Spatially Compact Dense Block Mining in Spatial Tensors

Weike Tang
tangweike2022@email.szu.edu.cn
Shenzhen University
Shenzhen, China

Dingming Wu*
dingming@szu.edu.cn
Shenzhen University
Shenzhen, China

Tsz Nam Chan
edisonchan@szu.edu.cn
Shenzhen University
Shenzhen, China

Kezhong Lu
kzlu@szu.edu.cn
Shenzhen University
Shenzhen, China

Abstract

Spatial tensors have been extensively used in a wide range of applications, including remote sensing, geospatial information systems, conservation planning, and urban planning. We study the problem of Spatially Compact Dense (SCD) block mining in a spatial tensor, which targets for discovering dense blocks that cover small spatial regions. However, most of existing dense block mining (DBM) algorithms cannot solve the SCD-block mining problem since they only focus on maximizing the density of candidate blocks, so that the discovered blocks are spatially loose, i.e., covering large spatial regions. Therefore, we first formulate the problem of mining *top-k Spatially Compact Dense blocks (SCD-blocks)* in spatial tensors, which ranks SCD-blocks based on a new scoring function that takes both the density value and the spatial coverage into account. Then, we adopt a filter-refinement framework that first generates candidate SCD-blocks with good scores in the filtering phase and then uses the traditional DBM algorithm to further maximize the density values of the candidates in the refinement phase. Due to the NP-hardness of the problem, we develop two types of solutions in the filtering phase, namely the top-down solution and the bottom-up solution, which can find good candidate SCD-blocks by approximately solving the new scoring function. The evaluations on four real datasets verify that compared with the dense blocks returned by existing DBM algorithms, the proposed solutions are able to find SCD-blocks with comparable density values and significantly smaller spatial coverage.

CCS Concepts

• **Information systems → Data mining**; **Data mining**; • **Theory of computation → Design and analysis of algorithms**.

Keywords

Spatially Compactness, Dense Block, Spatial Tensor

ACM Reference Format:
Weike Tang, Dingming Wu, Tsz Nam Chan, and Kezhong Lu. 2025. Spatially Compact Dense Block Mining in Spatial Tensors. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3690624.3709221

1 Introduction

A tensor is a relation with N attributes, a.k.a a multi-dimensional array with N indices. Tensor analysis [1, 6] is an important field in data science and data mining communities. Among most of the tensor analysis tools, dense block mining (DBM), a.k.a. dense subtensor mining, is an important operation for discovering patterns from a dataset, which has been extensively used in different applications. Some representative examples include network intrusion detection [23, 27], retweet boosting detection [15, 16], synchronized behaviour detection [5, 12, 32], and genetics applications [23, 25].

The spatial tensor is one important type of tensor where two of the dimensions are spatial dimensions, i.e., the longitude and the latitude dimensions. The two spatial dimensions of the tuples in the spatial tensor indicate the spatial locations of the tuples. It enables the representation and analysis of complex spatial data, aiding in remote sensing [24], geospatial information systems [21], conservation planning [22], and urban planning [4]. Traditional DBM aims for mining blocks with high density in a tensor. Differently, we study the problem of Spatially Compact Dense (SCD) block mining in a spatial tensor, which targets for discovering dense blocks that cover small spatial regions. SCD-block mining is pivotal in a wide range of applications, such as identifying and analyzing localized abnormalities in medical images [20], compressing and representing large-scale urban planning maps [19], and identifying localized trends or anomalies in high-frequency trading data and providing insights into market behavior [7].

However, most of existing DBM algorithms [28, 29, 32] cannot solve the SCD-block mining problem since they only focus on maximizing the density of candidate blocks, so that the discovered blocks are spatially loose, i.e., covering large spatial regions. As an example, Figure 1c plots the spatial locations of the tuples contained in the top-3 dense blocks found by the state-of-the-art DBM algorithm M-Biz [28] on a check-in dataset Gowalla_NA in North America. It is observed that each of these blocks covers a large spatial region, i.e., almost the whole spatial region of the dataset. In contrast, the SCD-blocks studied in this paper are exemplified in Figures 1a where the three SCD-blocks cover small spatial regions. Each SCD-block represents a group of users visiting a set of nearby

[*] Corresponding author.

KDD '25, August 3–7, 2025, Toronto, ON, Canada

ACM ISBN 979-8-4007-1245-6/25/08
https://doi.org/10.1145/3690624.3709221

places, which indicates that these users may share similar living habits.

To solve the SCD-block mining problem, one may consider two possible options to extend existing DBM algorithms. One option is applying existing DBM algorithms on a subregion of the dataset to obtain SCD-blocks. However, this option fails. As shown in Figure 1d, algorithm M-Biz is applied on a subregion (the region of Texas) of dataset Gowalla_NA. Unlike Figure1b, the returned blocks are still spatially loose, almost covering the entire subregion. The other option is filtering the discovered dense blocks by imposing a spatial constraint. However, this option may return no result when all discovered dense blocks do not satisfy the spatial constraint. As an example in Figure 7 in the experiment, the top-5 densest blocks returned by existing DBM algorithms all cover large spatial regions. If setting the spatial constraint to 0.2, none of the blocks returned by existing DBM algorithms has the spatial coverage less than 0.2. The reason why these two options fail is as follows. Existing DBM algorithms try to remove slices with low mass in order to obtain blocks with high density. However, this removing operation is ineffective in reducing the spatial coverage of blocks, so that the obtained blocks are spatially loose.

Therefore, we ask a question in this paper. *Can we discover dense blocks with compact spatial coverage (shown in Figure 1a) in spatial tensors?* To provide an affirmative answer to this question, we first formulate the problem of mining *top-k Spatially Compact Dense blocks (SCD-blocks)* in spatial tensors. Compared with the traditional dense block mining that only takes the density value as the objective function, the top-k SCD-blocks are ranked based on a new scoring function that takes both the density value and the spatial coverage into account. Then, we adopt a filter-refinement framework that first generates candidate SCD-blocks with good scores in the filtering phase and then uses the traditional DBM algorithm to further maximize the density values of the candidates in the refinement phase.

In the filtering phase, the way of generating candidate SCD-blocks determines the density and the spatial coverage of the final results. In order to discover spatially compact dense blocks, we develop two types of solutions, namely the top-down solution and the bottom-up solution, which can find good candidate SCD-blocks by approximately solving the new scoring function (as computing the exact value for this scoring function is an NP-hard problem [2, 17]). Specifically, the proposed top-down solution introduces a partition operation that divides the spatial tensor into sub-blocks, so that the spatial coverage ratio can be reduced. The challenge of the partition operation is where to divide the spatial tensor to get a sub-block with low spatial coverage ratio and high density value. To tackle this issue, we propose the concave policy. The proposed bottom-up solution locates all tuples of a spatial tensor in a two dimensional spatial space. It first divides the two dimensional spatial space into $h \times h$ cells, and then assigns each tuple to the cell where the tuple is located. Next, candidate SCD-blocks are generated by merging nearby dense cells. The top-down solution not only achieves the same approximation guarantee as traditional DBM algorithms, but also is able to discover SCD-blocks that cannot be found by traditional DBM algorithms. Although the bottom-up solution has no approximation guarantee, it is able to discover fine-grained SCD-blocks.

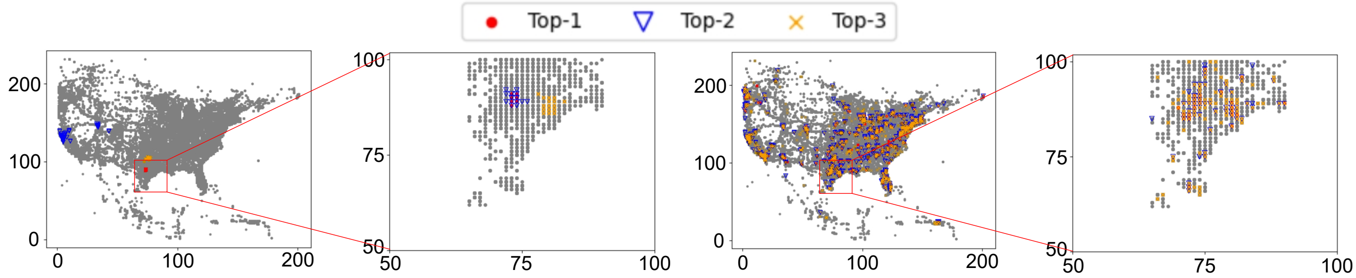
The evaluations on four real datasets verify that the proposed solutions are able to find SCD-blocks, such that compared with the dense blocks returned by existing DBM algorithms, the SCD-blocks have comparable density values and significantly small spatial coverage. Since the top-down solution and the bottom-up solution approach the problem differently, their results have different features. In general, the spatial coverages of the SCD-blocks found by the bottom-up solution are smaller than that found by the top-down solution. In the datasets used in the experiment, the top-down solution discovers country-level SCD-blocks and the bottom-up solution discovers city-level SCD-blocks. The SCD-blocks with different levels of granularity may be useful for various levels of decision making.

The remainder of the paper is organized as follows. Related work is reviewed in Section 2. Section 3 first introduces the problem of mining top-k SCD-blocks and relevant concepts, and then describes the filter-refinement framework that is applied for solving the problem. The top-down solution is proposed in Section 4 and the bottom-up solution is proposed in Section 5. Performance evaluations are presented in Section 6. Conclusions are given in Section 7.

2 Related Work

Decomposition based algorithms. One type of algorithms uses tensor decomposition, such as CP Decomposition (CPD) [18] and HOSVD [18] for dense block mining. For instance, MAF [23] adopts CPD to spot dense blocks in the network traffic logs. STenSr [26] models traffic sensor data as a spatio-temporal tensor stream $X_1, X_2, \cdots, X_t, \cdots$ where X_t is a 2-mode tensor contains vehicle information of 5 minutes. It applies HOSVD to detect abnormal tensors indicating traffic jams in the stream. SamBaTen [13] and OnlineCP [33] conduct the incremental tensor decomposition. D-Tucker [14] tries to reduce the computational cost by slicing the input tensor into matrices. Then, it computes factor matrices and the core tensor based on the randomized SVD results on sliced matrices.

Search based algorithms. Previous studies have shown that the tensor decomposition based algorithms are outperformed by the search based algorithms [15, 27] in terms of efficiency and accuracy. The search based algorithms provide guarantees on the density values of the discovered blocks. We proceed to review state-of-the-art search based algorithms. CrossSpot [16] discovers dense blocks that worth inspecting and sorts them in terms of the importance. M-Zoom [27] removes dimension values one by one in a greedy way until no dimension value is left. Before removing each dimension value, a snapshot of the current block is stored as a candidate. The densest block is the snapshot with the highest density. The density of the found block is guaranteed to be $1/N$ optimal where N is the number of tuples in the tensor. M-Biz [28] is a variant of M-Zoom. It starts from a seed block, adds or removes dimension values in a greedy way until the block reaches a local optimum. D-Cube [29] applies the idea of M-Zoom to disk-resident tensors, allowing to process data that are too large to fit in main memory. Following the idea of M-Zoom, DenseStream [30] incrementally maintains and updates a dense block in a tensor stream in real time as events arrive. It maintains the D-order to search for the densest block. The D-order is updated for every single new tuple. DenseAlert [30]



(a) RPA (Gowalla_NA) (b) RPA (Gowalla_Texas) (c) M-Biz (Gowalla_NA) (d) M-Biz (Gowalla_Texas)

Figure 1: Spatially compact dense blocks vs. traditional dense blocks.

adopts DenseStream to spot the sudden appearances of dense blocks. MUST [9, 10] provides better guarantees on the densities of the dense blocks found by previous studies, such as DenseAlert, M-Zoom, and M-Biz. AugSplicing [32] is a fast streaming algorithm that incrementally splices dense blocks of previous detections and the new blocks detected in new tuples. It outperforms DenseStream in terms of efficiency.

Variants of dense blocks. Besides traditional definition of the dense block, variants of dense blocks have also been investigated. For instance, ISG+D-Spot [3] studies the problem of finding hidden dense blocks that do not have a high-density signal on all dimensions but on a subset of dimensions. It constructs an information sharing graph and finds dense subgraphs. CatchCore [11, 12] aims for detecting hierarchical dense blocks. It formulates the densest block detection problem in an optimization perspective, so that gradient-based methods can be applied.

Summary. All the above studies indifferently treat the spatial and the traditional dimensions in the tensor. None of them studies the spatial characteristics of the discovered dense blocks. We propose an interesting scenario of dense block mining where the spatial region covered by blocks are taken into account. The experiment results show that existing algorithms find large-sized dense blocks that cover large spatial regions, whereas our algorithms can discover moderate- and small-sized dense blocks with compact spatial coverage.

3 Problem Definition and Framework

3.1 Problem Definition

Notations for spatial tensors. The spatial tensor considered in this paper is a relation with $d-2$ traditional dimensions (a.k.a. attributes), denoted by $A_1, A_2, \cdots, A_{d-2}$, two spatial dimensions Υ_x and Υ_y, and a non-negative numerical dimension X, denoted by $\mathcal{R}(A_1, A_2, \cdots, A_{d-2}, \Upsilon_x, \Upsilon_y, X)$. Spatial dimensions Υ_x and Υ_y denote longitude and latitude dimensions, respectively. For each tuple $t \in \mathcal{R}$, we use $t[A_i]$, $t[\Upsilon_x]$, $t[\Upsilon_y]$, and $t[X]$ to denote the values of corresponding dimensions in t. The set of distinct values of a dimension A_i in $\mathcal{R}$ is denoted by $\mathcal{R}(i) = \{t[A_i] \mid t \in \mathcal{R}\}$. Similarly, we have the sets of distinct values of spatial dimensions, denoted by $\mathcal{R}(\Upsilon_x)$ and $\mathcal{R}(\Upsilon_y)$. Tuple t exists in the spatial tensor if and only if $t[X] \neq 0$. Relation $\mathcal{R}$ is an d-way spatial tensor of size $|\mathcal{R}(1)| \times \cdots \times |\mathcal{R}(d-2)| \times |\mathcal{R}(\Upsilon_x)| \times |\mathcal{R}(\Upsilon_y)|$. The number of tuples in a spatial tensor $\mathcal{R}$ is denoted by $n = |\mathcal{R}|$.

Notations for spatially compact dense blocks (SCD-blocks). Let $\mathcal{B}(i)$, $\mathcal{B}(\Upsilon_x)$, and $\mathcal{B}(\Upsilon_y)$ be subsets of $\mathcal{R}(i)$, $\mathcal{R}(\Upsilon_x)$, and $\mathcal{R}(\Upsilon_y)$, respectively. An *SCD-block* $\mathcal{B}$ in $\mathcal{R}$ is defined as $\mathcal{B}(A_1, A_2, \cdots, A_{d-2}, \Upsilon_x, \Upsilon_y, X) = \{t \in \mathcal{R} \mid t[\Upsilon_x] \in \mathcal{B}(\Upsilon_x) \wedge t[\Upsilon_y] \in \mathcal{B}(\Upsilon_y) \wedge t[A_i] \in \mathcal{B}(i), i \in [1, d-2]\}$. Block $\mathcal{B}$ forms a subtensor of $\mathcal{R}$ of size $|\mathcal{B}(1)| \times \cdots \times |\mathcal{B}(d-2)| \times |\mathcal{B}(\Upsilon_x)| \times |\mathcal{B}(\Upsilon_y)|$. For convenience of presentation, when it is clear from the context, we use A_{d-1} and A_d to represent spatial dimensions Υ_x and Υ_y, respectively. A *slice* $\mathcal{B}(a, i)$ of block $\mathcal{B}$ consists of the tuples whose values of dimension A_i equal a, i.e., $\mathcal{B}(a, i) = \{t \mid t \in \mathcal{B} \wedge t[A_i] = a\}, i \in [1, d]$.

Density measures. The *mass* of an SCD-block $\mathcal{B}$ is defined as $M_{\mathcal{B}} = \sum_{t \in \mathcal{B}} t[X]$, i.e., the sum of the numerical dimension X of the tuples in $\mathcal{B}$. The *cardinality sum* of SCD-block $\mathcal{B}$ is defined as $D_{\mathcal{B}} = \sum_{i=1}^{N} |\mathcal{B}(i)|$. We follow previous studies and adopt the widely used density measure: the arithmetic average mass [27], i.e.,

$$\rho(\mathcal{B}) = \frac{M_{\mathcal{B}}}{D_{\mathcal{B}}/N}. \tag{1}$$

Spatial coverage ratio. Let $\Upsilon_x^L(\mathcal{R})$ and $\Upsilon_x^U(\mathcal{R})$ ($\Upsilon_y^L(\mathcal{R})$ and $\Upsilon_y^U(\mathcal{R})$) be the minimum and the maximum values of the longitude (latitude) of the tuples in a spatial tensor $\mathcal{R}$, respectively. The *spatial range* of $\mathcal{R}$ is defined as $\mathrm{SR}(\mathcal{R}) = (\Upsilon_x^U(\mathcal{R}) - \Upsilon_x^L(\mathcal{R})) \cdot (\Upsilon_y^U(\mathcal{R}) - \Upsilon_y^L(\mathcal{R}))$. Similarly, the spatial range of an SCD-block $\mathcal{B}$ is denoted by $\mathrm{SR}(\mathcal{B})$. The spatial ratio of SCD-block $\mathcal{B}$ is defined as:

$$\delta_{\Upsilon}(\mathcal{B}) = \mathrm{SR}(\mathcal{B})/\mathrm{SR}(\mathcal{R}). \tag{2}$$

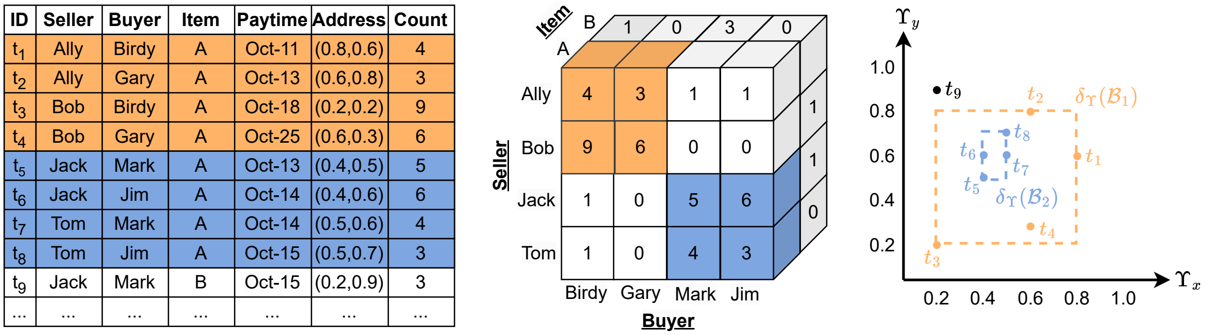
Problem statement: top-k SCD-block mining. (i) **Given:** a spatial tensor $\mathcal{R}$, the number k of blocks, a scoring function $f(\mathcal{B})$. (ii) **Find:** k distinct SCD-blocks in $\mathcal{R}$ with the highest scores in terms of scoring function $f(\mathcal{B})$.

Scoring function $f(\mathcal{B})$ aggregates the density ρ and the spatial coverage ratio δ_{Υ}. We aim for the SCD-blocks with high scores, i.e., the blocks with high density values and low spatial coverage ratios. The following scoring function $f(\mathcal{B})$ is used in the experiment. However, our proposed algorithms can be easily extended to other forms of scoring functions.

$$f(\mathcal{B}) = \alpha \cdot \tilde{\rho}(\mathcal{B}) + (1-\alpha) \cdot \delta_{\Upsilon}^{-1}(\mathcal{B}), \tag{3}$$

where $\tilde{\rho}(\mathcal{B})$ is the normalized density, $\delta_{\Upsilon}^{-1}(\mathcal{B}) = 1 - \delta_{\Upsilon}(\mathcal{B})$, and $\alpha \in (0, 1)$.

*Example 3.1 (**Orders on e-commerce platform**).* Figure 2 shows an example spatial tensor $\mathcal{R}$ with four traditional dimensions *Seller*, *Buyer*, *Item*, and *Paytime*, longitude and latitude dimensions denoted by *Address*, and a non-negative numerical dimension *Count*.



ID	Seller	Buyer	Item	Paytime	Address	Count
t_1	Ally	Birdy	A	Oct-11	(0.8,0.6)	4
t_2	Ally	Gary	A	Oct-13	(0.6,0.8)	3
t_3	Bob	Birdy	A	Oct-18	(0.2,0.2)	9
t_4	Bob	Gary	A	Oct-25	(0.6,0.3)	6
t_5	Jack	Mark	A	Oct-13	(0.4,0.5)	5
t_6	Jack	Jim	A	Oct-14	(0.4,0.6)	6
t_7	Tom	Mark	A	Oct-14	(0.5,0.6)	4
t_8	Tom	Jim	A	Oct-15	(0.5,0.7)	3
t_9	Jack	Mark	B	Oct-15	(0.2,0.9)	3
...	...	...	...	...	...	...

(a) Tensor (b) Blocks (c) Spatial coverage

Figure 2: Example blocks in a tensor of e-commerce platform.

Each tuple in $\mathcal{R}$ is a transaction record: a *Buyer* places *Count* orders of *Item* from a *Seller* at *Paytime* and these orders are shipped to *Address*. In Figure 2, block $\mathcal{B}_1$ consists of orange tuples t_1, t_2, t_3, and t_4. Block $\mathcal{B}_2$ consists of blue tuples t_5, t_6, t_7, and t_8. The density of $\mathcal{B}_1$ is calculated as $\rho(\mathcal{B}_1) = \frac{4+3+9+6}{(2+2+1+4+3+4)/6} = 8.25$ and the density of $\mathcal{B}_2$ is calculated as $\rho(\mathcal{B}_2) = \frac{5+6+4+3}{(2+2+1+3+2+3)/6} \approx 8.31$. After normalization using the sigmoid function, we get $\tilde{\rho}(\mathcal{B}_1) = \frac{1}{1+e^{-8.25}} \approx 0.99$ and $\tilde{\rho}(\mathcal{B}_2) = \frac{1}{1+e^{-8.31}} \approx 0.99$. The spatial coverage ratios of blocks $\mathcal{B}_1$ and $\mathcal{B}_2$ are $\delta_\Upsilon(\mathcal{B}_1) = \frac{SR(\mathcal{B}_1)}{SR(\mathcal{R})} = \frac{(0.8-0.2)^2}{1} = 0.36$ and $\delta_\Upsilon(\mathcal{B}_2) = \frac{SR(\mathcal{B}_2)}{SR(\mathcal{R})} = \frac{(0.7-0.5)\times(0.5-0.4)}{1} = 0.02$, respectively. Then, we have $\delta_\Upsilon^{-1}(\mathcal{B}_1) = 1 - \delta_\Upsilon(\mathcal{B}_1) = 0.64$ and $\delta_\Upsilon^{-1}(\mathcal{B}_2) = 1 - \delta_\Upsilon(\mathcal{B}_2) = 0.98$. According to the scoring function (Equation 3), given $\alpha = 0.5$, we have $f(\mathcal{B}_1) = 0.815$ and $f(\mathcal{B}_2) = 0.985$. Although the density values of blocks $\mathcal{B}_1$ and $\mathcal{B}_2$ are the same, block $\mathcal{B}_2$ is more spatially compact, i.e., having smaller spatial coverage ratio. Thus, block $\mathcal{B}_2$ is ranked higher than block $\mathcal{B}_1$.

NP-hardness. Our top-k SCD-block mining problem is a generalization of the traditional DBM problem. By setting $\alpha = 1$ and $k = 1$, our top-k SCD-block mining problem reduces to a special case, i.e., the traditional DBM problem. Since the traditional DBM problem is an NP-hard problem [2, 17], our problem is also an NP-hard problem.[1] Therefore, there is currently no polynomial algorithm for computing exact result of our problem. We develop the first polynomial-time approximate algorithms for solving the top-k SCD-block mining problem.

3.2 Filter-Refinement Framework

State-of-the-art DBM algorithms [28, 29, 32] equally treat all the dimensions in a spatial tensor and only aim for maximizing the block density, so that they find blocks with large spatial coverage ratios. To obtain blocks with small spatial coverage ratios and high density values, we adopt a filter-refinement framework that separates the processing of the spatial dimensions and the traditional dimensions. It consists of two phases as follows.

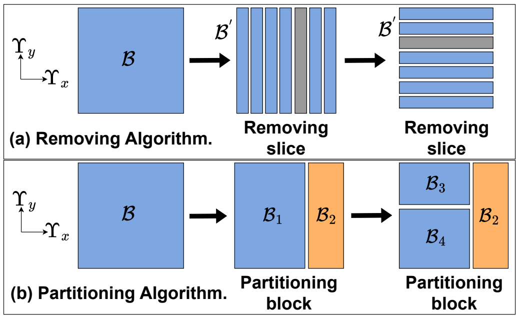
Filtering Phase. It processes the spatial dimensions and generates candidate SCD-blocks with good scores. The way of generating candidate SCD-blocks plays an important role in determining the scores of the result SCD-blocks. We propose two different solutions for the filtering phase, i.e., a top-down solution in Section 4 and a bottom-up solution in Section 5.

Refinement Phase. It applies existing DBM algorithms on traditional dimensions to further maximize the density of the candidate SCD-blocks. After that, the k candidate SCD-blocks are returned as the final results.

4 Top-Down Solution

4.1 Baseline: Removing Algorithm (RA)

We propose a baseline, the removing algorithm (RA), that extends state-of-the-art DBM algorithms [28, 29, 32] to process the spatial dimensions and generate candidate SCD-blocks. Figure 3a shows the working process of the RA algorithm. Given a spatial tensor $\mathcal{R}$, it generates candidate SCD-blocks as follows. Initially, let $\mathcal{B}$ be the input spatial tensor $\mathcal{R}$ and $\mathcal{B}'$ be a copy of $\mathcal{B}$. Among all the slices of the spatial dimensions, it selects a slice based on a slice selection policy and removes it from $\mathcal{B}'$. This step is repeated until the score of $\mathcal{B}'$ cannot be improved. When the iteration stops, block $\mathcal{B}'$ is considered as a candidate SCD-block and is passed to the refinement phase. Next, after candidate $\mathcal{B}'$ is finalized in the refinement phase, it is removed from $\mathcal{B}$. After resetting $\mathcal{B}'$ to a copy of the current $\mathcal{B}$, algorithm RA is applied to generate the next candidate SCD-block from $\mathcal{B}'$. The pseudo code of RA is shown in Algorithm 1 in Section A in the appendix.



(a) Removing Algorithm.

(b) Partitioning Algorithm.

Figure 3: Top-down solution.

However, algorithm RA is not effective in reducing the spatial coverage ratio. The reason is as follows. Consider block $\mathcal{B}'$ in the current iteration. According to Equation 2, the spatial coverage

[1] Given a spatial tensor containing n tuples, the time complexity of brute-force exact algorithm is $C_1^n + C_2^n + \cdots + C_n^n = O(2^n)$.

ratio $\delta_{\Upsilon}(\mathcal{B}')$ is calculated based on spatial boundary values $\Upsilon_x^L(\mathcal{B}')$, $\Upsilon_x^U(\mathcal{B}')$, $\Upsilon_y^L(\mathcal{B}')$, and $\Upsilon_y^U(\mathcal{B}')$. Suppose a slice $\mathcal{B}(a,i)$, $i \in \{\Upsilon_x, \Upsilon_y\}$ is removed. If the dimension value a is not one of the spatial boundary values, the spatial coverage ratio does not decrease. For instance, considering the orange block in Figure 2, if the slice with value 0.3 of the latitude dimension (Υ_y) is removed, the spatial coverage ratio of the orange block does not change. The spatial coverage ratio can be largely reduced in RA only when slices with spatial boundary values are consecutively removed. However, this case seldom happens since, at each step, the slice that mostly improves the score of the current block is not always at the spatial boundary.
Time complexity. The time complexity of the RA algorithm is $O(kLd^2n + kL^2)$ where $L = \max\{|\mathcal{B}(i)| \mid i \in [1,N]\}$. Details can be found in Lemma B.1 in Section B in the appendix.

4.2 Partitioning Algorithm (PA)

As discussed in the previous section, algorithm RA may return candidate SCD-blocks with large spatial coverage ratios. We propose the partitioning algorithm (PA) that performs partitioning operations on blocks, instead of the removing operations in the RA algorithm. The motivation is that when a block $\mathcal{B}$ is divided into two sub-blocks $\mathcal{B}_1$ and $\mathcal{B}_2$ at slice $\mathcal{B}(a,i)$, $i \in \{\Upsilon_x, \Upsilon_y\}$, the spatial coverage ratio of either of the sub-blocks will be at most 50% of that of block $\mathcal{B}$ when the dimension value a is in the middle of the range of the spatial dimension Υ_x or Υ_y.

Figure 3b shows the working process of the PA algorithm. Given a spatial tensor $\mathcal{R}$, it generates candidate SCD-blocks as follows.

Step I: Initially, let block $\mathcal{B}$ be the input spatial tensor $\mathcal{R}$.
Step II: Select a slice based on a slice selection policy among all the spatial slices of block $\mathcal{B}$.
Step III: Partition block $\mathcal{B}$ into two sub-blocks $\mathcal{B}_1$ and $\mathcal{B}_2$ at the selected slice.
Step IV: Take the sub-block with higher score, i.e., $\mathcal{B}' = \max\{f(\mathcal{B}_1), f(\mathcal{B}_2)\}$ and discard the other sub-block.
Step V: If $f(\mathcal{B}') > f(\mathcal{B})$, set $\mathcal{B} = \mathcal{B}'$ and go to Step II. Otherwise, it returns $\mathcal{B}'$ as a candidate and terminates.

The pseudo code of PA is shown in Algorithm 2 in Section A in the appendix.

Lemma 4.1 shows that the top-1 SCD-block discovered by the PA algorithm achieves the same $1/N$ bound approximation guarantee as traditional top-1 dense block found by existing DBM algorithms [28, 29, 32]. The proof is available in Lemma C.1 in Section C in the appendix.

Lemma 4.1. ***[1/N-Approximation Guarantee]*** *Given a spatial tensor $\mathcal{B}$, let $\mathcal{B}^* \subseteq \mathcal{B}$ be the block that maximizes the scoring function $f(\cdot)$ (Equation 3). Let $\hat{\mathcal{B}}$ be the block returned by the PA algorithm. If $\forall i \in \{\Upsilon_x, \Upsilon_y\}$, $|\mathcal{B}(i)| \geq \frac{1}{N} D_{\mathcal{B}}$, $\delta_{\Upsilon}^{-1}(\hat{\mathcal{B}}) \geq \frac{1}{N}$, in each partition operation, the sub-block $\mathcal{B}'$ with higher score satisfies that $\rho(\mathcal{B}') \geq \rho(\mathcal{B})$ and each slice $\mathcal{B}(a,i)$ in the discarded sub-block satisfies that $M_{\mathcal{B}(a,i)} \leq \frac{M_{\mathcal{B}}}{|\mathcal{B}(i)|}$, then we have $f(\hat{\mathcal{B}}) \geq \frac{1}{N} f(\mathcal{B}^*)$.*

Time complexity. The time complexity of the PA algorithm is $O(kLd^2n + kL^2)$ where $L = \max\{|\mathcal{B}(i)| \mid i \in [1,N]\}$. Details can be found in Lemma B.1 in Section B in the appendix.

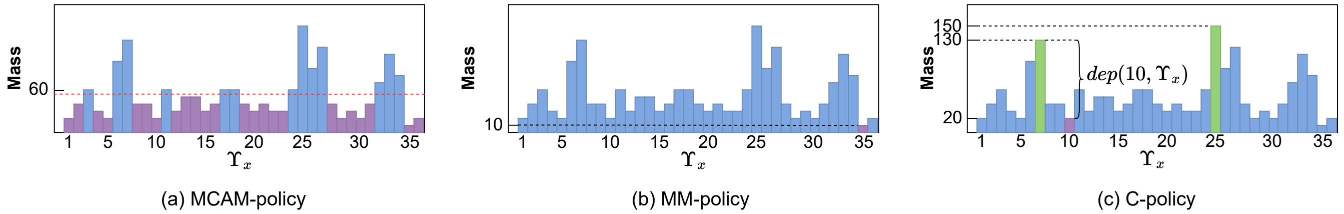
4.3 Combining RA and PA

Baseline RA finds candidate SCD-blocks by removing slices with low mass, while the proposed PA algorithm generates candidate SCD-blocks by partitioning blocks. Intuitively, we can combine the removing operation and the partitioning operation and derive the removing-partitioning algorithm (RPA). Given a spatial tensor $\mathcal{R}$, it generates candidate SCD-blocks as follows. Initially, spatial tensor $\mathcal{R}$ is taken as the block to be processed, denoted by $\mathcal{B}$. Then, it iteratively performs the removing operation or the partitioning operation on $\mathcal{B}$ until the score of $\mathcal{B}$ cannot be improved. Let $\mathcal{B}$ be the current block to be processed. Among all the slices of the spatial dimensions, a slice is selected based on a policy. It determines whether performing the removing operation or performing the partitioning operation according to which operation can result in higher score. If neither the removing operation nor the partitioning operation can improve the score of $\mathcal{B}$, it is regarded as a candidate and is passed to the refinement phase. The pseudo code of RPA is shown in Algorithm 3 in Section A in the appendix.
Approximation guarantee. Algorithm RPA follows Lemma 4.1 and has $1/N$-approximation guarantee. The proof is available in Lemma C.2 in Section C in the appendix.
Time complexity. The time complexity of the RPA algorithm is $O(kLd^2n + kL^2)$ where $L = \max\{|\mathcal{B}(i)| \mid i \in [1,N]\}$. Details can be found in Lemma B.1 in Section B in the appendix.

4.4 Slice Selection

In this section, we first briefly describe two slice selection polices in existing algorithms. Then, we propose a new slice selection policy, named the concave policy.
Maximum cardinality with average mass policy (MCAM-policy). It first selects the dimension with the largest cardinality in the current block, i.e., $\arg\max_{A_i} |\mathcal{B}(i)|, i \in [1,N]$. Next, it computes the average mass of the slices of the selected dimension, i.e., $\overline{M}_i = M_{\mathcal{B}}/|\mathcal{B}(i)|$. Then, the slices whose mass are below the average mass are selected, i.e., $\{\mathcal{B}(a,i) \mid a \in \mathcal{B}(i) \wedge M_{\mathcal{B}(a,i)} < \overline{M}_i\}$. This policy may select multiple slices and is used in D-Cube [29].
Minimum mass policy (MM-policy). It considers all slices of all dimensions. The slice with the minimum mass is selected, i.e., $\arg\min_{\mathcal{B}(a,i)} M_{\mathcal{B}(a,i)}$, $i \in [1,N]$, $a \in \mathcal{B}(i)$. This policy is used in M-Zoom [27], M-Biz [27], DenseStream [29], and DenseAlert [29].
Concave policy (C-policy). Although the MCAM-policy and the MM-policy have been widely adopted in the literature [27, 29], there are several issues for using these two policies. First, the MCAM-policy selects multiple slices at a time (e.g., the 24 purple slices in Figure 4a) which is not suitable for the partition operation. Because it may generate a lot of small-sized blocks. Second, the MM-policy chooses the slice with the minimum mass. However, this slice can be possibly close to the boundary value (e.g., the purple slice with dimension value $\Upsilon_x = 35$ in Figure 4b), so that the spatial coverage ratio cannot be reduced. In order to overcome these two issues, we propose the following concave policy.

Given block $\mathcal{B}$, considering two slices $\mathcal{B}(a_1,i)$ and $\mathcal{B}(a_2,i)$, if $a_1 < a_2$, slice $\mathcal{B}(a_1,i)$ is called on the left side of $\mathcal{B}(a_2,i)$ and slice $\mathcal{B}(a_2,i)$ is called on the right side of $\mathcal{B}(a_1,i)$. Given block $\mathcal{B}$, considering slice $\mathcal{B}(a,i)$, let $M_{max}^L(a,i) = \max_{a'<a} M_{\mathcal{B}(a',i)}$ be the maximum mass of the slices on the left side of $\mathcal{B}(a,i)$ and



(a) MCAM-policy (b) MM-policy (c) C-policy

Figure 4: Slice selection.

$M^R_{max}(a, i) = \max_{a'>a} M_{\mathcal{B}(a',i)}$ be the maximum mass of the slices on the right side of $\mathcal{B}(a, i)$. The concave depth of slice $\mathcal{B}(a, i)$ is defined as

$$\text{dep}(a, i) = \min\{M^L_{max}(a, i), M^R_{max}(a, i)\} - M_{\mathcal{B}(a,i)}. \quad (4)$$

Among all slices of all dimensions, the proposed concave policy selects the slice with the maximum concave depth. Figure 4c shows the masses of 36 slices of the longitude dimension Υ_x in an example block. Considering slice $\mathcal{B}(10, \Upsilon_x)$, we have $M_{\mathcal{B}(10,\Upsilon_x)} = 20$, $M^L_{max}(10, \Upsilon_x) = M_{\mathcal{B}(7,\Upsilon_x)} = 130$, $M^R_{max}(10, \Upsilon_x) = M_{\mathcal{B}(25,\Upsilon_x)} = 150$, and $\text{dep}(a, i) = 130 - 20 = 110$. Comparing with all the other slices, the concave depth of slice $\mathcal{B}(10, \Upsilon_x)$ is the largest, so that it will be selected.

5 Bottom-Up Solution

The top-down solution starts from the entire spatial tensor and mines top-k SCD-blocks by partitioning blocks or removing slices with low density values. This section proposes a bottom-up solution, i.e., the expansion algorithm (EA), which finds candidate SCD-blocks in a different way. It first divides the spatial tensor into small cells. Then, potential small cells are combined to form candidate SCD-blocks with good scores.

5.1 Cell Construction

The bottom-up solution considers the tuples of a spatial tensor $\mathcal{R}$ in a two dimensional spatial space with axes Υ_x and Υ_y, denoted by $\mathbb{R}^2$. The ranges of axes Υ_x and Υ_y are defined as $[\Upsilon^L_x(\mathcal{R}), \Upsilon^U_x(\mathcal{R})]$ and $[\Upsilon^L_y(\mathcal{R}), \Upsilon^U_y(\mathcal{R})]$, respectively. Each axis is split into h consecutive intervals, so that space $\mathbb{R}^2$ is divided into $h \times h$ cells. Each cell c is represented by a pair $(c.x, c.y)$ that indicates the positions of c at the two axes. A tuple t belongs to a cell c if the spatial dimensions of t fall within the cell, denoted by $t \in c$. Figure 5a illustrates the 7×7 cells in the space $\mathbb{R}^2$ of a spatial tensor. The bottom-up solution considers the cells containing tuples and ignores empty cells. A cell $c = (c.x, c.y)$ has the following four neighbors in space $\mathbb{R}^2$: $(c.x + 1, c.y)$, $(c.x - 1, c.y)$, $(c.x, c.y + 1)$, and $(c.x, c.y - 1)$. For example, considering the orange cell $c = (2, 3)$ shown in Figure 5a, its four neighbors (shown as grey cells) are $(3, 3)$, $(1, 3)$, $(2, 4)$, and $(2, 2)$. The mass of a cell is calculated as $M_c = \sum_{t \in c} t[X]$.

5.2 Expansion Algorithm (EA)

The proposed EA algorithm processes the cells in space $\mathbb{R}^2$ in descending order of their mass. It starts with the cell having the largest mass and tries to generate a candidate SCD-block by combining it with its nearby cells. In other words, starting from a cell, a candidate SCD-block is formed by several times of expansions. Specifically, let $\mathcal{B}$ be the current block that is composed of a set of cells. Initially, $\mathcal{B}$ only contains the starting cell. For each cell $c \in \mathcal{B}$, considering each neighbor n^c of cell c, if the score of block $\mathcal{B}$ is improved by adding n^c, the current block $\mathcal{B}$ is expanded by combining $\mathcal{B}$ and n^c. The expansion of block $\mathcal{B}$ stops when no neighbor can improve the score of $\mathcal{B}$. After that, block $\mathcal{B}$ is passed to the refinement phase. The cells included in $\mathcal{B}$ are removed from the spatial space. Next, algorithm EA selects the cell having the largest mass in the rest of the cells and generates the next candidate SCD-block using the same process above.

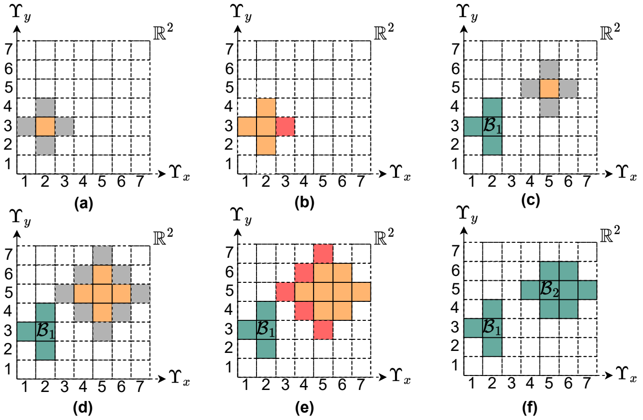


Figure 5: Expansion algorithm.

Figure 5 illustrates the process of the EA algorithm. Suppose that one block expansion starts from the orange cell shown in Figure 5a. The four neighbors of the orange cell are shown as grey cells. In Figure 5b, three neighbors are combined with the current block, shown as the orange part, while the red neighbor does not improve the score, so that it is discarded. Next, no neighbor can further improve the score, so that the current block expansion stops. The green block $\mathcal{B}_1$ shown in Figure 5c is taken as a candidate SCD-block and passed to the refinement phase. After that, suppose that another block expansion starts from the orange cell shown in Figure 5c. Since the four neighbors (shown as the grey cells in Figure 5c) improve the score, they are combined with the current block, shown as the orange part in Figure 5d. Next, the neighbor cells of the current block are considered (the grey cells in Figure 5d). As shown in Figure 5e, three neighbors are combined with the current block and five neighbors are discarded. Then, the score of the current block cannot be further improved, so that the green block $\mathcal{B}_2$ shown in Figure 5f is taken as a candidate SCD-block and passed to the refinement phase. The pseudo code of EA is shown in Algorithm 4 in Section A in the appendix.

Time complexity. The time complexity of the EA algorithm is $O(h^2 \log h^2 + h^2 n + Ld^2 n)$ where $L = \max\{|\mathcal{B}(i)| \mid i \in [1, N]\}$. Details can be found in Lemma B.2 in Section B in the appendix.

Table 1: Statistics of datasets.

Dataset	Schema	Volume	#Tuples
Gowalla_NA	user, place, check-in-time, address (longitude, latitude)	56K × 691K × 596 × 231 × 201	6,442,890
Brightkite_Euro	user, place, check-in-time, address (longitude, latitude)	5K × 14K × 203 × 239 × 151	51,352
Foursquare	user, place, check-in-time, address (longitude, latitude)	266K × 3.68M × 455 × 1000 × 1000	33,263,632
EOrder	buyer, seller, item, pay-time, address (longitude, latitude)	71.11M × 55K × 2.30M × 122 × 1000 × 1000	115,631,169

6 Experiments

Datasets. In our experiments, four real datasets are used to evaluate the performance of the proposed algorithms. Gowalla_NA, Brightkite_Euro, and Foursquare [31] are check-in datasets from location-based social networking websites. Gowalla_NA is a subset of the Gowalla dataset [8]. It contains the check-in records located in North America. Brightkite_Euro is a subset of the Brightkite dataset [8]. It contains the check-in records located in Europe. EOrder contains orders from an E-commerce website. In each dataset, we discretize the ranges of the longitude and the latitude dimensions into slots. The unit of the spatial space of each dataset is defined as 1 longitude slot × 1 latitude slot. Table 1 shows the statistics of these datasets.

Evaluated algorithms. We compare the proposed algorithms PA, RPA, and EA with the baseline algorithm RA and four existing DBM algorithms: D-Cube [29], M-Zoom [28], M-Biz [28], and AugSplicing [32]. By default, algorithms RA, PA, and RPA adopt the C-policy for slice selection. Algorithm M-Biz is used in the refinement phase. The source code of the proposed algorithms is available at https://anonymous.4open.science/r/SP_Block-C0BB.

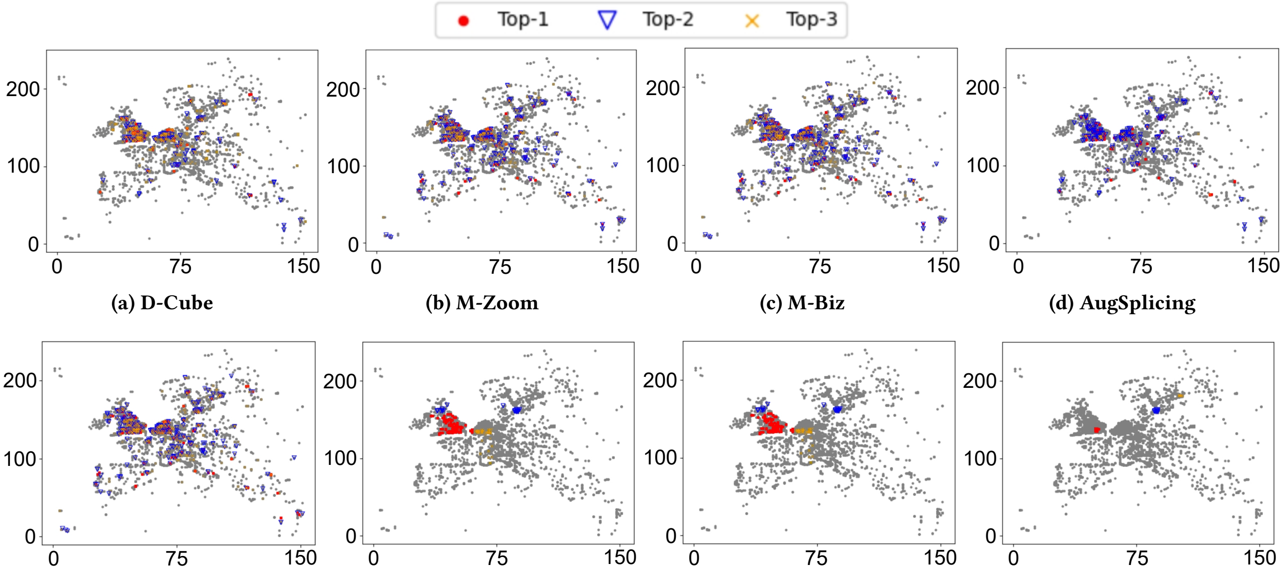
Settings. All the algorithms are running on a machine with 4 Intel Xeon E7-4830 CPUs (56 cores, 2.0 GHz) and 2TB main memory. By default, parameter α in Equation 3 and parameter h in the EA algorithm are set to 0.5 and 50, respectively.

Metrics. To evaluate the SCD-blocks returned by different algorithms, we report the density values, the spatial coverage ratios, and the numbers of tuples contained.

Spatial compactness visualization. Figure 6 plots the spatial locations of the tuples in the top-3 blocks returned by different algorithms on dataset Brightkite_Euro. Considering the top-3 blocks found by the four existing algorithms D-Cube, M-Zoom, M-Biz, and AugSplicing and the baseline algorithm RA, they are spatially loose and overlap each other, almost covering the whole spatial region of the dataset. Regarding the top-3 blocks discovered by our algorithms PA, RPA, and EA, they are spatially compact and well separated. Specifically, algorithms PA and RPA find country-level SCD-blocks, i.e., two SCD-blocks in the region of UK and one SCD-block in the region of Germany. Algorithm EA discovers city-level SCD-blocks, i.e, SCD-blocks in London, Stockholm, and Boden. These SCD-blocks cannot be discovered by existing DBM algorithms. Algorithms PA and RPA produce similar SCD-blocks. It is because that in the RPA algorithm, the partitioning operation improves the scores more than the removing operation does in most cases, so that the number of performed partitioning operations is more than the number of performed removing operations. In other words, the proposed partitioning operation is effective in mining SCD-blocks. The SCD-blocks found by the EA algorithm are smaller than that of the PA and the RPA algorithms. The reason is that the EA algorithm stops expansions of blocks when encountering cells with low mass. More visualization results on the other datasets are included in the appendix. They are consistent with the observations described above.

Assessment of the top-k SCD-blocks. The four figures in the first row of Figure 7 compares all algorithms in terms of density $\tilde{\rho}$ and spatial coverage ratio δ_{Υ} on the four datasets. The four figures in the second row of Figure 7 show the numbers of tuples contained in the top-5 blocks returned by different algorithms on the four datasets. In most cases, the proposed algorithms PA, RPA, and EA find significantly spatially compact blocks than RA, D-Cube, M-Zoom, M-Biz, and AugSplicing. On dataset Foursquare, AugSplicing returns only two blocks, because AugSplicing combines temporally adjacent sub-blocks. When these sub-blocks happen to contain spatially close tuples, AugSplicing may find spatially compact dense blocks. The density values of the blocks found by our algorithms PA, RPA, and EA are comparable with that of algorithms D-Cube, M-Zoom, M-Biz, and RA, in most cases. On dataset EOrder, the density values of the blocks found by our algorithms are lower than that of existing algorithms. The reason is that EOrder is sparse compared with the other datasets, so that the density values of the blocks found by existing algorithms are contributed by the tuples spatially scattered. In terms of the size of the block, i.e., the number of tuples contained, algorithms D-Cube, M-Zoom, M-Biz, and RA return large blocks, whereas, algorithms AugSplicing, PA, RPA, and EA find either moderate or small blocks. In community and marketing applications, moderate and small blocks are preferred than large blocks for fine-grained analysis.

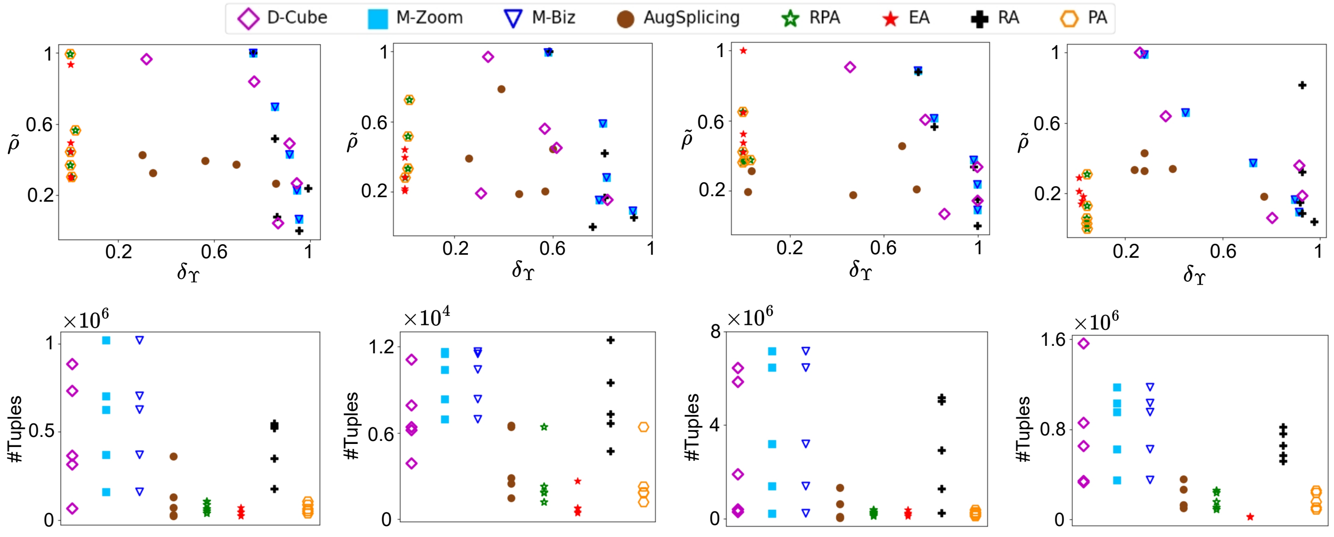
According to the reported results, we summarize the characteristics of different algorithms as follows. Existing algorithms D-Cube, M-Zoom, M-Biz, and AugSplicing treat spatial dimensions the same as traditional dimensions and find blocks with high density values by removing slices with low mass. Thus, they return large-sized blocks, so that their spatial coverage ratios are high. Although baseline RA processes spatial dimensions in the first phase and then refines traditional dimensions in the second phase, the removing operation cannot reduce the spatial coverage ratio. Hence, RA cannot find SCD-blocks. The partition operations in PA and RPA are effective in reducing the spatial coverage ratio and the proposed C-policy is able to discover dense blocks. Thus, our algorithms PA and RPA produce SCD-blocks. Our EA algorithm discovers fine-grained SCD-blocks. The reason is that EA starts from a small cell and expands it by combining neighboring small cells. The current block stops being expanded when encountering low-density small cells. Unless there exist many consecutive high-density small cells in the dataset, EA may find moderate-sized blocks.



(a) D-Cube (b) M-Zoom (c) M-Biz (d) AugSplicing

(e) RA (f) PA (g) RPA (h) EA

Figure 6: Spatial locations of the tuples in the top-3 blocks found by different algorithms on Brightkite_Euro.



(a) Gowalla_NA (b) Brightkite_EA (c) Foursquare (d) EOrder

Figure 7: Assessment of the top-5 blocks found by different algorithms.

Effect of slice selection policy. To show the effectiveness of the proposed C-policy, we adopt the MM-policy and the C-policy in algorithms RA, PA, and RPA. In addition, since the MCAM policy selects multiple slices, we only adopt it in the RA algorithm. Figure 8 shows the density values $\tilde{\rho}$ and the spatial coverage ratios δ_Υ of the top-5 blocks returned by algorithms RA, PA, and RPA using different slice selection policies on dataset Foursquare. Regarding algorithm RA, the spatial coverage ratios of the blocks using different policies all exceed 50% that is worse than algorithms PA and RPA. In algorithms PA and RPA, the C-policy outperforms the MM-policy in terms of the spatial coverage ratios and is comparable to the MM-policy according the density values. Considering algorithm PA, although the MM-policy returns one block whose density value is higher than that of the C-policy, its spatial coverage ratio is larger than that of the C-policy. Thus, the proposed C-policy helps algorithms PA and RPA finding SCD-blocks.

Varying h in the EA algorithm. By default, h is set to 50, meaning that in algorithm EA, the two dimensional $\Upsilon_x\Upsilon_y$ space of the dataset is divided into 50 × 50 cells. To evaluate the effect of parameter h, we also try another four values, i.e., $h = 10$, $h = 20$, $h = 100$, and $h = 150$. Figure 9 shows the density values and the spatial coverage ratios of the top-5 blocks returned by the EA algorithm when varying h on dataset Foursquare. When h increases from 50 to

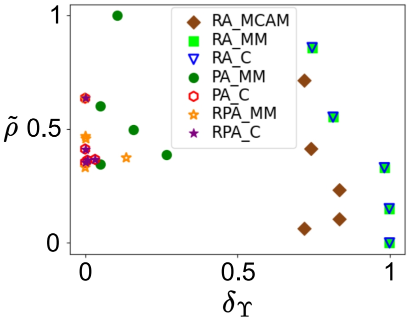


Figure 8: Slice selection.

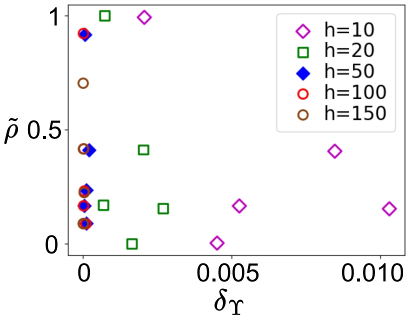


Figure 9: Varying h.

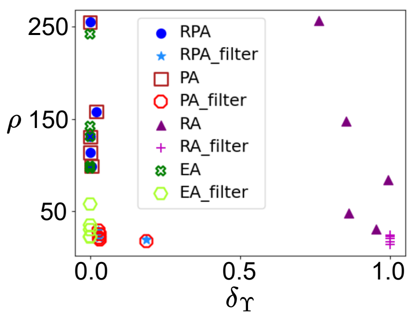


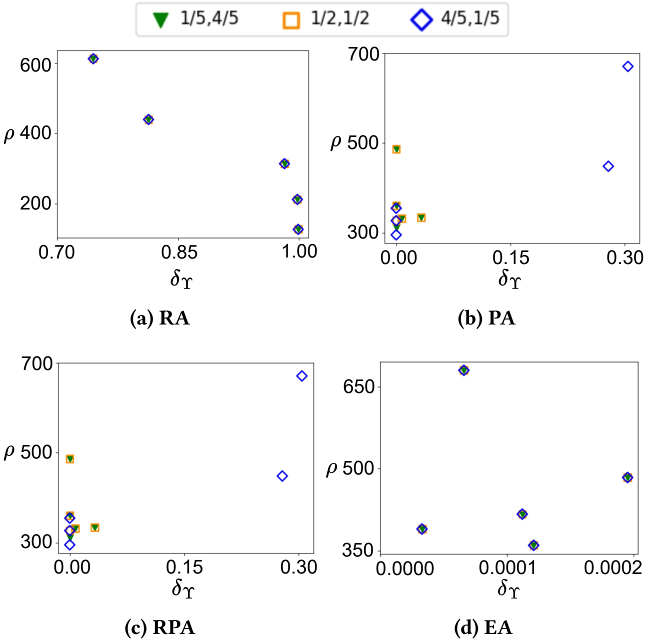
Figure 10: Effect of refinement phase.

150, the spatial coverage ratios of the blocks do not change, while the density values of the blocks decrease slightly. Using large h tends to find small blocks. This is because small cells contain few tuples, so that the chance of increasing the density value of the current block by combining neighbor cell is low. When h decreases from 50 to 10, the spatial coverage ratios of the blocks become large and the density values of the blocks increase a bit. Using small h tends to find large blocks. The reason is that large cells include more tuples, so that it is probably that more neighbor cubes will be combined. Hence, we can tune h to get blocks with good density values and small spatial coverage ratios.

Effect of refinement phase. We compare the densities and spatial coverage ratios of the SCD-blocks before and after refinement. Figure 10 show the results on dataset Gowalla_NA. We observe that the spatial coverage ratios of the SCD-blocks do not change much while the densities increase a lot after the refinement phase.

Varying parameter α in the scoring function. By default, parameter α is set to 1/2, indicating that the density value and the spatial coverage ratio are considered equally in the scoring function. To evaluate the effect of different weighing schemes, we also apply another two parameter settings, i.e., $\alpha = 1/5$ and $\alpha = 4/5$ in algorithms RA, PA, RPA, and EA. Figure 11 shows the density values and the spatial coverage ratios of the top-5 blocks returned by algorithms RA, PA, RPA, and EA when varying the weighing scheme in the scoring function on dataset Foursquare.

Algorithms RA and EA do not affected by the weighing scheme. This is because algorithm RA only removes a few slices in the filtering phase, so that the weights in the scoring function does not change the blocks much in the result. Algorithm EA divides the spatial space into 50 × 50 cells. The neighboring cells that can be combined do not vary much with the weights. The weighing scheme has similar effect on PA and RPA. When $\alpha = 1/5$, i.e., more attention is given to the spatial coverage ratio, as expected, the returned blocks have better spatial ratios. When $\alpha = 4/5$, i.e., more attention is given to the density values, the returned blocks have worse spatial ratios as expected and the density values increase.



(a) RA **(b) PA**

(c) RPA **(d) EA**

Figure 11: Varying parameter α in the scoring function.

7 Conclusion

We formulate the new problem of mining top-k SCD-blocks in spatial tensors, which aims to discover dense blocks with small spatial coverage ratio. Since it is an NP-hard problem, we approximately solve this problem using a filter-refinement framework by incorporating the newly developed top-down solution and bottom-up solution. The top-down solution proposes a partition operation with a slice selection policy that generates candidate SCD-blocks with high density values and low spatial coverage ratios. The bottom-up solution divides the spatial tensor into small cells in the spatial space and generate good candidates by combining nearby cells. We verify the effectiveness of the proposed solutions on four real datasets.

In the future, we plan to further investigate how different types of scoring functions can affect the effectiveness of this DBM task and explore blocks that are compact in both temporal dimension and spatial dimensions.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grants 62372308 and 62202401 and the GuangDong Basic and Applied Basic Research Foundation under Grant 2023A1515011619.

References

[1] Ralph Abraham, Jerrold E Marsden, and Tudor Ratiu. 2012. *Manifolds, tensor analysis, and applications*. Vol. 75. Springer Science & Business Media.

[2] Yuichi Asahiro, Refael Hassin, and Kazuo Iwama. 2002. Complexity of finding dense subgraphs. *Discret. Appl. Math.* 121, 1-3 (2002), 15–26.

[3] Yikun Ban, Xin Liu, Ling Huang, Yitao Duan, Xue Liu, and Wei Xu. 2019. No Place to Hide: Catching Fraudulent Entities in Tensors. In *WWW*. 83–93.

[4] Michael Batty. 2013. *The New Science of Cities*. MIT Press.

[5] Siddharth Bhatia, Arjit Jain, Pan Li, Ritesh Kumar, and Bryan Hooi. 2021. MStream: Fast Anomaly Detection in Multi-Aspect Streams. In *WWW*. 3371–3382.

[6] Louis Brand. 2020. *Vector and tensor analysis*. Courier Dover Publications.

[7] Nicholas H. Chan and Steven C. Chan. 2011. *Financial Econometrics: Models and Methods*. Wiley.

[8] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *KDD*. 1082–1090.

[9] Quang-Huy Duong, Heri Ramampiaro, and Kjetil Nørvåg. 2020. Multiple Dense Subtensor Estimation with High Density Guarantee. In *ICDE*. 637–648.

[10] Quang-Huy Duong, Heri Ramampiaro, Kjetil Nørvåg, and Thu-Lan Dam. 2021. Density Guarantee on Finding Multiple Subgraphs and Subtensors. *ACM Trans. Knowl. Discov. Data* 15, 5 (2021), 76:1–76:32.

[11] Wenjie Feng, Shenghua Liu, and Xueqi Cheng. 2019. CatchCore: Catching Hierarchical Dense Subtensor. In *PKDD*. 156–172.

[12] Wenjie Feng, Shenghua Liu, and Xueqi Cheng. 2023. Hierarchical Dense Pattern Detection in Tensors. *ACM Trans. Knowl. Discov. Data* 17, 6 (2023), 81:1–81:29.

[13] Ekta Gujral, Ravdeep Pasricha, and Evangelos E. Papalexakis. 2018. *SamBaTen: Sampling-based Batch Incremental Tensor Decomposition*. SIAM, 387–395.

[14] Jun-Gi Jang and U Kang. 2023. Static and Streaming Tucker Decomposition for Dense Tensors. *ACM Trans. Knowl. Discov. Data* 17, 5 (2023), 66:1–66:34.

[15] Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos. 2015. A General Suspiciousness Metric for Dense Blocks in Multimodal Data. In *ICDM*. 781–786.

[16] Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos. 2016. Spotting suspicious behaviors in multimodal data: A general metric and algorithms. *IEEE Transactions on Knowledge and Data Engineering* 28, 8 (2016), 2187–2200.

[17] Samir Khuller and Barna Saha. 2009. On Finding Dense Subgraphs. In *ICALP*, Vol. 5555. 597–608.

[18] Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review* 51, 3 (2009), 455–500.

[19] X. Li and L. Wang. 2007. A review of spatial data compression techniques. *International Journal of Remote Sensing* 28, 16 (2007), 3675–3700.

[20] Geert Litjens, Thijs Kooi, Behzad Etemadi Bejnordi, et al. 2017. A survey on deep learning in medical image analysis. *Medical Image Analysis* 42 (2017), 60–88.

[21] Paul A. Longley, Michael F. Goodchild, David J. Maguire, and David W. Rhind. 2015. *Geographic Information Systems and Science* (4th ed.). Wiley.

[22] C. R. Margules and R. L. Pressey. 2000. Systematic conservation planning. *Nature* 405, 6783 (2000), 243–253.

[23] Koji Maruhashi, Fan Guo, and Christos Faloutsos. 2011. MultiAspectForensics: Pattern Mining on Large-Scale Heterogeneous Networks with Tensor Analysis. In *ASONAM*. 203–210.

[24] John A. Richards. 2013. *Remote Sensing Digital Image Analysis* (5th ed.). Springer.

[25] Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. 2010. Dense subgraphs with restrictions and applications to gene annotation graphs. In *RECOMB*. 456–472.

[26] Lei Shi, Aryya Gangopadhyay, and Vandana P. Janeja. 2015. STenSr: Spatio-temporal tensor streams for anomaly detection and pattern discovery. *Knowl. Inf. Syst.* 43, 2 (2015), 333–353.

[27] Kijung Shin, Bryan Hooi, and Christos Faloutsos. 2016. M-Zoom: Fast Dense-Block Detection in Tensors with Quality Guarantees. In *ECML/PKDD*, Vol. 9851. 264–280.

[28] Kijung Shin, Bryan Hooi, and Christos Faloutsos. 2018. Fast, Accurate, and Flexible Algorithms for Dense Subtensor Mining. *ACM Trans. Knowl. Discov. Data* 12, 3 (2018), 28:1–28:30.

[29] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. 2017. D-Cube: Dense-Block Detection in Terabyte-Scale Tensors. In *WSDM*. 681–689.

[30] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. 2017. DenseAlert: Incremental Dense-Subtensor Detection in Tensor Streams. In *SIGKDD*. 1057–1066.

[31] Dingqi Yang, Daqing Zhang, and Bingqing Qu. 2016. Participatory Cultural Mapping Based on Collective Behavior Data in Location-Based Social Networks. *ACM Trans. Intell. Syst. Technol.* 7, 3 (2016), 30:1–30:23.

[32] Jiabao Zhang, Shenghua Liu, Wenting Hou, Siddharth Bhatia, Huawei Shen, Wenjian Yu, and Xueqi Cheng. 2021. AugSplicing: Synchronized Behavior Detection in Streaming Tensors. In *AAAI*. 4653–4661.

[33] Shuo Zhou, Nguyen Xuan Vinh, James Bailey, Yunzhe Jia, and Ian Davidson. 2016. Accelerating Online CP Decompositions for Higher Order Tensors. In *SIGKDD*. 1375–1384.

A Pseudo Code

Algorithm 1 shows the pseudo code of the RA algorithm in Section 4.1. Algorithm 2 shows the pseudo code of the PA algorithm in Section 4.2. Algorithm 3 shows the pseudo code of the RPA algorithm in Section 4.3. Algorithm 4 shows the pseudo code of the EA algorithm in Section 5.2.

Algorithm 1: Removing Algorithm (RA)

```
Input  : spatial tensor $\mathcal{R}$, the number $k$ of required
         SCD-blocks.
Output : the list $\mathbb{B}$ of $k$ candidate SCD-blocks.
1  $\mathbb{B} \leftarrow \emptyset$;
2  for $i \leftarrow 1 \cdots k$ do
3      $\mathcal{B} \leftarrow \mathcal{R}$;
4      while true do
5          $\mathcal{B}(a, i) \leftarrow$ sliceSelection($\Upsilon_x, \Upsilon_y$)
            $\mathcal{B}' \leftarrow$ remove($\mathcal{B}, a, i$);
6          if $f(\mathcal{B}') > f(\mathcal{B})$ then
7              $\mathcal{B} \leftarrow \mathcal{B}'$;
8              go to line 4;
9          break;
10     $\mathcal{B} \leftarrow$ refine($\mathcal{B}$);
11     $\mathcal{R} \leftarrow \mathcal{R} - \mathcal{B}$;
12     Add $\mathcal{B}$ to $\mathbb{B}$;
13 return $\mathbb{B}$;
```

Algorithm 2: Partitioning Algorithm (PA)

```
Input  : spatial tensor $\mathcal{R}$, the number $k$ of required
         SCD-blocks.
Output : the list $\mathbb{B}$ of $k$ candidate SCD-blocks.
1  $\mathbb{B} \leftarrow \emptyset$;
2  for $i \leftarrow 1 \cdots k$ do
3      $\mathcal{B} \leftarrow \mathcal{R}$;
4      while true do
5          $\mathcal{B}(a, i) \leftarrow$ sliceSelection($\Upsilon_x, \Upsilon_y$)
            $\mathcal{B}_1, \mathcal{B}_2 \leftarrow$ partition($\mathcal{B}, a, i$);
6          if $max(f(\mathcal{B}_1), f(\mathcal{B}_2)) > f(\mathcal{B})$ then
7              if $f(\mathcal{B}_1) > f(\mathcal{B}_2)$ then
8                  $\mathcal{B} \leftarrow \mathcal{B}_1$;
9              else if $f(\mathcal{B}_1) \leq f(\mathcal{B}_2)$ then
10                 $\mathcal{B} \leftarrow \mathcal{B}_2$;
11             go to line 4;
12         break;
13     $\mathcal{B} \leftarrow$ refine($\mathcal{B}$);
14     $\mathcal{R} \leftarrow \mathcal{R} - \mathcal{B}$;
15     Add $\mathcal{B}$ to $\mathbb{B}$;
16 return $\mathbb{B}$;
```

Algorithm 3: Removing-Partitioning Algorithm (RPA)

```
Input  : spatial tensor $\mathcal{R}$, the number $k$ of required
         SCD-blocks.
Output : the list $\mathbb{B}$ of $k$ candidate SCD-blocks.
1  $\mathbb{B} \leftarrow \emptyset$;
2  for $i \leftarrow 1 \cdots k$ do
3      $\mathcal{B} \leftarrow \mathcal{R}$;
4      while true do
5          $\mathcal{B}(a, i) \leftarrow$ sliceSelection($\Upsilon_x, \Upsilon_y$)
            $\mathcal{B}_1, \mathcal{B}_2 \leftarrow$ partition($\mathcal{B}, a, i$);
6          $\mathcal{B}' \leftarrow$ remove($\mathcal{B}, a, i$);
7          if $max(f(\mathcal{B}_1), f(\mathcal{B}_2)) > f(\mathcal{B}')$ then
8              if $f(\mathcal{B}_1) > f(\mathcal{B}_2)$ then
9                  $\mathcal{B} \leftarrow \mathcal{B}_1$;
10             else if $f(\mathcal{B}_1) \leq f(\mathcal{B}_2)$ then
11                 $\mathcal{B} \leftarrow \mathcal{B}_2$;
12             go to line 4;
13         else if $max(f(\mathcal{B}_1), f(\mathcal{B}_2)) \leq f(\mathcal{B}')$ then
14             $\mathcal{B} \leftarrow \mathcal{B}'$;
15             go to line 4;
16         break;
17     $\mathcal{B} \leftarrow$ refine($\mathcal{B}$);
18     $\mathcal{R} \leftarrow \mathcal{R} - \mathcal{B}$;
19     Add $\mathcal{B}$ to $\mathbb{B}$;
20 return $\mathbb{B}$;
```

Algorithm 4: Expansion Algorithm (EA)

```
Input  : spatial tensor $\mathcal{R}$, the number $k$ of required
         SCD-blocks.
Output : the list $\mathbb{B}$ of $k$ candidate SCD-blocks.
$G \leftarrow$ partition $\mathcal{R}$ into $h \times h$ cells along the
    latitude-longitude dimensions;
$\mathbb{B} \leftarrow \emptyset$;
Sort the small blocks in $G$ in descending order of mass;
for each small block $\mathcal{B} \in G$ do
    if $\mathcal{B}$ has been processed then
        continue;
    while true do
        $\mathcal{N} \leftarrow$ getNeighbors($\mathcal{B}$);
        $\mathcal{B}' \leftarrow \mathcal{B}$;
        for each neighbor $b \in \mathcal{N}$ do
            if $f(\mathcal{B} + b) > f(\mathcal{B})$ then
                $\mathcal{B} \leftarrow \mathcal{B} + b$;
        if $f(\mathcal{B}) = f(\mathcal{B}')$ then
            break;
    Add $\mathcal{B}$ to $\mathbb{B}$;
for each block $\mathcal{B} \in \mathbb{B}$ do
    refine($\mathcal{B}$);
return $\mathbb{B}$;
```

B Time Complexity

We provide the worst-case time complexity of algorithms PA, RA, and RPA in Lemma B.1 and the worst-case time complexity of algorithm EA in Lemma B.2. As a remark, although PA, RA, and RPA have the same worst-case time complexity, in practice, PA and RPA are considerable faster than RA because the partitioning operation reduces the size of the blocks under processing.

Lemma B.1. *The worst-case time complexity of algorithms PA, RA, and RPA are both $O(kLd^2n + kL^2)$, where $L = \max\{|\mathcal{B}(i)| \mid i \in [1, N]\}$, n is the number of tuples in $\mathcal{R}$, and k is the number of SCD-blocks required.*

Proof. In the filtering phase, the RPA algorithm executes the removing and the partitoning operations, while the PA algorithm only executes the partitioning operation, and the RA algorithm only executes the removing operation. In the worst case, the partition operation is equivalent to the removing operation, i.e., dividing the block into a sub-block and a slice. Thus, the time complexity of the partition operation is the same as that of the removing operation, i.e., $O(n)$ time. A slice $\mathcal{B}(a, i)$ of dimension i cannot be removed when a is the only one value left. Let L be the maximum value of the numbers of distinct values of all dimensions. Then, the removing operation is executed at most $2L - 2$ times. For each removing operation, the cost of selecting a slice is $O(L)$. Hence, the time complexity of the filtering phase is $O(k(2L - 2)(n + L))$. In the refinement phase, existing DBM algorithms are used that takes $O(kLd^2n)$ time [29]. Therefore, the worst-case time complexity of algorithms PA, RA, and RPA are both $O(k(2L-2)(n+L)+kLd^2n)) = O(kL^2 + kLn + kLd^2n) = O(kLd^2n + kL^2)$. □

Lemma B.2. *The worst-case time complexity of the EA algorithm is $O(h^2 \log h^2 + h^2 n + Ld^2n)$ where $L = \max\{|\mathcal{B}(i)| \mid i \in [1, N]\}$, n is the number of tuples in $\mathcal{R}$, and k is the number of SCD-blocks required.*

Proof. In the EA algorithm, there are h^2 cells. Calculating the masses of all cells takes $O(n)$ time and sorting cells takes $O(h^2 \log h^2)$ time. Then, the EA algorithm tries to expand each cell by checking it neighbors. Thus, the number of iterations is at most h^2. It expands at most n tuples. Hence, the time complexity of the expansion is $O(h^2n)$. The refinement phase uses an existing DBM algorithm whose time complexity is $O(kLd^2n)$ [29]. Then, the worst-case time complexity of the EA algorithm is $O(h^2 \log h^2 + h^2n + kLd^2n)$. □

C Approximation guarantee

Lemma C.1. *Given a spatial tensor $\mathcal{B}$, let $\mathcal{B}^* \subseteq \mathcal{B}$ be the block that maximizes the scoring function $f(\cdot)$ (Equation 3). Let $\hat{\mathcal{B}}$ be the block returned by the PA algorithm. If $\forall i \in \{\Upsilon_x, \Upsilon_y\}$, $|\mathcal{B}(i)| \geq \frac{1}{N} D_{\mathcal{B}}$, $\delta_{\Upsilon}^{-1}(\hat{\mathcal{B}}) \geq \frac{1}{N}$, in each partition operation, the sub-block with higher score (e.g., $\mathcal{B}_1$ in Step IV) satisfies that $\rho(\mathcal{B}_1) \geq \rho(\mathcal{B})$, and all slices in the discarded sub-block (e.g., $\mathcal{B}_2$ in Step IV) $\mathcal{B}(a, i)$ satisfies that $M_{\mathcal{B}(a,i)} \leq \frac{M_{\mathcal{B}}}{|\mathcal{B}(i)|}$, then we have $f(\hat{\mathcal{B}}) \geq \frac{1}{N} f(\mathcal{B}^*)$.*

Proof. Since $\mathcal{B}^*$ maximizes $f(\mathcal{B})$, removing slice $\mathcal{B}^*(a, i)$ from $\mathcal{B}^*$ will decrease its score, i.e., $f(\mathcal{B}^* - \mathcal{B}^*(a, i)) \leq f(\mathcal{B}^*)$. Then, according to Equation 3, we have

$$\tilde{\rho}(\mathcal{B}^* - \mathcal{B}^*(a,i)) + \delta_\Upsilon^{-1}(\mathcal{B}^* - \mathcal{B}^*(a,i)) \leq \tilde{\rho}(\mathcal{B}^*) + \delta_\Upsilon^{-1}(\mathcal{B}^*)$$

Obviously, removing a slice does not increase the spatial coverage ratios, i.e., $\delta_\Upsilon^{-1}(\mathcal{B}^* - \mathcal{B}^*(a,i)) \geq \delta_\Upsilon^{-1}(\mathcal{B}^*)$. Then, we can derive that $\rho(\mathcal{B}^* - \mathcal{B}^*(a,i)) \leq \rho(\mathcal{B}^*)$. By substituting Equation 1 in, we have that $\frac{M_{\mathcal{B}^*} - M_{\mathcal{B}^*(a,i)}}{(D_{\mathcal{B}^*}-1)/N} \leq \frac{M_{\mathcal{B}^*}}{D_{\mathcal{B}^*}/N}$ and derive that

$$M_{\mathcal{B}^*(a,i)} \geq \rho(\mathcal{B}^*)/N. \tag{5}$$

In each partition operation, we have

$$\rho(\hat{\mathcal{B}}) \geq \rho(\mathcal{B}_1) \geq \rho(\mathcal{B}). \tag{6}$$

Since $|\mathcal{B}(i)| \geq \frac{1}{N} D_{\mathcal{B}}$, we can derive that

$$\rho(\mathcal{B}) = \frac{M_{\mathcal{B}}}{D_{\mathcal{B}}/N} \geq \frac{M_{\mathcal{B}}}{|\mathcal{B}(i)|}. \tag{7}$$

For all the slices in the discarded sub-block $\mathcal{B}(a,i)$, we have

$$M_{\mathcal{B}(a,i)} \leq \frac{M_{\mathcal{B}}}{|\mathcal{B}(i)|}. \tag{8}$$

Since $\mathcal{B}^* \subseteq \mathcal{B}$, we have that

$$M_{\mathcal{B}(a,i)} \geq M_{\mathcal{B}^*(a,i)}. \tag{9}$$

Combining Equations 5–9, we can derive that

$$\rho(\hat{\mathcal{B}}) \geq \frac{1}{N}\rho(\mathcal{B}^*). \tag{10}$$

Since $\delta_\Upsilon^{-1}(\hat{\mathcal{B}}) \geq \frac{1}{N}$, and $0 \leq \delta_\Upsilon^{-1}(\mathcal{B}^*) \leq 1$, it can be inferred that

$$\delta_\Upsilon^{-1}(\hat{\mathcal{B}}) \geq \frac{1}{N}\delta_\Upsilon^{-1}(\mathcal{B}^*). \tag{11}$$

Combining Equations 10 and 11, we have $f(\hat{\mathcal{B}}) \geq \frac{1}{N} f(\mathcal{B}^*)$. □

Lemma C.2. *Given a spatial tensor $\mathcal{B}$, let $\mathcal{B}^* \subseteq \mathcal{B}$ be the block that maximizes the scoring function $f(\cdot)$ (Equation 3). Let $\hat{\mathcal{B}}$ be the block returned by the RPA algorithm. If $\forall i \in \{\Upsilon_x, \Upsilon_y\}$, $|\mathcal{B}(i)| \geq \frac{1}{N} D_{\mathcal{B}}$, $\delta_\Upsilon^{-1}(\hat{\mathcal{B}}) \geq \frac{1}{N}$, in each partition operation, the sub-block with higher score (e.g., $\mathcal{B}_1$ in Step IV) satisfies that $\rho(\mathcal{B}_1) \geq \rho(\mathcal{B})$, and all slices in the discarded sub-block (e.g., $\mathcal{B}_2$ in Step IV) $\mathcal{B}(a,i)$ satisfies that $M_{\mathcal{B}(a,i)} \leq \frac{M_{\mathcal{B}}}{|\mathcal{B}(i)|}$. In each removing operation, the removed slice satisfies that $M_{\mathcal{B}(a,i)} \leq \frac{M_{\mathcal{B}}}{|\mathcal{B}(i)|}$ and the sub-block $\mathcal{B}_3$ after the removing operation satisfies condition $\rho(\mathcal{B}_3) \geq \rho(\mathcal{B})$, then we have $f(\hat{\mathcal{B}}) \geq \frac{1}{N} f(\mathcal{B}^*)$.*

Proof. The same as Lemma C.1 shows that the result of the partition operation satisfies the 1/N-approximation guarantee. The removing operation given below can also obtain the 1/N-approximation guarantee, that is, the approximation ratio of RPA. Since $\mathcal{B}^*$ maximizes $f(\mathcal{B})$, removing slice $\mathcal{B}^*(a,i)$ from $\mathcal{B}^*$ will decrease its score, i.e., $f(\mathcal{B}^* - \mathcal{B}^*(a,i)) \leq f(\mathcal{B}^*)$. Then, according to Equation 3, we have

$$\tilde{\rho}(\mathcal{B}^* - \mathcal{B}^*(a,i)) + \delta_\Upsilon^{-1}(\mathcal{B}^* - \mathcal{B}^*(a,i)) \leq \tilde{\rho}(\mathcal{B}^*) + \delta_\Upsilon^{-1}(\mathcal{B}^*)$$

Obviously, removing a slice does not increase the spatial coverage ratios, i.e., $\delta_\Upsilon^{-1}(\mathcal{B}^* - \mathcal{B}^*(a,i)) \geq \delta_\Upsilon^{-1}(\mathcal{B}^*)$. Then, we can derive that $\rho(\mathcal{B}^* - \mathcal{B}^*(a,i)) \leq \rho(\mathcal{B}^*)$. By substituting Equation 1 in, we have that $\frac{M_{\mathcal{B}^*} - M_{\mathcal{B}^*(a,i)}}{(D_{\mathcal{B}^*}-1)/N} \leq \frac{M_{\mathcal{B}^*}}{D_{\mathcal{B}^*}/N}$ and derive that

$$M_{\mathcal{B}^*(a,i)} \geq \rho(\mathcal{B}^*)/N. \tag{12}$$

In each removing operation, we have

$$\rho(\hat{\mathcal{B}}) \geq \rho(\mathcal{B}_3) \geq \rho(\mathcal{B}). \tag{13}$$

Since $|\mathcal{B}(i)| \geq \frac{1}{N} D_{\mathcal{B}}$, we can derive that

$$\rho(\mathcal{B}) = \frac{M_{\mathcal{B}}}{D_{\mathcal{B}}/N} \geq \frac{M_{\mathcal{B}}}{|\mathcal{B}(i)|}. \tag{14}$$

For the removed slice in removing operation $\mathcal{B}(a,i)$, we have

$$M_{\mathcal{B}(a,i)} \leq \frac{M_{\mathcal{B}}}{|\mathcal{B}(i)|}. \tag{15}$$

Since $\mathcal{B}^* \subseteq \mathcal{B}$, we have that

$$M_{\mathcal{B}(a,i)} \geq M_{\mathcal{B}^*(a,i)}. \tag{16}$$

Combining Equations 12–16, we can derive that

$$\rho(\hat{\mathcal{B}}) \geq \frac{1}{N}\rho(\mathcal{B}^*). \tag{17}$$

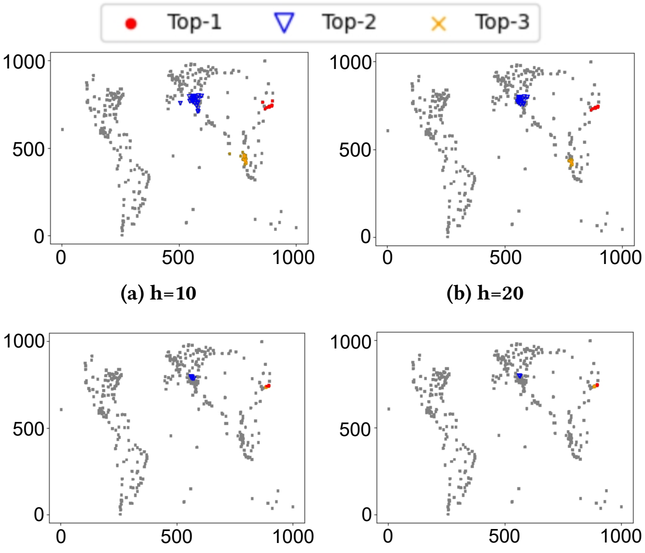
Since $\delta_\Upsilon^{-1}(\hat{\mathcal{B}}) \geq \frac{1}{N}$, and $0 \leq \delta_\Upsilon^{-1}(\mathcal{B}^*) \leq 1$, it can be inferred that

$$\delta_\Upsilon^{-1}(\hat{\mathcal{B}}) \geq \frac{1}{N}\delta_\Upsilon^{-1}(\mathcal{B}^*). \tag{18}$$

Combining Equations 17 and 18, we have $f(\hat{\mathcal{B}}) \geq \frac{1}{N} f(\mathcal{B}^*)$. Combined with lemma C.1, we can get that the RPA algorithm satisfies $f(\hat{\mathcal{B}}) \geq \frac{1}{N} f(\mathcal{B}^*)$. □

D More Experiment Results

Spatial compactness visualization of the EA algorithm when varying *h*. Figure 12 plots the spatial locations of the tuples in the top-3 blocks returned by the EA algorithm when varying h on dataset Foursquare. As h increases, the spatial coverage ratio becomes smaller. The reason is that the probability of combining neighbor cells become low, since the mass of small cells is probably low and cannot improve the score of the block to be expanded.



(a) h=10

(b) h=20

(c) h=50

(d) h=100

Figure 12: Spatial locations of the tuples in the top-3 blocks found by the EA algorithm when varying *h* on Foursquare.