BERKELEY LAB

U.S. DEPARTMENT OF ENERGY
Office of Science

Fast Exact Leverage Score Sampling from Khatri-Rao Products with Applications to Tensor Decomposition

Vivek Bharadwaj [1], Osman Asif Malik [2], Riley Murray [3,2,1], Laura Grigori [4], Aydın Buluç [2,1], James Demmel [1]

[1] Electrical Engineering and Computer Science Department, UC Berkeley

[2] Computational Research Division, Lawrence Berkeley National Lab

[3] International Computer Science Institute

[4] Sorbonne Université, Inria, CNRS, Université de Paris, Laboratoire Jacques-Louis Lions

Introduction

The Khatri-Rao Product

- The Khatri-Rao product (KRP, denoted $\odot$) is the column-wise Kronecker product of two matrices:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \odot \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} aw & bx \\ cw & dx \\ ay & bz \\ cy & dz \end{bmatrix}$$

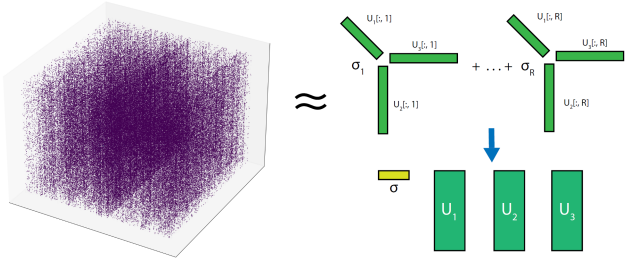
- Our goal: efficiently solve an overdetermined linear least-squares problem

$$\min_X \|AX - B\|_F$$

where $A = U_1 \odot \ldots \odot U_N$ with $U_j \in \mathbb{R}^{|I_j| \times R}$.

Motivation

This least-squares problem is the computational bottleneck in alternating least-squares Candecomp / PARAFAC (CP) decomposition.



Focus on large sparse tensors (mode sizes in the millions) and moderate decomposition rank $R \approx 10^2$. Assume $|I_j| = I$ for all j and $I \geq R$.

Randomized Least-Squares

- Well-studied approach: apply sketching operator S to both A and B, solve reduced problem

$$\min_{\tilde{X}} \left\| SA\tilde{X} - SB \right\|_F$$

- Want an (ε, δ) guarantee on solution quality: with high probability $(1 - \delta)$,

$$\left\| A\tilde{X} - B \right\|_F \leq (1 + \varepsilon) \min_X \|AX - B\|$$

- Restrict S to be a *sampling* matrix: selects and reweights rows from A and B. **How do we downsample a Khatri-Rao product accurately and efficently?**

Our Contributions

- We design a sampling data structure for the Khatri-Rao product requiring
  - Persistent space overhead at most the size of the input
  - Runtime **logarithmic** in the height of the Khatri-Rao product and quadratic in R to draw a single sample from the KRP, after moderate one-time costs
  - Only $O(R/(\varepsilon\delta))$ samples to achieve the (ε, δ) guarantee (ignoring $\log R$ factors)
- Yields the **STS-CP** algorithm: achieves lower asymptotic runtime for randomized CP decomposition than recent SOTA methods
- STS-CP achieves higher accuracy **and** faster progress to solution on sparse tensors with billions of nonzeros

Complexity Comparison

Method	Complexity per ALS Round
CP-ALS	$N(N+I)I^{N-1}R$
CP-ARLS-LEV	$N(R+I)R^N/(\varepsilon\delta)$
TNS-CP	$N^3IR^3/(\varepsilon\delta)$
Gaussian Tensor Network Embedding	$N^2(N^{1.5}R^{3.5}/\varepsilon^3 + IR^2)/\varepsilon^2$
STS-CP (ours)	$N(NR^3\log I + IR^2)/(\varepsilon\delta)$

Factors involving $\log R$ and $\log(1/\delta)$ omitted.

Prior Work and Main Result

Leverage Score Sampling

We will sample rows i.i.d. from A according to the *leverage score distribution* on its rows. Leverage score ℓ_i of row i is

$$\ell_i = A\,[i,:]\,(A^\top A)^+ A\,[i,:]^\top$$

Theorem (Leverage Score Sampling Guarantees)

Suppose $S \in \mathbb{R}^{J\times I}$ is a leverage-score sampling matrix for $A \in \mathbb{R}^{I\times R}$, and define

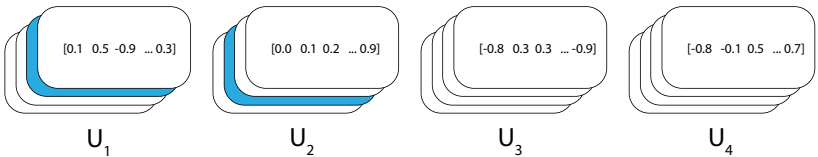
$$\tilde{X} := \arg\min_{\tilde{X}} \left\| SA\tilde{X} - SB \right\|_F$$

If $J \gtrsim R\max(\log(R/\delta), 1/(\varepsilon\delta))$, then with probability at least $1-\delta$,

$$\left\| A\tilde{X} - B \right\|_F \leq (1+\varepsilon) \min_X \|AX - B\|_F$$

Leverage Score Sampling

- For $I = 10^7, N = 3$, matrix A has 10^{21} rows. Far too expensive to compute all leverage scores - can't even index rows with 64-bit integers.
- Instead: draw a row from each of $U_1, ..., U_N$, return their Hadamard product.



- Let $\hat{s}_j$ be a random variable for the row index drawn from U_j. Assume $(\hat{s}_1, ..., \hat{s}_N)$ jointly follows the leverage score distribution on $U_1 \odot ... \odot U_N$.

A Problem of Dependence

- **Problem:** Variables are not independent! In general,

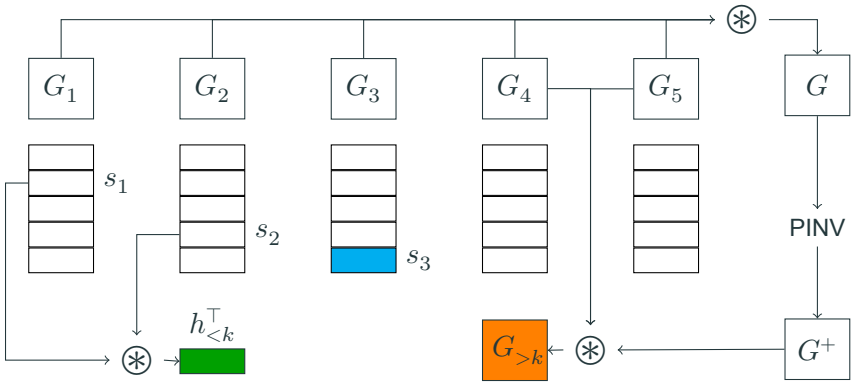
$$p(\hat{s}_2 = s_2) \neq p(\hat{s}_2 = s_2 \mid \hat{s}_1 = s_1)$$

- How do we deal with the dependence? Several approaches.

Algorithm	Preprocessing	Sampling Time	J Required
Precompute all	$\Omega(I^N)$	$O(JN)$	$O(R/(\varepsilon\delta))$
Malik et al.	$O(NIR^2)$	$O(JNR^2I)$	$O(R/(\varepsilon\delta))$
Larsen & Kolda	$O(NIR^2)$	$O(JN)$	$O(R^N/(\varepsilon\delta))$
Our Algorithm	$O(NIR^2)$	$O(NR^3 + JNR^2 \log I)$	$O(R/(\varepsilon\delta))$

Outline of Sampling Procedure

The Conditional Distribution of $\hat{s}_k$



Theorem

$$p(\hat{s}_k = s_k \mid \hat{s}_{<k} = s_{<k}) \propto \langle h_{<k} h_{<k}^\top, U_k\left[s_k, :\right]^\top U_k\left[s_k, :\right], G_{>k} \rangle$$

Stage 1: Sample Eigenvector of $G_{>k}$

Compute symmetric eigendecomposition $G_{>k} = V\Lambda V^T$, break the conditional distribution into components:

$$1 = \sum_{s_k=1}^{I} p(\hat{s}_k = s_k \mid \hat{s}_{<k} = s_{<k})$$

$$= C^{-1}\langle h_{<k}h_{<k}^{\top}, G_k, G_{>k}\rangle = C^{-1}\Big\langle h_{<k}h_{<k}^{\top}, G_k, \lambda_1 \; + \ldots + \lambda_R \; \Big\rangle$$

Sample component in time $O(R^2 \log R)$

λ_u

Stage 2: Sample Row Index Based on Eigenvector

Break remaining sample space further into components:

$$C^{-1}\langle h_{<k}h_{<k}^{\top}, G_k, \lambda_u V[:,u]\, V[:,u]^{\top}\rangle$$

$$= C^{-1}\Big\langle h_{<k}h_{<k}^{\top}, \; + \ldots + \; , \lambda_u \; \Big\rangle$$

Sample row index
in time $O(R^2 \log(I/R))$

Sampling Time: $O\left(R^2 \log(I/R) + R^2 \log R\right) = O(R^2 \log I)$

Analysis of Sampler Design

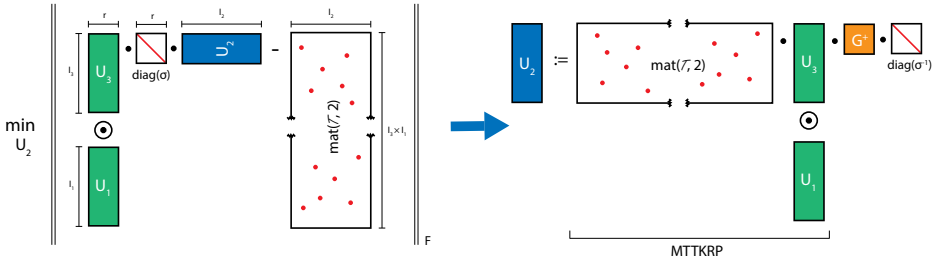
- First stage selects a one-dimensional subspace of $G_{>k}$.
- Second stage samples according to the squared-norms in a 1-dimensional inner product space, reducing time / space complexity.

- Without two-stage sampling design, would incur either
  - $O(R^3 + R^2 \log I)$ time per sample (ours: $O(R^2 \log I)$)
  - $O(NIR^2)$ space usage (ours: $O(NIR)$)
- Connections to the **Maximum Squared Inner Product Search** problem.

Application to ALS CP Decomposition

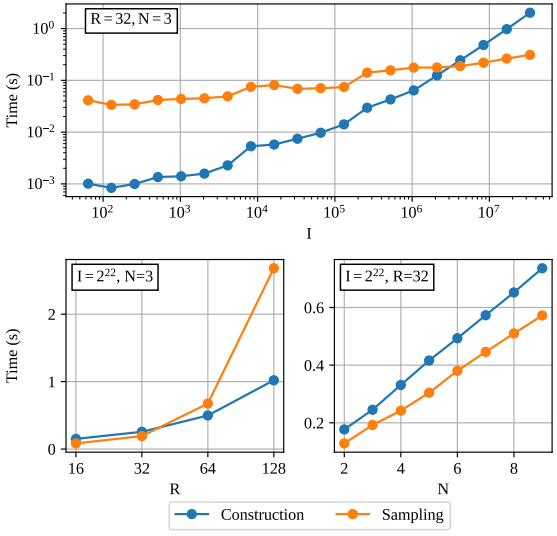
CP Decomposition: Represents an N-dimensional tensor $\mathcal{T}$ as a weighted sum of generalized outer products. Iteratively solve least-squares problems of the form

$$\min_{\hat{U}_j} \left\| \left[\bigodot_{k \neq j} U_k \right] \cdot \mathrm{diag}(\sigma) \cdot \hat{U}_j^\top - \mathrm{mat}(\mathcal{T}, j)^\top \right\|_F$$



Experiments

Runtime Benchmarks (LBNL Perlmutter CPU)

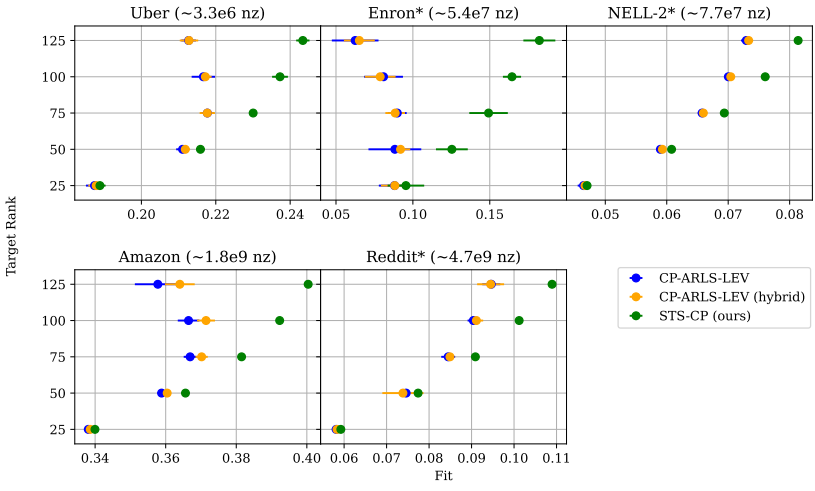


Sparse Tensor Decomposition

Tensor	Dimensions	Nonzeros
Uber Pickups	$183 \times 24 \times 1{,}140 \times 1{,}717$	3,309,490
Enron Emails	$6{,}066 \times 5{,}699 \times 244{,}268 \times 1{,}176$	54,202,099
NELL-2	$12{,}092 \times 9{,}184 \times 28{,}818$	76,879,419
Amazon Reviews	$4{,}821{,}207 \times 1{,}774{,}269 \times 1{,}805{,}187$	1,741,809,018
Reddit-2015	$8{,}211{,}298 \times 176{,}962 \times 8{,}116{,}559$	4,687,474,081

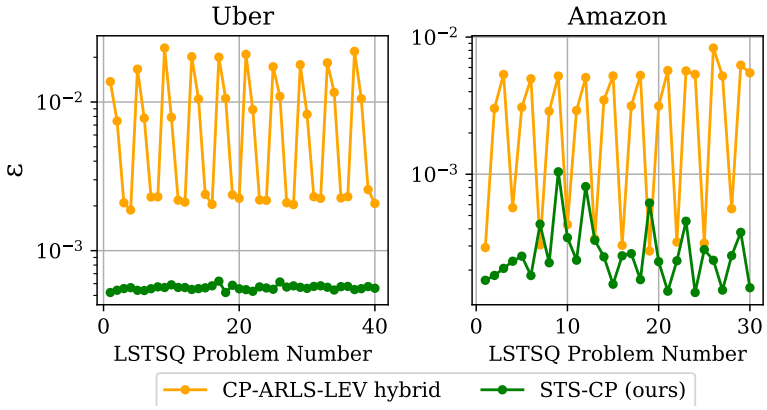
- Ran sparse CP decomposition on tensors from the FROSTT collection.
- Compared STS-CP against random and hybrid versions of CP-ARLS-LEV.
  - One of few randomized algorithms designed for sparse tensors.
  - For an N-dimensional tensor, sample complexity is $O(R^{N-1}/(\varepsilon\delta))$ per solve.

Accuracy Comparison for Fixed Sample Count

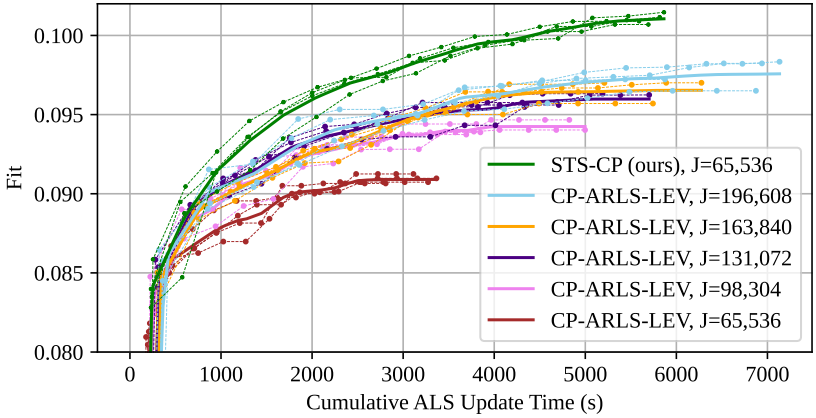


ALS Accuracy Comparison for $J = 2^{16}$ samples.

Accuracy on Individual Least-Squares Problems



Fit vs. ALS Update Time



Fit vs. ALS Update Time, Reddit Tensor, $R = 100$.

Future Work

- Can the quadratic-in-R sampling cost be reduced?
- Can we apply to other tensor formats (e.g. MPS / tensor train)?
- Work in progress: distributed-memory sampling.

Thank You! Read the preprint, and try out the code.

https://arxiv.org/abs/2301.12584

https://github.com/vbharadwaj-bk/fast_tensor_leverage

Backup Slides

Comparison to Countsketch

Countsketch Matrix:

$$\begin{bmatrix} 0 & 0 & -1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

One nonzero per **column**. Every row in my input is added / subtracted to exactly one row of my output.

Sampling Matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

One nonzero per **row**. Every row in my output is a copy-pasted row from my input.

Page 19, “Sketching as a Tool for Numerical Linear Algebra”, Woodruff.

Approaches to KRP Leverage Score Sampling

Approach 1: Exhaustive Precomputation

- Only a finite number of values for $\hat{s}_1$. Precompute and store all possible conditional distributions for $\hat{s}_2$, and similarly for $\hat{s}_3, \hat{s}_4 ...$
- Preprocessing time is $\Omega\left(I^N\right)$, not viable for large I.

Preprocessing Time	Time for J Samples	# Samples Required
$\Omega(I^N)$	$O(JN)$	$O(R/(\varepsilon\delta))$

Approach 2: Ignore the Dependence

- Sample **independently** from $U_1, ..., U_N$ based on the leverage scores of each factor matrix. Approach used by Cheng et al., Larsen and Kolda.
- No longer sampling from the exact leverage score distribution, so require $O(R^N/(\varepsilon\delta))$ samples to achieve the (ε, δ) guarantee.
- Efficient if R, N low enough. Can easily update if one matrix U_j changes.

Preprocessing Time	Time for J Samples	# Samples Required
$O(NIR^2)$	$O(JN)$	$O(R^N/(\varepsilon\delta))$

Approach 3: Compute Full Conditional Distribution for each Sample

- Compute the full conditional distribution $p(\hat{s}_2 = s_2 \mid \hat{s}_1 = s_1)$ for each draw *during* sampling. Approach used by Malik et al. (TNS-CP).
- Costs $O(IR^2)$ per matrix U_j **per sample**.
- Works well if I is low enough (many dense tensor applications), but performance degrades for $I \geq 10^3$.

Preprocessing Time	Time for J Samples	# Samples Required
$O(NIR^2)$	$O(JNR^2I)$	$O(R/(\varepsilon\delta))$

Approach 4: Segment Tree Sampling (Ours)

- View the conditional distribution as a *mixture* of several components.
- After preprocessing, sample a component of the mixture via binary search **without** computing all values from the conditional distribution.
- For $R \approx 10^2$, we achieve a sampling time that is practical for sparse tensor decomposition with mode sizes in the tens of millions.

Preprocessing Time	Time for J Samples	# Samples Required
$O(NIR^2)$	$O(NR^3 + JNR^2 \log I)$	$O(R/(\varepsilon\delta))$

Main Theorem

Theorem

Given matrices $U_1, ..., U_N$, $U_j \in \mathbb{R}^{I \times R}$ $\forall j$, there exists a data structure with the following properties:

1. *Its construction time is $O(NIR^2)$, and its storage cost is $O(NIR)$. If matrix U_j changes, it can be updated in time $O(IR^2)$*
2. *Using $O(R^3)$ scratch space, it can draw J samples from the KRP $U_1 \odot ... \odot U_N$ according the leverage score distribution on its rows in time*

$$O\left(NR^3 + JNR^2 \log I\right).$$

It can also draw samples from the KRP of all matrices excluding one.

Complete Proof of Main Result

Part 1: Segment Tree Sampling

- Given probability distribution $q_1, ..., q_I$, how do you sample from it efficiently?

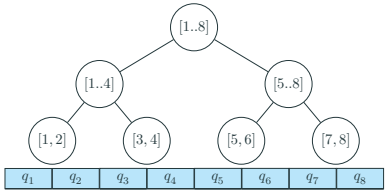
- Simple Algorithm: Binary-Search Inversion Sampling
  1. Divide $[0, 1]$ into I bins. Bin i has endpoints $\left[\sum_{j=0}^{i-1} q_j, \sum_{j=0}^{i} q_j\right)$.
  2. For each sample, draw a real number D uniformly from $[0, 1]$. Binary search on the list of endpoints to find the containing bin, return its index.

- Preprocessing cost: $O(I)$ (prefix sum). Per-sample cost: $O(\log I)$ (binary search).

Part 1: Segment Tree Sampling

Modify the previous procedure as follows:

- Binary search until remaining interval has at most F bins, iterate through what remains to find bin containing D.
- View as a traversal of a *segment tree* $T_{I,F}$ from root to a leaf. Each node equipped with segment $S(v) \subseteq [1..I]$.
- **Key**: At internal nodes, don't need individual probabilities q_j - only their **sum**.



Part 1: Segment Tree Sampling

Define functions $\tilde{m} : T_{n,F} \to \mathbb{R}_+$ and $\tilde{q} : T_{n,F} \to \mathbb{R}_+^F$. Use these functions to branch at internal nodes and search the leaf intervals $S(v)$.

Proposition

If $\tilde{m}(v) = \sum_{i \in S(v)} q_i$ and $\tilde{q}(v) = \{q_i \mid i \in S(v)\}$ at each leaf, STSample returns index i with probability q_i.

Algorithm 1 STSample($T_{I,F}$, $\tilde{m}(\cdot), \tilde{q}(\cdot)$)

$c := \text{root}(T_{I,F}), \text{low} = 0.0, \text{high} = 1.0$
Sample $D \sim \text{Uniform}(0.0, 1.0)$
while $c \notin \text{leaves}(T_{I,F})$ **do**
  $\text{cutoff} := \text{low} + \tilde{m}(L(c))$
  if $\text{cutoff} \geq D$ **then**
    $c := L(c), \text{high} := \text{cutoff}$
  else
    $c := R(c), \text{low} := \text{cutoff}$
return $S_0(v) + \text{argmin}_{i \geq 0} \left(\text{low} + \sum_{j=1}^{i} \tilde{q}(c)[j] < D \right)$

Part 1: Segment Tree Sampling

- If $\tilde{m}$ runs in time τ_1 per call and $\tilde{q}$ runs in time $\tau_2(F)$ per call, the complexity of STSample is

$$O(\tau_1 \log\lceil I/F \rceil + \tau_2(F))$$

- If we have efficient functions to compute $\tilde{m}$ and $\tilde{q}$, we can avoid a linear factor I when drawing each sample.

Part 2: A Simpler Row Sampling Problem

- Suppose we wish to sample J rows from a matrix $A \in \mathbb{R}^{I \times R}$. Let $\hat{s}$ be the RV for a sample index, $h \in \mathbb{R}^R, Y \in \mathbb{R}^{R \times R}$ be a vector and a p.s.d. matrix.
- Impose

$$p(\hat{s} = s \mid h, U, Y) := q_{h,U,Y}[s] := C^{-1} \langle hh^\top, U[s,:]^\top U[s,:], Y \rangle$$

  Here, $\langle \cdot, \cdot, \cdot \rangle$ means "multiply three matrices elementwise, take sum of all entries in product" (generalized inner product).
- The twist: Y is the same for all row samples, but h is potentially *unique* for each one.

Part 2: A Simpler Row Sampling Problem

- Solution: initialize a segment tree $T_{I,F}$. For any segment $S(v)$ associated with a node v, sum both sides:

$$\begin{aligned}\sum_{s \in S(v)} p(\hat{s} = s \mid h, U, Y) &= \sum_{s \in S(v)} C^{-1} \langle hh^\top, U[s,:]^\top U[s,:], Y \rangle \\ &= C^{-1} \langle hh^\top, \sum_{s \in S(v)} U[s,:]^\top U[s,:], Y \rangle \\ &:= C^{-1} \langle hh^\top, G^v, Y \rangle\end{aligned}$$

- If G^v is precomputed for each node $v \in T_{I,F}$, last line of equation above computable in $O(R^2)$ time. Produces efficient function $\tilde{m}$ for STSample.

Part 2: A Simpler Row Sampling Problem

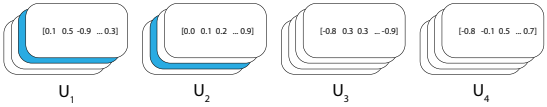
Lemma (Efficient Row Sampler)

There is a data structure parameterized by integer F that, given a matrix A and a p.s.d. matrix Y, satisfies the following:

- *Has construction time $O(IR^2)$ and space complexity $O(R^2\lceil I/F\rceil)$.*
- *After construction, produces sample from $q_{h,U,Y}$ in time $O(R^2\log\lceil I/F\rceil + FR^2)$ for any vector h.*
- *If Y is a matrix of all ones, the time per sample drops to $O(R^2\log\lceil I/F\rceil + FR)$.*

Main Proof Idea: Precompute matrices G^v in construction phase, call STSample during the sampling phase.

Part 3: Assembling the Khatri-Rao Product Sampler



- Let $A = U_1 \odot ... \odot U_N$. Let $G_k = U_k^\top U_k$, $G = \circledast_{k=1}^{N} G_k$.
- Suppose we have sampled $\hat{s}_1 = s_1, ..., \hat{s}_{k-1} = s_{k-1}$. Define

$$h_{<k} = \overset{k-1}{\underset{i=1}{\circledast}} U_i\,[s_i, :]$$

$$G_{>k} = G^{+} \circledast \overset{N}{\underset{i=k+1}{\circledast}} G_i$$

Part 3: Assembling the Khatri-Rao Product Sampler

- What is the distribution of $\hat{s}_k$ conditioned on prior draws?

Theorem (Heavily Adapted Version of Malik 2022)

$$p(\hat{s}_k = s_k \mid \hat{s}_1 = s_1 ... \hat{s}_{k-1} = s_{k-1}) = q_{h_{<k}, U_k, G_{>k}}[s_k]$$

- Matches our lemma! Use the data structure that we developed earlier.

Part 3: Assembling the Khatri-Rao Product Sampler

$$p(\hat{s}_k = s_k \mid \hat{s}_1 = s_1 ... \hat{s}_{k-1} = s_{k-1}) = q_{h_{<k}, U_k, G_{>k}}[s_k]$$

- If lemma applied directly to conditional distribution $p(\hat{s}_k = s_k \mid \hat{s}_{<k} = s_{<k})$, you either incur
  - $O(IR^2)$ space complexity for $F = 1$
  - $O(NR^3 \log I)$ time per sample for $F = R$
- To fix: observe that $G_{>k}$ is p.s.d., identical for all samples. Take its eigendecomposition

$$G_{>k} = V \Lambda V^\top$$

Part 3: Assembling the Khatri-Rao Product Sampler

- Define matrix $W \in \mathbb{R}^{I \times R}$ elementwise by

$$W[t,u] := \langle h_{<k} h_{<k}^\top, U_k[t,:]^\top U_k[t,:], V[:,u] V[:,u]^\top \rangle$$

- After some manipulation, we can write

$$q_{h_{<k}, U_k, G_{>k}} = \sum_{u=1}^{R} e[u] \frac{W[:,u]}{\|W[:,u]\|_1},$$

where $e[u] = \lambda_u \|W[:,u]_1\|$. Since $\lambda_u \geq 0$, this is a *mixture* distribution. Sample in two steps:

- Choose a component u according to weight vector e
- Sample an index in I_k according to $W[:,u]$.

Part 3: Assembling the Khatri-Rao Product Sampler

- Let $\overline{e}$ be a normalized version of e. Manipulation yields

$$\overline{e} = q_{h_{<k}, \sqrt{\Lambda} V^\top, G_k}$$

  Use our lemma with $F = 1$ to efficiently select a component.

- Suppose we select component $\hat{u} = u$. Then the row index $\hat{t}$ drawn according to distribution $\overline{W[:, u]}$ is distributed as

$$p(\hat{t} = t \mid \hat{u} = u) = q_{h_{<k} \circledast V[:,u], U_k, [1]}[t].$$

  Use our lemma again with $F = R$ to draw a row index.

Part 3: Assembling the Khatri-Rao Product Sampler

- First sampling phase reduces the gram matrix $G_{>k}$ to an outer product from one of its eigenvectors. Reduces runtime for procedure $\tilde{q}$ in the second sampling phase

- Second sampling phase can choose $F = R$ to control space complexity.

Application to ALS CP Decomposition

Corollary (STS-CP)

Suppose $\mathcal{T}$ is dense, and suppose we solve each least-squares problem in ALS with a randomized method. Leverage score sampling using our data structure achieves the (ε, δ) guarantee with $O(R/(\varepsilon\delta))$ samples. The overall complexity is

$$O\left(\frac{\#it \cdot N}{\varepsilon\delta}\left(NR^3 \log I + IR^2\right)\right)$$

For sparse tensors, STS-CP **preserves tensor sparsity** in the downsampled least-squares problem.

Additional Results

Reference Implementation

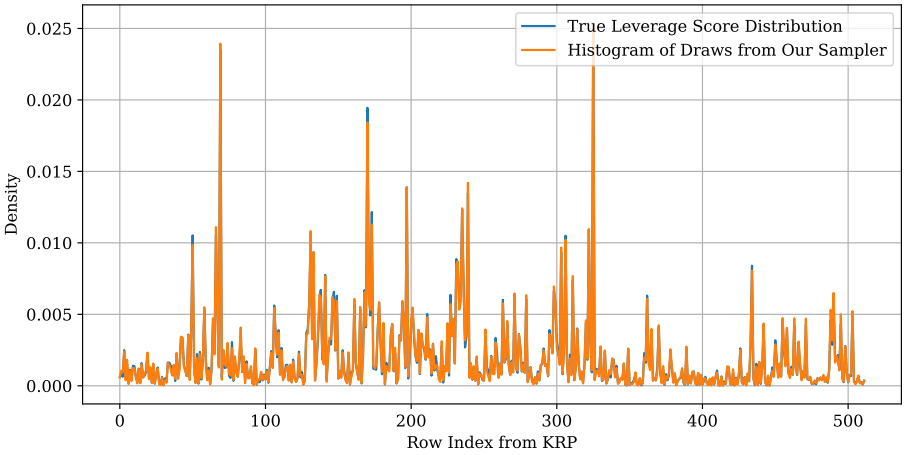
Algorithm 2 Snippet of KRPSample Pseudocode

1: ...
2: **for** $d = 1...J$ **do**
3: $h = [1, ..., 1]^\top$
4: **for** $k \neq j$ **do**
5: $\hat{u}_k := \text{RowSample}(E_k, h)$
6: $\hat{t}_k := \text{RowSample}(Z_k, h \circledast (V_k[:, \hat{u}_k]))$
7: $h *= U_k[\hat{t}_k, :]$
8: $s_d = (\hat{t}_k)_{k \neq j}$
9: **return** $s_1, ..., s_J$

Python Reference Implementation

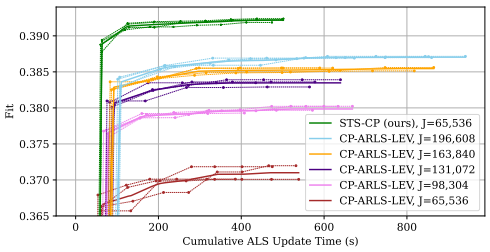
```
...
samples = []
for _ in range(J):
    h = np.ones(self.R)
    sample = []
    for k in range(self.N):
        if k == j:
            continue
        u_k = E_samplers[k].RowSample(h)
        h_scl = h * Lambda_VT[k][u_k]
        t_k = self.Z_samplers[k].RowSample(h_scl)
        h *= self.U[k][t_k, :]
        sample.append(t_k)
    samples.append(sample)
return samples
```

Verifying Our Sampler's Output

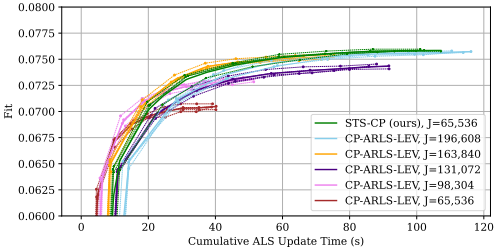


Distribution Comparison for $U_1 \odot U_2 \odot U_3$, $U_j \in \mathbb{R}^{8\times 8}$ initialized i.i.d. Gaussian.

Fit vs. ALS Update Time



(a) Amazon



(b) NELL-2

Fit vs. ALS Update Time, $R = 100$.

Discussion

- Theoretically superior sample complexity of STS-CP verified through experiments.
- Higher accuracy per least-squares solve for STS-CP translates to better final tensor fit.
- The runtime overhead of STS-CP is justified on sparse tensors with billions of nonzeros.
- On smaller tensors, STS-CP may benefit from dynamically adapting the sample count J.