CatchCore: Catching Hierarchical Dense Subtensor

Wenjie Feng (✉), Shenghua Liu, Huawei Shen, and Xueqi Cheng

CAS Key Laboratory of Network Data Science & Technology,
Institute of Computing Technology, Chinese Academy of Sciences, 100190
University of Chinese Academy of Sciences (UCAS), Beijing 100049, China.
fengwenjie@ict.ac.cn, liu.shengh@gmail.com, cxq@ict.ac.cn

Abstract. In this supplementary document, we provide additional examples, proofs, dataset description, and experimental results, all of which supplement the main paper.

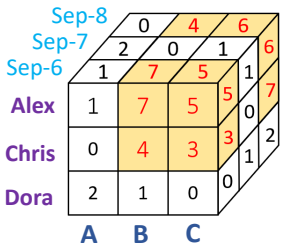
1 Tensor Example.

Example 1 (Review History). Assume a relation $\mathcal{R}$(user, item, date, #count), its three dimension attributes are { user, item, date }, the other one X = #count is the measure attribute. Each tuple $t = (u, i, d, c)$ in $\mathcal{R}$ indicates that user u visited item i on date d in total c times.

As a toy example of relation $\mathcal{R}$ in the Fig. 1(a), $\mathcal{R}_1$ = { Alex, Chris, Dora }, $\mathcal{R}_2$ = { A, B, C }, $\mathcal{R}_3$ = { Sep-6, Sep-7, Sep-8 }. The colored regions in Fig. 1(b) indicates a subtensor $\mathcal{B} \preccurlyeq \mathcal{R}$, which is composed of $\mathcal{B}_1$ = { Alex, Chris }, $\mathcal{B}_2$= { B, C }, $\mathcal{B}_3$ = { Sep-6, Sep-8 }. We use $\mathcal{R}$ and an indicator vectors collection $\mathbf{X}_{\mathcal{B}}$ to facilitate many computations w.r.t $\mathcal{B}$, and $\mathbf{X}_{\mathcal{B}} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}$ as Fig. 1(c) shown; taking the indicator vector $\boldsymbol{x}_1$ as an example, $\boldsymbol{x}_1 = [1, 1, 0]$ because $\mathcal{B}_1$ only contains "Alex" and "Chris" of $\mathcal{R}_1$ but no "Dora".

User	Item	Date	#Count
Alex	A	Sep-6	1
Alex	B	Sep-6	7
Chris	B	Sep-8	4
Dora	A	Sep-6	2
Alex	C	Sep-7	1
⋮	⋮	⋮	⋮

(a) 3 dimensions relation $\mathcal{R}$. (only a small part of colored)



(b) Tensor Representation of the relation $\mathcal{R}$.

User: $\mathbf{x}_1 = [1, 1, 0]$
Item: $\mathbf{x}_2 = [0, 1, 1]$
Date: $\mathbf{x}_3 = [1, 0, 1]$

(c) Indicator vectors for attributes of $\mathcal{B}$.

Fig. 1. Pictorial description of Example 1. (a) Relation $\mathcal{R}$(user, item, date, #count). (b) 3-way tensor representation of $\mathcal{R}$, the colored region indicates a subtensor $\mathcal{B}$ formed by some colored tuples in relation $\mathcal{R}$. (c) The indicator vectors collection representation for subtensor $\mathcal{B}$ can be denoted as $\mathbf{X}_{\mathcal{B}} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}$, and $V_{\mathcal{B}} = \prod_{i=1}^{3} ||\boldsymbol{x}_i||_1 = 2 * 2 * 2 = 8$.

Algorithm 1 OneWayOpt for one indicator vector updating

Input: (1) the indicator vector $\boldsymbol{x}_n$ to be updated for $\mathbf{X}^k$ (2) the initial update step size: α
(3) the ratio of decreasing the step size: $\beta \in (0, 1)$ (default 1/10)
(4) the linear search parameter: $\sigma \in (0, 1)$
Output: the new updated indicator vector $\tilde{\boldsymbol{x}}_n$.

1: compute the gradient $\nabla_{\boldsymbol{x}_n} f$ (as the statement in main paper)
2: compute the objective function $f(\cdot, \mathbf{X}^k, \cdot)$ with $\boldsymbol{x}_n$ (as the Eq. (6) shows)
3: initial $\alpha \leftarrow 1$ ▷ step size for updating
4: **while not** satisfy the Armijo's condition **do**
5: $\tilde{\boldsymbol{x}}_n = P(\boldsymbol{x}_n - \alpha \nabla_{\boldsymbol{x}_n} f)$ ▷ the new $\boldsymbol{x}_n$
6: compute the objective function $f(\cdot, \mathbf{X}^k_{\boldsymbol{x}_n \to \tilde{\boldsymbol{x}}_n}, \cdot)$ with $\tilde{\boldsymbol{x}}_n$
7: adjust the step size α with β based on the condition in Eq. (1).
8: **if** the condition in Eq. (1) is not satisfied **then**
9: $\alpha \leftarrow \beta \cdot \alpha$ ▷ **decrease** α
10: **else**
11: $\alpha \leftarrow \alpha / \beta$ ▷ **increase** α
12: **return** the final $\tilde{\boldsymbol{x}}_n$.

2 Indicator Vector Updating.

Here we introduce the OneWayOpt, which is described in Algorithm 1, is used to update any indicator vector $\boldsymbol{x}_n$ for $\mathbf{X}^k$ ($k \in \lfloor K \rceil$).

The condition for searching a good step size α with an Armijo's rule line search is

$$f(\cdot, \mathbf{X}^k_{\boldsymbol{x}_n \to \tilde{\boldsymbol{x}}_n}, \cdot) - f(\cdot, \mathbf{X}^k, \cdot) \leq \sigma (\nabla_{\boldsymbol{x}_n} f)^T (\tilde{\boldsymbol{x}}_n - \boldsymbol{x}_n). \tag{1}$$

Searching step size α is the most time consuming process, we use a more flexible line search [3] method to decrease the number of checks for the stop criterion Eq. (1). Specifically, let α denote the step size for updating $\boldsymbol{x}_n$, and we assume that it is similar to the step size for updating $\tilde{\boldsymbol{x}}_n$. Then, to update $\tilde{\boldsymbol{x}}_n$, we use α as the initial guess and either increase or decrease it with a scale factor β ($0 < \beta < 1$) until find the best step size satisfying Eq. (1). The core steps for efficiently searching α correspond to the Line 8-11, and $\beta = 1/10$ is chosen in our paper.

3 Proofs of Convergence and Complexity

3.1 Proofs of the Convergence (Lemma 1).

Proof (Convergence). Considering the subtensor detection as Eq. (3) defined (in main paper), the alternative updating steps in CatchCore come down to the block nonlinear Gauss-Seidel method.

Based on the convergence conclusion [2] for quasi-convex objective function, the indicator vectors in each dimension are bounded with a closed convex sets, the Armijo-type line search (LS) adopted in Algorithm 1 generates a sequence of points for any dimension, and there exits at least one limit point $\tilde{\boldsymbol{x}}$, then we have $(\nabla_{\boldsymbol{x}} f)^T (\tilde{\boldsymbol{x}} - \boldsymbol{x}) \geq 0$, i.e., $\tilde{\boldsymbol{x}}$ is a critical (stationary) point. This convergence result is still satisfied without any convexity assumption on the objective function.

Table 1. Accuracy of hierarchical subtensors detection for Synthetic dataset.

K	Injected Densities	H1				H2				H3				H4			
		CC	D/M	CS	CPD	**CC**	D/M	CS	CPD	**CC**	D/M	CS	CPD	**CC**	D/M	CS	CPD
2	$0.01 + 0.001$	1	1	0.14	0.14	1	0.183	0.89	0.89								
	$0.1 + 0.01$	1	1	0.25	0.25	1	1	0.89	0.89		–						
	$0.25 + 0.1$	1	1	0.35	0.35	1	0.257	1	1						–		
3	$0.1 + 0.01 + 0.001$	1	1	0.17	0.17	1	1	0.20	0.20	1	0.321	0.74	0.87				
	$0.25 + 0.1 + 0.01$	1	0	0.98	0.98	1	1	0.20	0.20	1	1	0.85	0.98				
4	$0.25 + 0.1 + 0.01 + 0.001$	1	0	0.96	0.96	1	1	0.51	0.51	1	1	0.19	0.19	1	0.359	0.79	0.95

* The algorithm abbreviations mean that CC: CatchCore, D: D-Cube, M: M-Zoom, CS: CrossSpot.
* The shape corresponding to different injected density is: 0.001: $200 \times 250 \times 100$, 0.01: $200 \times 100 \times 60$, 0.1:$80 \times 80 \times 30$, 0.25: $50 \times 60 \times 10$.

To detect hierarchical subtensors, CatchCore updates indicator vectors hierarchy-by-hierarchy, the convergence property is maintained since the subproblem for detecting subtensor in one hierarchy is same as the Eq. (3). Therefore, CatchCore algorithm converges to a critical point for each limit point. ■

3.2 Proofs of Time Complexity (Theorem 1).

Proof (Worst-case Time Complexity). Regarding the computational complexity for $M_{\mathbf{X}^k}$ and $\nabla_{\boldsymbol{x}_n} f$ in OneWayOpt Algorithm 1, they rely on the full-mode product $\bar{\times}$ or $\bar{\times}_{(-n)}$, which takes at most $O(nnz(\mathcal{R}))$ by conducting elementwise n-mode product for tensor and all vectors. Computing the norm of indicator vectors takes $O(\sum_{n=1}^{N} |\mathcal{R}_n|)$, which is included in the objective function f, vector gradient of f, and stop-criteria. Thus, for t_{als} searching iterations, the time complexity of OneWayOpt is $O(t_{\text{als}} \cdot (nnz(\mathcal{R}) + c \cdot D_{\mathcal{R}}))$.

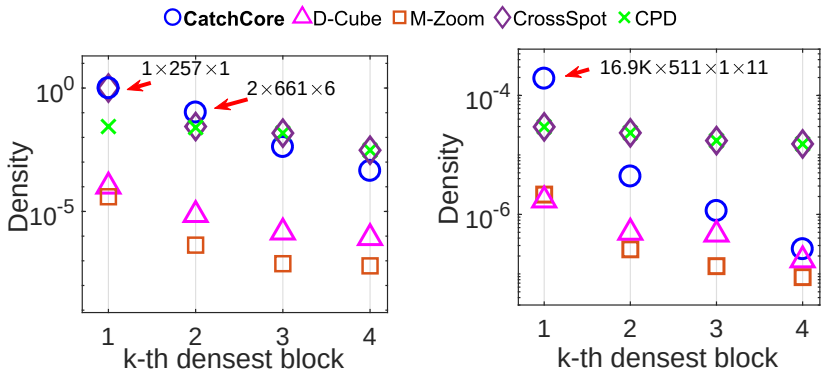
The alternative updating for all variables in CatchCore Algorithm runs $t_{\max}$ times at most, So there will be $K \cdot N \cdot t_{\max}$ iterations to call OneWayOpt in total. To select the significant subtensors, all indicator vectors in each of the K hierarchies will be check at most one time, for a total of $O(K \cdot \sum_{n=1}^{N} |\mathcal{R}_n|)$ times. Hence, the worst-case time complexity of CatchCore algorithm is $O(K \cdot N \cdot t_{\max} \cdot t_{\text{als}} \cdot (nnz(\mathcal{R}) + c \cdot D_{\mathcal{R}}))$. ■

3.3 Proofs of Space Complexity (Theorem 2).

Proof (Memory Requirements). CatchCore stores the tensor $\mathcal{R}$ in sparse format, the indicator vectors and gradient vectors when compute $M_{\mathbf{X}^k}$ and $\nabla_{\boldsymbol{x}_n^k} f$ for indicator vectors collection $\mathbf{X}^k$. So, the memory requirements is $O(nnz(\mathcal{R}) + 2K \cdot D_{\mathcal{R}})$. ■

4 Description of AirForce Dataset.

The attributes in AirForce [1] dataset are as follows:
- *protocol*: type of protocol (udp, tcp, icmp).
- *service*: type of network service on destination (e.g. http, telnet, etc.).



(a) StackOverflow dataset (b) Android dataset

Fig. 2. The subtensors detected by CatchCore achieve higher densities. In each plot, points represent the density of k-th of top 4 blocks found by the methods. And the size of some blocks are labeled in text. CatchCore catches suspicious patterns for the datasets.

- *src_types*: amount of data bytes from source to destination.
- *dst_types*: amount of data bytes from destination to source.
- *flag*: normal or error status of each connection.
- *count*: number of connections to the same host as the current connection in the past two seconds.
- *srv_count*: number of connections to the same service as the current connection in the past two seconds.
- *#connections*: number of connections with the corresponding dimension attribute values.

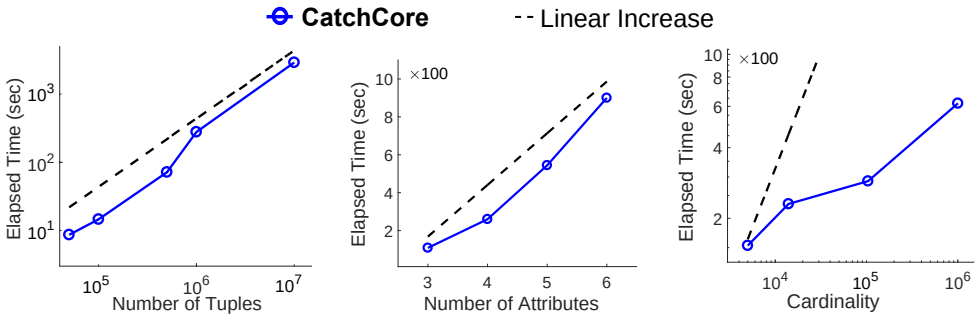
5 Additional Experiments.

5.1 Injected HDS-tensors for the synthetic dataset.

We inject K subtensors with different size and density into synthetic tensor $\mathcal{R}$ in a hierarchical manner. Table 1 lists the result, where H1 is the densest or the first subtensor detected by methods, and the density (order) decreases (increases) from H1 to H4, and the information of injected blocks is listed at the bottom. CatchCore can successfully detect all injected subtensors in various hierarchies, size, and density diversity. D-Cube and M-Zoom have the same performance, both of which fail to accurately detect the highest or lowest density blocks; CrossSpot and CPD also can not perfectly detect injected hierarchical blocks, especially for the densest block in $K = 2$ and middle-dense blocks for $K = 3$ and 4.

5.2 Patterns in real-world dataset.

The Fig. 2 shows the top 4 dense subtensors detected by different methods for StackOverflow and Android datasets. CatchCore detects higher-density dense blocks, which



(a) Number of Tuples ($nnz(\mathcal{R})$) (b) Number of Attributes (N) (c) Cardinalty of Attributes ($V_{\mathcal{R}}$)

Fig. 3. CatchCore is scalable. (a) (b) CatchCore scales linearly with the number of tuples and the number of attributes of $\mathcal{R}$. (c) CatchCore scales sub-linearly with the cardinalities of attributes of $\mathcal{R}$.

may contain more anomalous patterns than others. For the StackOverflow dataset, the user in the 1-st dense block marked 257 posts as favorites within an hour, which may be a bot activity; and the 2-nd block also shows suspicious dense pattern. For the Android dataset, the density of the 1-st dense subtensor detected by CatchCore is more 6 times than the results of other competitors.

5.3 Algorithm Scalability.

Fig. 3 shows that CatchCore scales (sub-) linearly with all aspects of the tensor, i.e, the number of tuples, the number of dimension attributes, and the cardinality of dimension attributes. Specifically, we measured the running time of CatchCore for a synthetic tensor $\mathcal{R}$ by changing one factor at a time. For the scalability of #tuples, $\mathcal{R}$ has three attributes each of whose cardinality is $10K$, with different density varying from $5 \cdot 10^{-8}$ to 10^{-5}; for the #attributes, we changed the dimension from 3 to 6 for $\mathcal{R}$ with $1M$ tuples and same cardinality; for the cardinality, the $D_{\mathcal{R}}$ of a 3-way tensor $\mathcal{R}$ with $1M$ tuples vary from $5K$ to 10^6. The result is consistent with the theoretical analysis in Theorem 1.

References

1. Kdd cup 1999 data (1999), `https://kdd.ics.uci.edu/databases/kddcup99/`
2. Grippo, L., Sciandrone, M.: On the convergence of the block nonlinear gauss-seidel method under convex constraints. Oper. Res. Lett. (2000)
3. Lin, C.J., Moré, J.J.: Newton's method for large bound-constrained optimization problems. SIAM J. on Optimization (1999)