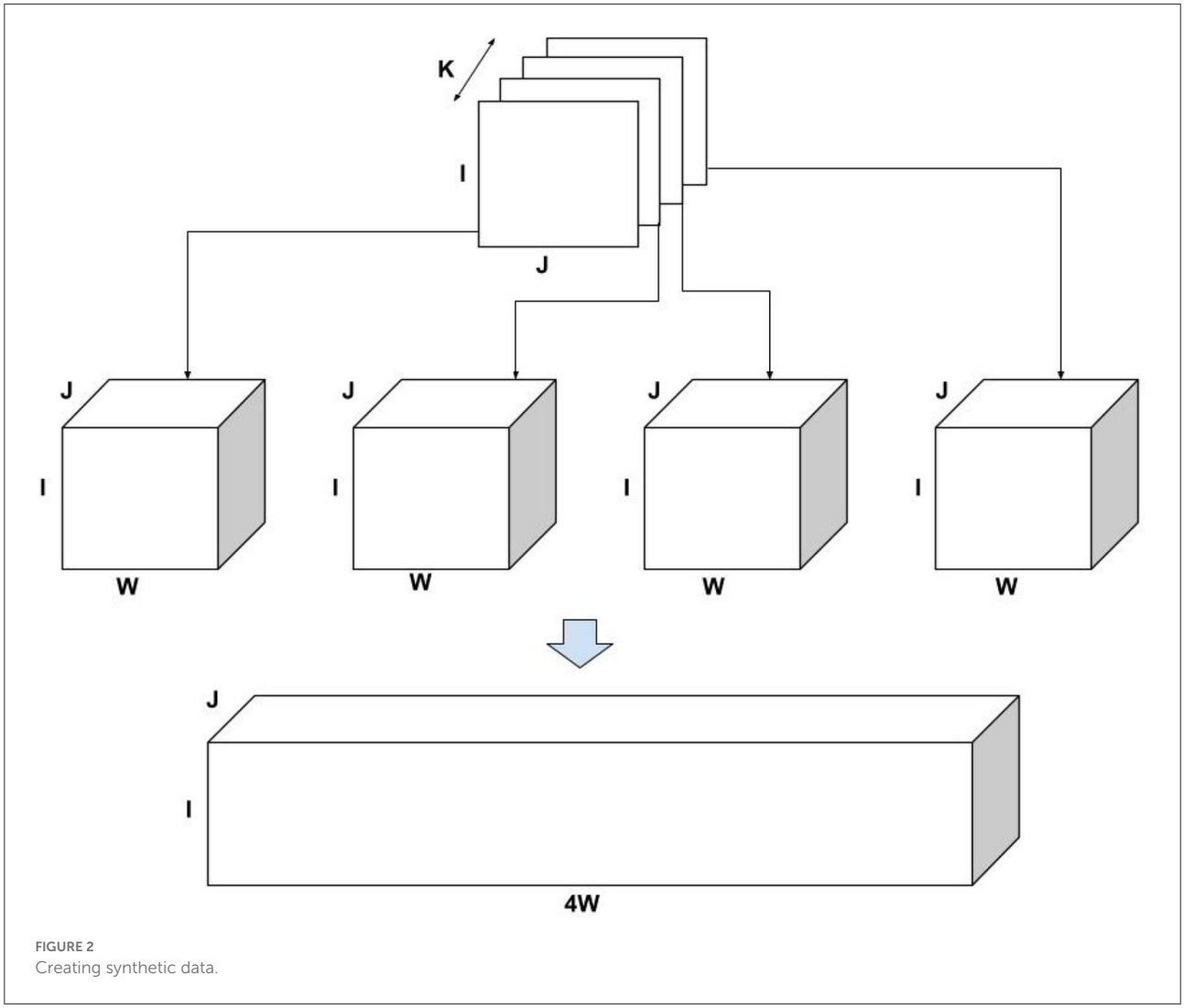

FIGURE 2
Creating synthetic data.

TABLE 1 Table of synthetic datasets analyzed.

Dataset	Original dimension	Window size (W)	Approximate final dimension	Number of datasets
SD1	$100 \times 100 \times 10$	50	$100 \times 100 \times 500$	10
SD2	$100 \times 100 \times 100$	50	$100 \times 100 \times 5,000$	10

the case of hourly, each non zero entry in the monthly slice was distributed over 720 slices (24×30).

4.4.2. Results for semi-synthetic data

The leftmost parts of Figures 5–7 show the fit of different iterations and the rightmost part of the Figures 5–7 show the CORCONDIA computed at different iterations. In the case of Enron Weekly, we see a sudden drop in CORCONDIA after iteration 1 as shown in Figure 5 and the corresponding tensor is of size $184 \times 184 \times 17$. In the case of Enron Daily, we do not see a significant change in CORCONDIA values in two iterations and corresponding tensors are of size $184 \times 184 \times 78$ and $184 \times 184 \times 5$ giving us tensors of different granularity.

In the case of Enron Hourly, we see a drop in CORCONDIA after iterations 1 and 2 as shown in Figure 7. In this case, practitioner can make choice between a tensor of resolution $184 \times 184 \times 469$ or $184 \times 184 \times 34$ depending on what evaluation metric they value more, fit, CORCONDIA, or both. Tensor after iteration 2 ($184 \times 184 \times 34$) seems to have a good score for both

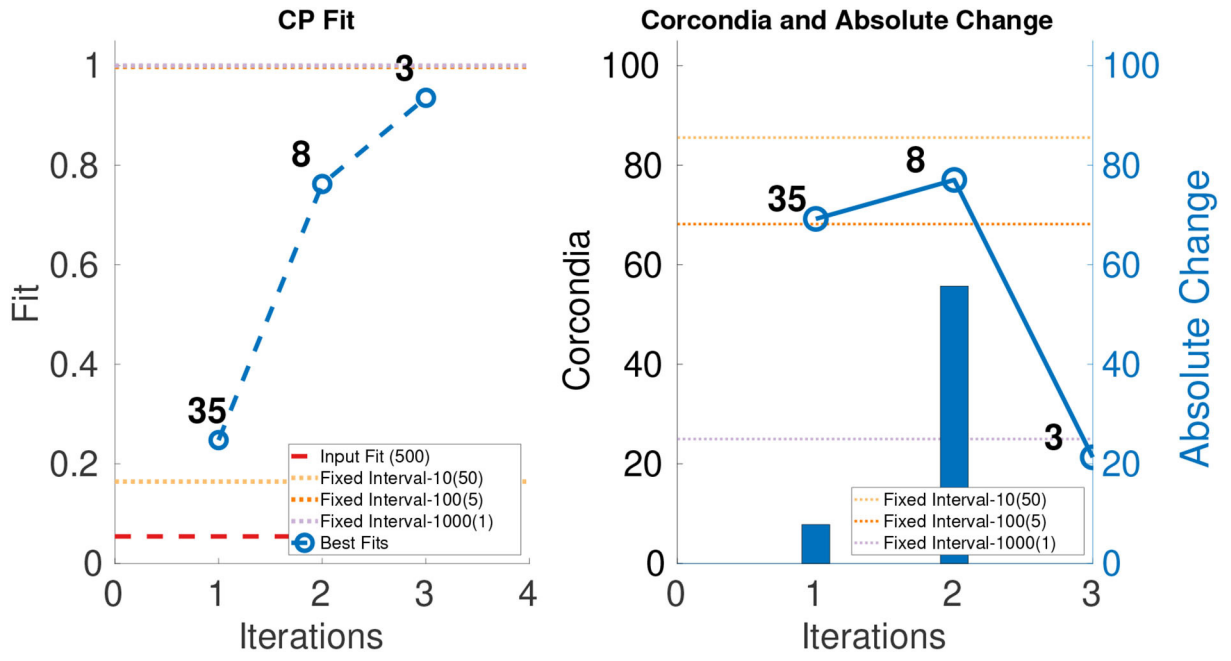

FIGURE 3
CANDECOMP/PARAFAC (CP) fit and Core Consistency Diagnostic (CORCONDIA) of best fit tensor and its absolute change at each iteration for SD1.

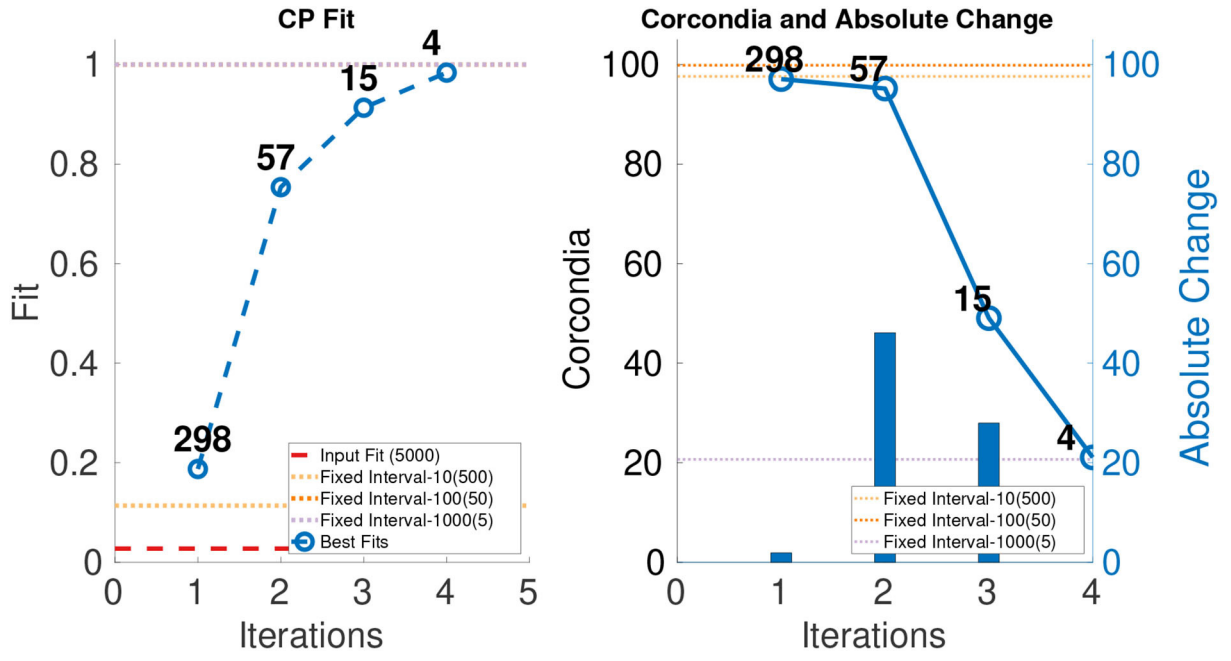

FIGURE 4
CP fit and CORCONDIA of best fit tensor and its absolute change at each iteration for SD2.

TABLE 2 Table of semi-synthetic datasets analyzed.

Dataset	Original dimension	Window size (W)	Approximate final dimension
Enron weekly	$184 \times 184 \times 44$	4	$184 \times 184 \times 176$
Enron daily	$184 \times 184 \times 44$	30	$184 \times 184 \times 1,320$
Enron hourly	$184 \times 184 \times 44$	720	$184 \times 184 \times 31,680$

fit and CORCONDIA whereas the Tensor after iteration 1 has a good CORCONDIA score but not a good CP fit.

4.5. Data mining case study

4.5.1. Chicago crime dataset

For our case study, we use a dataset provided by the city of Chicago[6] that records different types of crime committed in different areas of the city over a period of time (Smith et al., 2017). The tensor we create has modes (area, crime, and timestamp), where "community area" and "crime" are discretized by the city of Chicago, and "timestamp" is the coarsely aggregated (hourly) timestamp. The dates that we focused was on a span of 7 years, from 13 December 2010 to 11 December 2017.

We ran IceBreaker++ on this dataset which is of size $77 \times 32 \times 61,321$, and in the right most part of Figure 8, we show its CORCONDIA for each iteration and we observe that iterations 3, 4, and 5 have high values of CORCONDIA, which would suggest they offer a resolution with an exploitable structure. Iterations 1 and 2 also have decent CORCONDIA values. Given these two ranges of CORCONDIA values, we decided to drill down and look into the actual tensor components that can be extracted from those different tensors. In the interest of space, we took the tensor returned by iteration 2 as $\underline{\mathbf{X}}_1$, the tensor $\underline{\mathbf{X}}_2$ and tensor $\underline{\mathbf{X}}_3$ are returned by iterations 3 and 4, respectively. Tensor $\underline{\mathbf{X}}_1$ contains three high-quality components, whereas $\underline{\mathbf{X}}_2$ and $\underline{\mathbf{X}}_3$ contain two.

Figures 9–11 shows sets of patterns[7] for $\underline{\mathbf{X}}_1$, $\underline{\mathbf{X}}_2$, and $\underline{\mathbf{X}}_3$, respectively: interestingly, factor 1 of $\underline{\mathbf{X}}_1$ and factor 1 of $\underline{\mathbf{X}}_2$ pertain to similar spatial and criminal pattern. As shown in Figures 10, 11, we observed that both factors of tensors $\underline{\mathbf{X}}_2$ and $\underline{\mathbf{X}}_3$ pertain to similar spatial and criminal patterns. In summary, tensors $\underline{\mathbf{X}}_1$, $\underline{\mathbf{X}}_2$, and $\underline{\mathbf{X}}_3$ capture similar interpretable patterns over different temporal resolutions.

4.5.2. Comparison against fixed aggregation

A natural question is whether the results are qualitatively "better" than the ones by a fixed aggregation. Answering this question heavily depends on the application at hand, however, here we attempt to quantify this in the following way: intuitively, a good set of components offers more *diversity* in much of the data it covers. For instance, a practitioner would prefer a set of results for the Chicago crime dataset where the components span most of the regions of the city and uncover diverse patterns of crime, over a set of components that seem to uncover a particular type of crime. Even though there may be a number of confounding factors, aggregating on a regular time interval may be very good in capturing periodic activity (in this example, crime that exhibits normal periodicity happens to coincide with the aggregation resolution we have chosen), whereas aggregating adaptively may help discover the structure that is more erratic and more surprising. In order to capture this and test this hypothesis, we compute the coverage of entities for the first and second mode of the tensor (i.e., areas of Chicago and crime types in this example) in all the discovered components: for each component, we measure the top-k entities, and through that, we compute the empirical probability distribution of all entities in the results. A more preferable set of results will have higher coverage, resulting in distribution with higher entropy. In Table 3, we show the entropy for both modes 1 and 2 for IceBreaker++ and for the different fixed aggregations (averaged over 10 different runs), where IceBreaker++ overall offers more diverse patterns in both space and criminal activity.

5. Related work

To the best of our knowledge, this is the first attempt at formalizing and solving this problem, especially as it pertains to the tensor and multi-aspect data mining domain. Nevertheless, there has been significant amount of work on temporal aggregations in graphs (Sun et al., 2007; Sulo et al., 2010; Soundarajan et al., 2016) and in finding communities in temporal graphs (Gorovits et al., 2018). In the graph literature, the closest work to ours is Soundarajan et al. (2016), in which the authors look at aggregating stream of temporal edges to produce a sequence of structurally mature graphs based on a variety of network properties.

In the tensor literature, Almutairi et al. (2021) solved the inverse of this problem, where the goal is to disaggregate a tensor. Concurrently to our work, Kwon et al. (2021)

6 https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2

7 We omit plotting the temporal mode since we lack external information that we can potentially correlate it with, however, an analyst with such side information can find the different time resolutions of $\underline{\mathbf{X}}_1$, $\underline{\mathbf{X}}_2$, and $\underline{\mathbf{X}}_3$ useful.

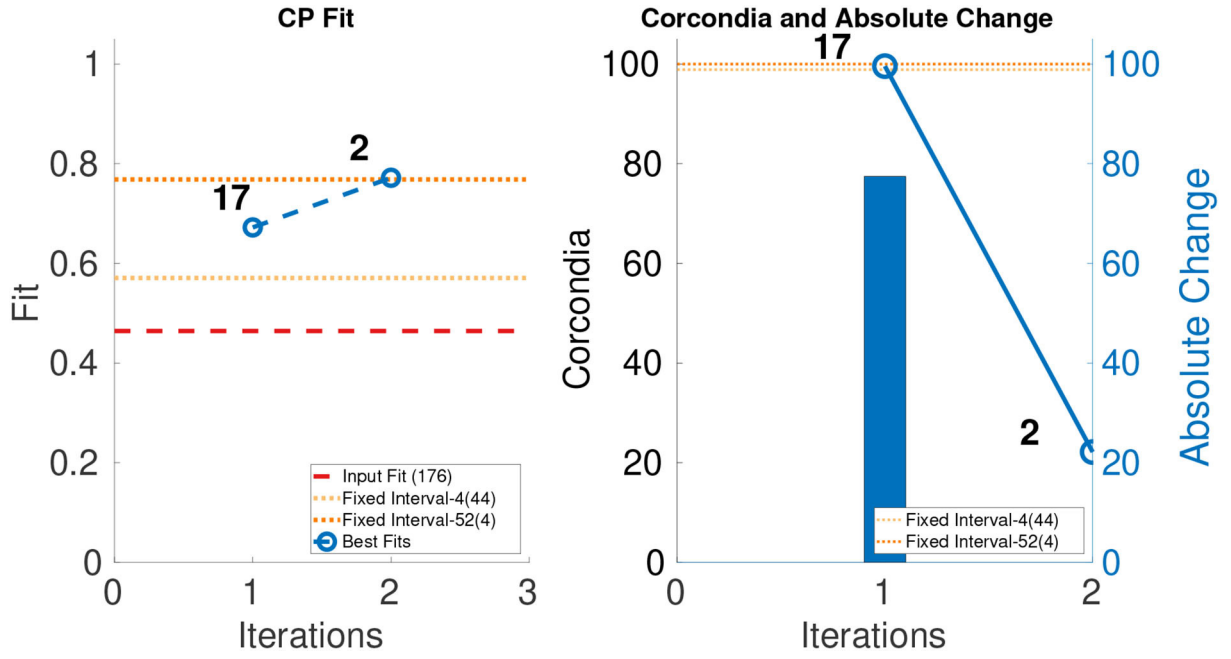

FIGURE 5
CP fit and CORCONDIA of best fit tensor and its absolute change at each iteration for Enron Weekly.

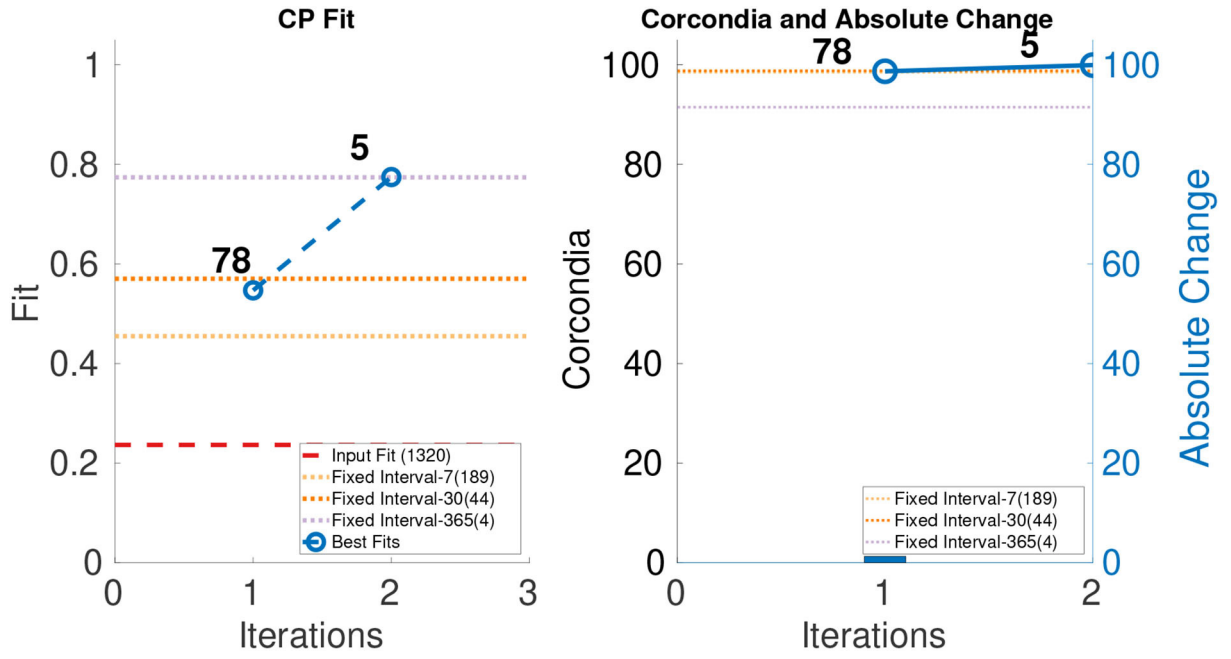

FIGURE 6
CP fit and CORCONDIA of best fit tensor and its absolute change at each iteration for Enron Daily.

developed a streaming CP decomposition that works on the original granularity of the data, instead of preprocessing the tensor in order to identify one or more optimal aggregations. We reserve a full investigation of connections between our problem formulation and Kwon et al. (2021)'s study for future work.

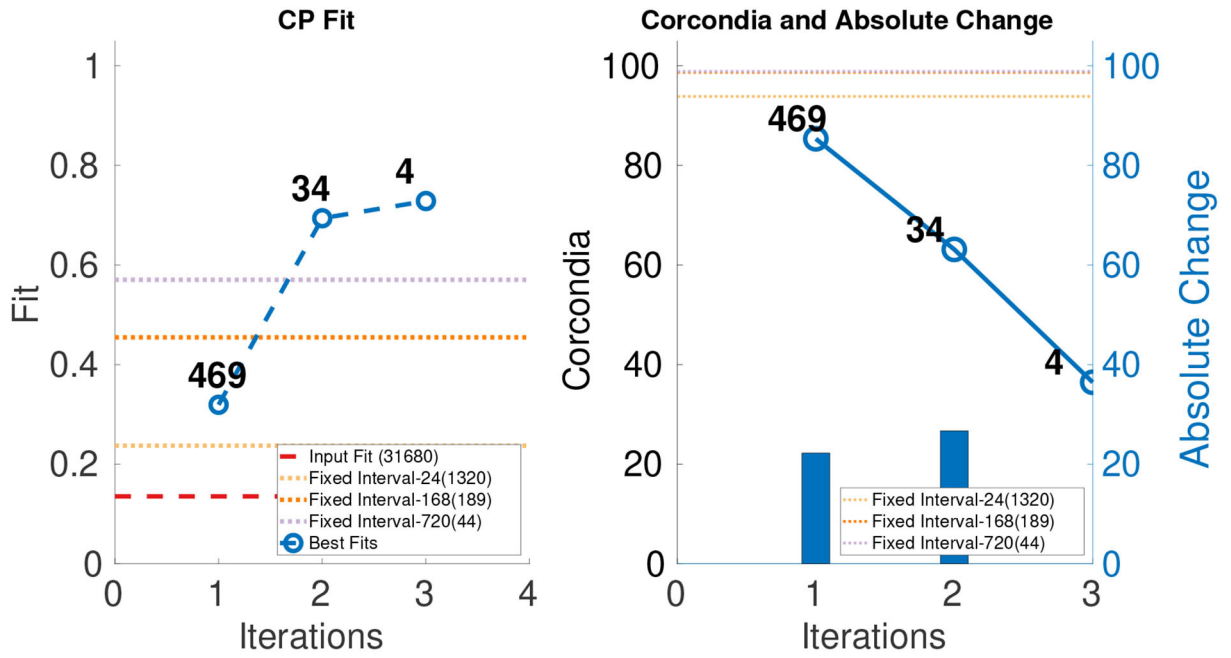

FIGURE 7
CP fit and CORCONDIA of best fit tensor and its absolute change at each iteration for Enron Hourly.

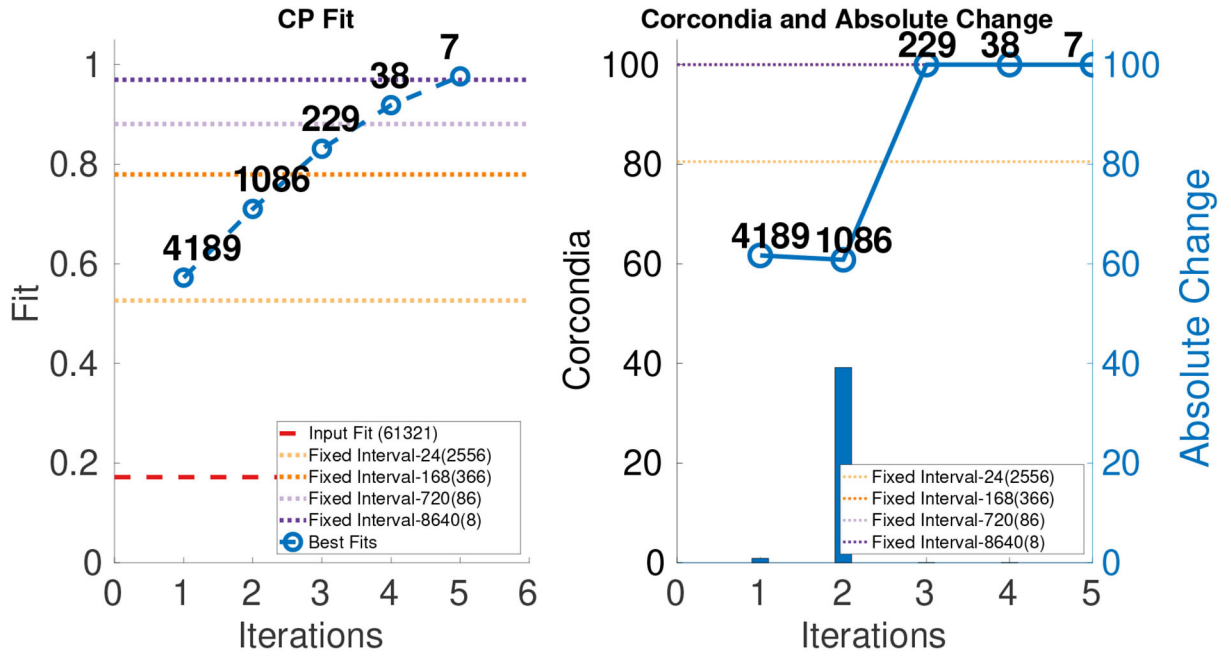

FIGURE 8
CP fit and CORCONDIA of best fit tensor and its absolute change at each iteration for Chicago Crime Dataset.

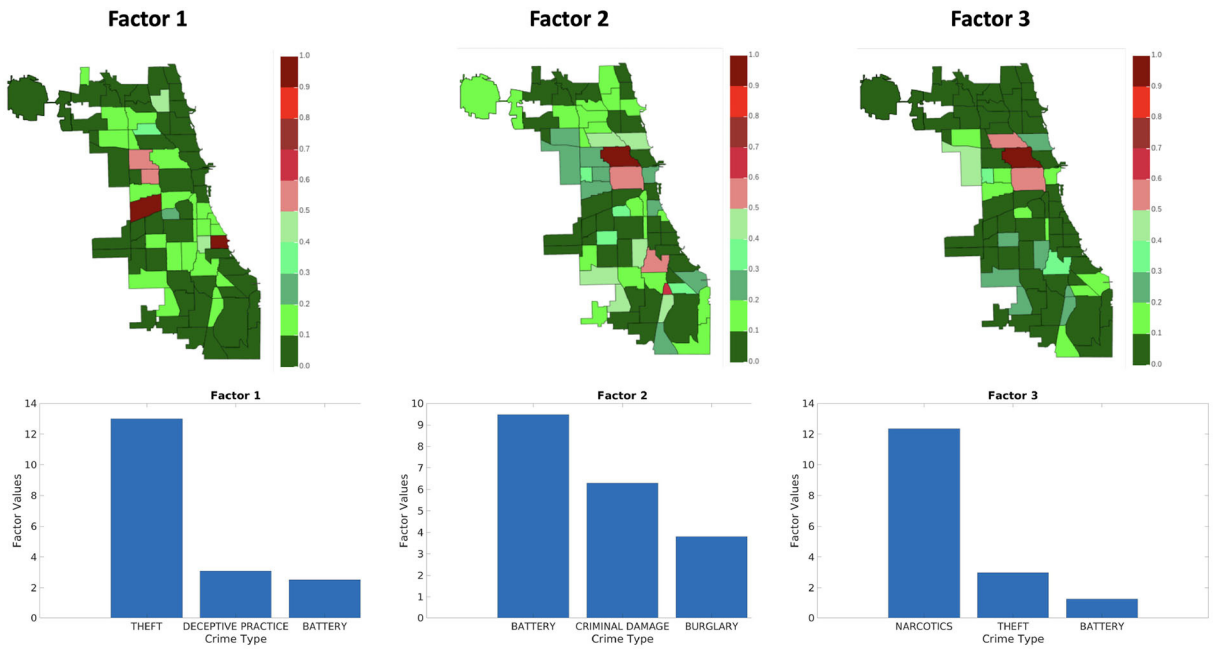

FIGURE 9
Analyzing the Chicago data from iteration-2 ($\underline{X}_1$). Chicago heatmap value ranges from 0.0 to 1.0.

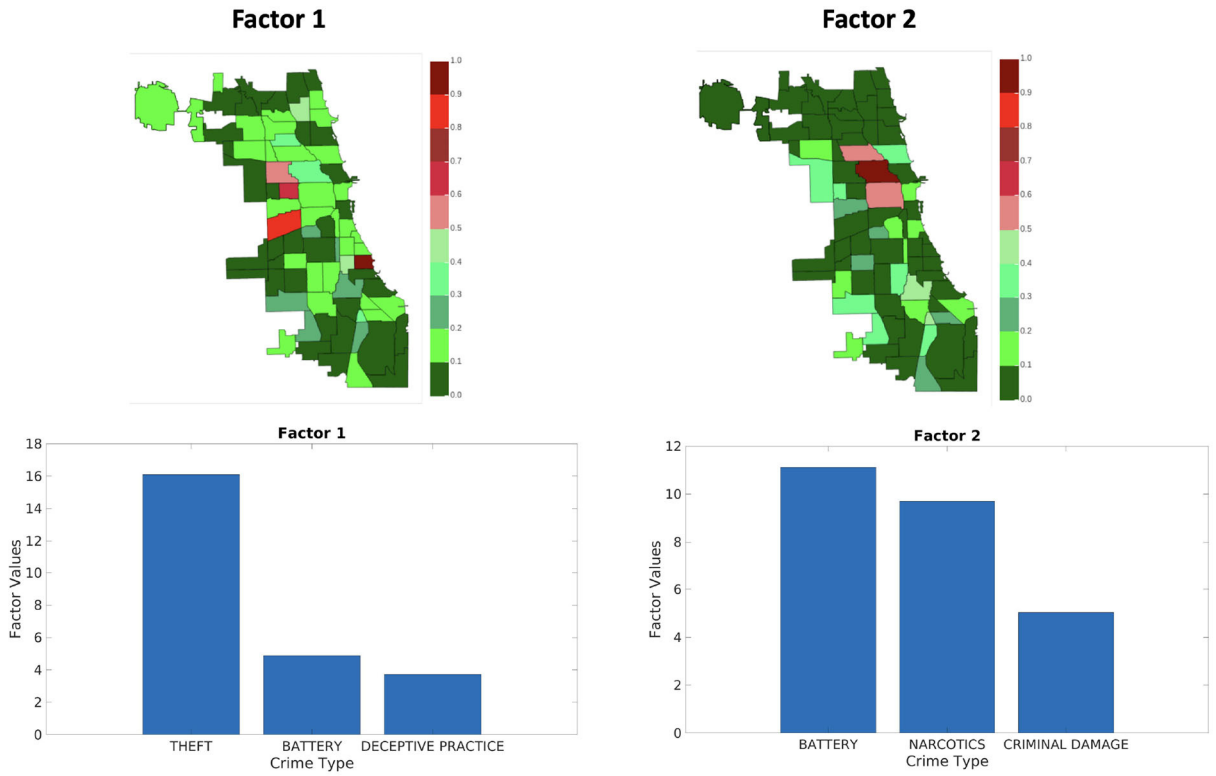

FIGURE 10
Analyzing the Chicago data from iteration-3 ($\underline{X}_2$). Chicago heatmap value ranges from 0.0 to 1.0.

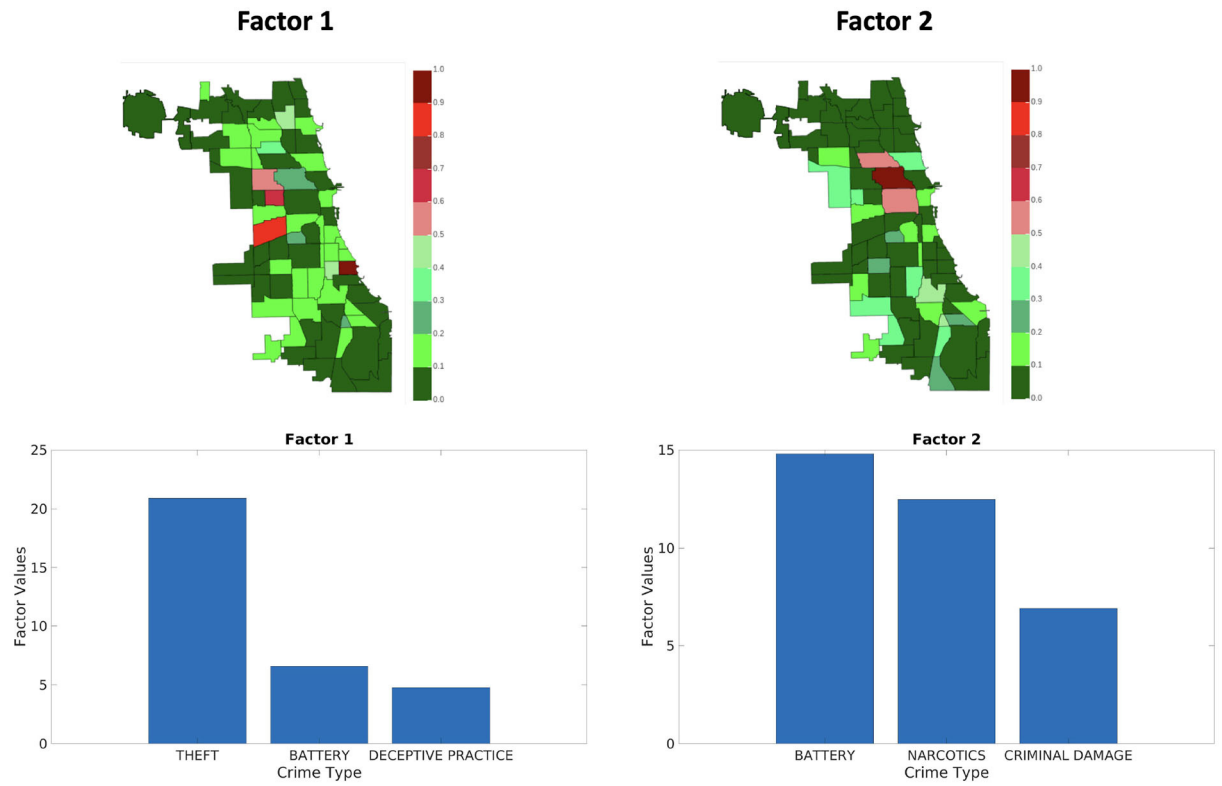

FIGURE 11
Analyzing the Chicago data from iteration-4 ($\underline{X}_3$). Chicago heatmap value ranges from 0.0 to 1.0.

TABLE 3 The entropy of top-3 components in factors for the area and crime type.

	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Fixed Interval-24	Fixed Interval-168	Fixed Interval-720	Fixed Interval-8640
Area	**2.8554**	2.6810	2.5850	2.5850	2.5850	2.7255	2.5850	2.5850	2.5850
Crime	**2.8783**	2.7255	2.2516	2.2516	2.0850	2.4362	2.2516	2.2516	2.1183

Black color is best icebreaker entropy and red color is best fixed interval entropy.

6. Conclusions

In this study, we are, to the best of our knowledge, the first to define and formalize the *Trapped Under Ice* problem in constructing a tensor from raw sparse data. We demonstrate that an optimal solution is intractable and subsequently proposed IceBreaker and IceBreaker++, a practical solution that is able to identify good tensor structure from raw data and construct tensors from the same dataset that pertain to multiple resolutions. Our experiments demonstrate the merit of IceBreaker++ in discovering useful and high-quality structures and providing tools to data analysts in automatically extracting multi-resolution patterns from raw multi-aspect data. In the future, we will work toward extending IceBreaker in cases where more than one mode is *Trapped Under Ice* (naively one can apply IceBreaker to each mode sequentially, but this disregards joint variation across modes) and extend IceBreaker for higher-order tensors.

Data availability statement

All publicly available data and the original contributions presented the original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author/s.

Author contributions

RP: design, implementation, experimental evaluation, and writing. EG: design and writing. EP: design, writing, and primary research advisor of RP and EG. All authors contributed to the article and approved the submitted version.

Funding

This research was supported by the National Science Foundation under CAREER grant no. IIS 2046086 and the Department of the Navy, Naval Engineering Education Consortium under award no. N00174-17-1-0005. It was sponsored by the Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA).

Acknowledgments

This study is based on the preliminary work which has appeared in arxiv (earlier version arXiv:1912.09009v1) (Pasricha et al., 2019) and non archival workshop (Pasricha et al., 2020).

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Author disclaimer

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory or the US Government. The US Government is authorized to reproduce and distribute reprints for government purposes not withstanding any copyright notation here on.

References

Almutairi, F. M., Kanatsoulis, C. I., and Sidiropoulos, N. D. (2021). PREMA: principled tensor data recovery from multiple aggregated views. *IEEE J. Select. Top. Signal Process.* 15, 535–549. doi: 10.1109/JSTSP.2021.3056918

Bader, B. W., Harshman, R. A., and Kolda, T. G. (2007). "Temporal analysis of semantic graphs using ASALSAN," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, Omaha, NE, 33–42. doi: 10.1109/ICDM.2007.54

Bader, B. W., and Kolda, T. G. (2015). *Matlab Tensor Toolbox Version 2.6.*

Bro, R., and Kiers, H. A. (2003). A new efficient method for determining the number of components in Parafac models. *J. Chemometr.* 17, 274–286. doi: 10.1002/cem.801

Carroll, J. D., and Chang, J.-J. (1970). Analysis of individual differences in multidimensional scaling *via* an n-way generalization of "eckart-young decomposition. *Psychometrika* 35, 283–319. doi: 10.1007/BF02310791

Gorovits, A., Gujral, E., Papalexakis, E. E., and Bogdanov, P. (2018). "LARC: learning activity-regularized overlapping communities across time," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (London), 1465–1474. doi: 10.1145/3219819.3220118

Harshman, R. A. (1970). Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA Work. Paper. Phonetic.* 16, 1–84. Available online at: https://www.psychology.uwo.ca/faculty/harshman/wpppfac0.pdf

Kolda, T. G., and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Rev.* 51, 455–500. doi: 10.1137/07070111X

Koren, Y. (2009). "Collaborative filtering with temporal dynamics," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Paris), 447–456. doi: 10.1145/1557019.1557072

Kwon, T., Park, I., Lee, D., and Shin, K. (2021). "Slicenstitch: continuous CP decomposition of sparse tensor streams," in *37th IEEE International Conference on Data Engineering, ICDE 2021* (Chania), 816–827. doi: 10.1109/ICDE51399.2021.00076

Papalexakis, E. E. (2016). "Automatic unsupervised tensor mining with quality assessment," in *Proceedings of the 2016 SIAM International Conference on Data Mining* (Miami, FL), 711–719. doi: 10.1137/1.9781611974348.80

Papalexakis, E. E., Faloutsos, C., and Sidiropoulos, N. D. (2016). Tensors for data mining and data fusion: models, applications, and scalable algorithms. *ACM Trans. Intell. Syst. Technol.* 8, 1–44. doi: 10.1145/2915921

Papalexakis, E. E., Sidiropoulos, N. D., and Bro, R. (2012). From k-means to higher-way co-clustering: multilinear decomposition with sparse latent factors. *IEEE Trans. Signal Process.* 61, 493–506. doi: 10.1109/TSP.2012.2225052

Pasricha, R., Gujral, E., and Papalexakis, E. (2020). "Adaptive granularity in time evolving graphs as tensors," in *16th International Workshop on Mining and Learning with Graphs (MLG)* (San Diego, CA).

Pasricha, R., Gujral, E., and Papalexakis, E. E. (2019). Adaptive granularity in tensors: a quest for interpretable structure. *arXiv preprint arXiv:1912.09009.* doi: 10.48550/arXiv.1912.09009

Priebe, C. E., Conroy, J. M., Marchette, D. J., and Park, Y. P. (2006). *Enron Data Set.* Available online at: http://www.cis.jhu.edu/~parky/Enron/

Sidiropoulos, N. D., De Lathauwer, L., Fu, X., Huang, K., Papalexakis, E. E., and Faloutsos, C. (2017). Tensor decomposition for signal processing and machine learning. *IEEE Trans. Signal Process.* 65, 3551–3582. doi: 10.1109/TSP.2017.2690524

Smith, S., Choi, J. W., Li, J., Vuduc, R., Park, J., Liu, X., et al. (2017). *FROSTT: The Formidable Repository of Open Sparse Tensors and Tools.* Available online at: http://frostt.io/

Soundarajan, S., Tamersoy, A., Khalil, E. B., Eliassi-Rad, T., Chau, D. H., Gallagher, B., et al. (2016). "Generating graph snapshots from streaming edge data," in *Proceedings of the 25th International Conference Companion on World Wide Web* (Québec), 109–110. doi: 10.1145/2872518.2889398

Sulo, R., Berger-Wolf, T., and Grossman, R. (2010). "Meaningful selection of temporal resolution for dynamic networks," in *Proceedings of the Eighth Workshop on Mining and Learning with Graphs* (Washington, DC), 127–136. doi: 10.1145/1830252.1830269

Sun, J., Faloutsos, C., Papadimitriou, S., and Yu, P. S. (2007). "Graphscope: parameter-free mining of large time-evolving graphs," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (California), 687–696. doi: 10.1145/1281192.1281266

Zheng, Y., Capra, L., Wolfson, O., and Yang, H. (2014). Urban computing: concepts, methodologies, and applications. *ACM Trans. Intell. Syst. Technol.* 5, 1–55. doi: 10.1145/2629592