

On Computing and the Complexity of Computing Higher-Order U-Statistics, Exactly

陈星宇

上海交通大学

数学科学学院

Email: xingyuchen0714@sjtu.edu.cn

Website: <https://cxy0714.github.io/>

上海交通大学首届全国高层次博士生学术论坛

2025 年 11 月 17 日

Coauthors



刘林
上海交通大学
自然科学研究院



张瑞琦
华东师范大学
数学科学学院

Outline

Introduction

On Computing U-Statistics

On Complexity of Computing U-Statistics

Applications

Conclusions

U/V-Statistics

U-statistics are generally used to construct unbiased estimators of population parameters.

- ▶ Given a kernel function $h(x_1, \dots, x_m) : \mathcal{X}^m \rightarrow \mathbb{R}$ and a sequence of samples X_1, \dots, X_n .
- ▶ The U-statistic takes the form

$$\mathbb{U}_{n,m}[h] = \sum_{1 \leq i_1 \neq i_2 \neq \dots \neq i_m \leq n} h(X_{i_1}, \dots, X_{i_m}).$$

- ▶ The V-statistic takes the form

$$\mathbb{V}_{n,m}[h] = \sum_{1 \leq i_1, i_2, \dots, i_m \leq n} h(X_{i_1}, \dots, X_{i_m}).$$

- ▶ Two problem in computation:

| Object | Time Complexity | Highly optimized library? |
|--|--|---------------------------|
| Matrix multiplication ($n \times n$) | $O(n^3)$, $O(n^{2.48})^1$, $O(n^{2.372})^2, \dots$ | Yes, BLAS, Eigen,... |
| m -th order U -statistic | (naively) $O(n^m)$ | No, but will use Einsum |

¹Strassen (1986)

²Williams et al. (2024)

Example 1: Dependence Measures

- ▶ High-order U-statistics are also used to estimate dependence measures, such as Distance Covariance (dCov^2) (Székely et al., 2007).
- ▶ dCov^2 can be represented as a 4-th order U-statistic (Yao et al., 2018):

$$\text{dCov}^2(X, Y) = \frac{(n-4)!}{n!} \sum_{i \neq j \neq q \neq r} a_{ij}b_{qr} + a_{ij}b_{ij} - a_{ij}b_{iq} - a_{ij}b_{jr}$$

where $a_{ij} = \|X_i - X_j\|_2$ and $b_{ij} = \|Y_i - Y_j\|_2$.

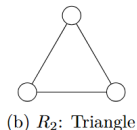
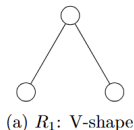
- ▶ The complexity will be $O(n^2)$ given the $n \times n$ matrices a_{ij}, b_{ij} .
- ▶ In Shao et al. (2025), randomized select $O(n^\alpha)$ pairs to approximate.

Table: Runtime (in seconds) comparison of various methods for computing dCov^2 . Experiments were run on Intel Xeon ICX Platinum 8358 CPUs (2.6GHz, 64 total cores) with memory of 512 GB. $n = 138$.

| u-stats | | Shao et al. (2025)'s MATLAB code | | | |
|-------------|----------|----------------------------------|------------------------------|------------------------------|-----------|
| No Parallel | Parallel | Randomized $\alpha = 1.5$ | Randomized $\alpha = 2.0$ | Randomized $\alpha = 2.5$ | Complete |
| 4.0928 | 0.1847 | 0.4211 | 4.5744 | 53.0265 | 3395.4001 |

Example 2: Motif Counts

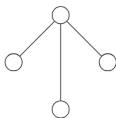
- ▶ Motif counts refer to the number of occurrences of small subgraphs (motifs) in a random graph.
- ▶ They can be used to test certain properties of the underlying random graphon ([Chatterjee et al., 2024](#)).
- ▶ The motif counts can also be written as a form similar to U-statistics.
- ▶ 3-node motifs,



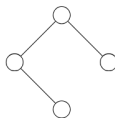
3-node motifs in a random graph

Example 2: Motif Counts

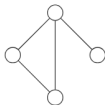
► 4-node motifs,



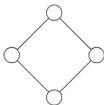
(c) R_3 : 3-star



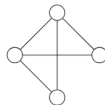
(d) R_4 : Fork



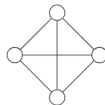
(e) R_5 : Tailed triangle



(f) R_6 : Square



(g) R_7 : House



(h) R_8 : 4-clique

4-node motifs in a random graph

Example 2: Motif Counts

► For example :

► V-shape

$$C(R_1) = \frac{1}{2} \sum_{i_1 \neq i_2 \neq i_3} A_{i_1 i_2} A_{i_2 i_3} B_{i_3 i_1}$$

► Triangle:

$$C(R_2) = \frac{1}{6} \sum_{i_1 \neq i_2 \neq i_3} A_{i_1 i_2} A_{i_2 i_3} A_{i_3 i_1}$$

► A is the adjacency matrix of the graph and $B = 1 - A$.



(a) R_1 : V-shape



(b) R_2 : Triangle

Example 3: U-Statistics in Causal Inference: HOIFs

- ▶ Higher-order influence functions (HOIFs) (Robins et al., 2008, 2016) are rate-optimal estimators for many causal parameters.
- ▶ HOIFs are high order U-statistics, whose order can be up to $m \sim \sqrt{\log n}$.
- ▶ For example, HOIF of the treatment-specific mean is the combination of functions like

$$\begin{aligned} h_m^{\text{HOIF}}(X_1, \dots, X_m) \\ &= [a_1 \phi(Z_1)^\top \phi(Z_2)] [\phi(Z_2)^\top \phi(Z_3)] \cdots [\phi(Z_{m-1})^\top \phi(Z_m) b_m] \\ &= f_1(X_1, X_2) f_2(X_2, X_3) \cdots f_{m-1}(X_{m-1}, X_m) \end{aligned}$$

- ▶ For complexity, given the $n \times n$ matrices $A_{ij}^{(k)} = f_k(X_i, X_j)$ for $k = 1, \dots, m-1$:

$$\text{Complexity}(\mathbb{U}_{n,m}(h_m^{\text{HOIF}})) = \begin{cases} O(n^2), & m \in \{2, 3\}, \\ O(n^3), & m \in \{4, 5, 6, 7\}, \\ O(n^4), & m \in \{8, 9, 10\}, \\ O(n^5), & m \in \{11, 12\}. \end{cases}$$

Outline

Introduction

On Computing U-Statistics

On Complexity of Computing U-Statistics

Applications

Conclusions

The answer: V-statistics can be computed by Einsum

- ▶ Einsum can be used to compute the V-statistic efficiently.
 - ▶ The Einsum operation mainly performs **unconstrained summation** over selected indices of input tensors.
 - ▶ Can call `numpy.einsum` or `pytorch.einsum` in practice.
 - ▶ `pytorch` provides parallel computing on CPU and GPU.
- ▶ Examples:

`Einsum('ij,jk->ik', A, B)`

$$\sum_j A_{ij} B_{jk} = D_{ik}$$

`Einsum('ijk->i', X)`

$$\sum_{j,k} X_{ijk} = D_i$$

`Einsum('ij,jk,kl->', A, B, C)`

$$\sum_{i,j,k} A_{ij} B_{jk} C_{kl} = D$$

- ▶ Then what's the exact formula of the $\mathbb{U} = \sum \mathbb{V}$?

Decomposition of U-statistics to V-statistics

- ▶ The exact formula can be written as following:
- ▶ The proof will use the **Möbius inversion** technique.

Lemma 1 (C., Zhang, Liu, 25)

Let \mathbb{X} be a non-empty set and $h : \mathbb{X}^m \rightarrow \mathbb{R}$ be a kernel function. Then for any $\mathbf{X} \in \mathbb{X}^n : n \geq m$,

$$\mathbb{U}[h] = \sum_{\pi \in \Pi_m} \mu_{\pi} \mathbb{V}[\pi](h),$$

where

$$\mu_{\pi} = (-1)^{(m-|\pi|)} \prod_{C \in \pi} (|C| - 1)!,$$

and Π_m denotes all partition of set $\{1, 2, \dots, m\}$.

Decomposition of U-statistics to V-statistics

- The definition of $\mathbb{V}[\pi](h)$ is as following:

$$\pi = \{\{1\}, \{2\}, \{3\}\}, \quad \mathbb{V}[\pi](h) = \sum_{i_1, i_2, i_3} h(X_{i_1}, X_{i_2}, X_{i_3}), \quad \mu_\pi = +1,$$

$$\pi = \{\{1, 2\}, \{3\}\}, \quad \mathbb{V}[\pi](h) = \sum_{i_1=i_2, i_3} h(X_{i_1}, X_{i_2}, X_{i_3}), \quad \mu_\pi = -1,$$

$$\pi = \{\{1, 3\}, \{2\}\}, \quad \mathbb{V}[\pi](h) = \sum_{i_1=i_3, i_2} h(X_{i_1}, X_{i_2}, X_{i_3}), \quad \mu_\pi = -1,$$

$$\pi = \{\{2, 3\}, \{1\}\}, \quad \mathbb{V}[\pi](h) = \sum_{i_2=i_3, i_1} h(X_{i_1}, X_{i_2}, X_{i_3}), \quad \mu_\pi = -1,$$

$$\pi = \{\{1, 2, 3\}\}, \quad \mathbb{V}[\pi](h) = \sum_{i_1=i_2=i_3} h(X_{i_1}, X_{i_2}, X_{i_3}), \quad \mu_\pi = +2.$$

- For $m = 3$:

$$\begin{aligned} \mathbb{U}[h] &= \sum_{i_1 \neq i_2 \neq i_3} h(X_{i_1}, X_{i_2}, X_{i_3}) \\ &= \left(\sum_{i_1, i_2, i_3} - \sum_{(i_1=i_2), i_3} - \sum_{(i_1=i_3), i_2} - \sum_{(i_2=i_3), i_1} + 2 \sum_{i_1=i_2=i_3} \right) h(X_{i_1}, X_{i_2}, X_{i_3}). \end{aligned}$$

Algorithm Framework

Basic Algorithm Framework

Input: Kernel function h , data samples X_1, \dots, X_n , order m

Output: U-statistic $\mathbb{U}[h]$

1. Initialize $\mathbb{U}[h] \leftarrow 0$
2. For each partition $\pi \in \Pi_m$:
 - 2.1 Compute coefficient μ_π
 - 2.2 Compute V-statistic $\mathbb{V}[\pi](h)$ via Einsum
 - 2.3 $\mathbb{U}[h] \leftarrow \mathbb{U}[h] + \mu_\pi \cdot \mathbb{V}[\pi](h)$
3. Return $\mathbb{U}[h]$

Key Assumption: Multiplicative-Decomposable

The key of answering the above questions is the **multiplicative-decomposition** of kernels.

- ▶ We observed that many kernels of U-statistics is product of some functions with less arguments.
- ▶ Let's give a notation to capture this structure, it mimics the Einsum notation "ij,jk -> ".
- ▶ For $h_3^{\text{HOIF}}(\mathbf{X}) = f_1(X_1, X_2)f_2(X_2, X_3)$:

$$\mathcal{A}_3^{\text{HOIF}} = ((1, 2), (2, 3)), T_{ij}^{(k)} = f_k(X_1, X_2), k = 1, 2.$$

- ▶ For a part of $\text{dCov}^2(X, Y)$: $\frac{(n-4)!}{n!} \sum_{i \neq j \neq q \neq r} a_{ij} b_{qr}$

$$\mathcal{A}^{\text{dCov},1} = ((1, 2), (3, 4)), T^{(1)} = a, T^{(2)} = b.$$

- ▶ For $C(R_1) = \frac{1}{2} \sum_{i_1 \neq i_2 \neq i_3} A_{i_1 i_2} A_{i_2 i_3} B_{i_3 i_1}$

$$\mathcal{A}_3^{\text{Motif}} = ((1, 2), (2, 3), (3, 1)), T^{(1)} = T^{(2)} = A, T^{(3)} = B.$$

Key Assumption: Multiplicative-Decomposable

The key of answering the above questions is the **multiplicative-decomposition** of kernels.

- ▶ We observed that many kernels of U-statistics is product of some functions with less arguments.
- ▶ Let's give a notation to capture this structure, it mimics the Einsum notation "ij,jk ->".
- ▶ For $h_3^{\text{HOIF}}(\mathbf{X}) = f_1(X_1, X_2)f_2(X_2, X_3)$:

$$\mathcal{A}_3^{\text{HOIF}} = ((1, 2), (2, 3)), T_{ij}^{(k)} = f_k(X_1, X_2), k = 1, 2.$$

- ▶ For a part of $\text{dCov}^2(X, Y)$: $\frac{(n-4)!}{n!} \sum_{i \neq j \neq q \neq r} a_{ij} b_{qr}$

$$\mathcal{A}^{\text{dCov}, 1} = ((1, 2), (3, 4)), T^{(1)} = a, T^{(2)} = b.$$

- ▶ For $C(R_1) = \frac{1}{2} \sum_{i_1 \neq i_2 \neq i_3} A_{i_1 i_2} A_{i_2 i_3} B_{i_3 i_1}$

$$\mathcal{A}_3^{\text{Motif}} = ((1, 2), (2, 3), (3, 1)), T^{(1)} = T^{(2)} = A, T^{(3)} = B.$$

Key Assumption: Multiplicative-Decomposable

The key of answering the above questions is the **multiplicative-decomposition** of kernels.

- ▶ We observed that many kernels of U-statistics is product of some functions with less arguments.
- ▶ Let's give a notation to capture this structure, it mimics the Einsum notation "ij,jk -> ".
- ▶ For $h_3^{\text{HOIF}}(\mathbf{X}) = f_1(X_1, X_2)f_2(X_2, X_3)$:

$$\mathcal{A}_3^{\text{HOIF}} = ((1, 2), (2, 3)), T_{ij}^{(k)} = f_k(X_1, X_2), k = 1, 2.$$

- ▶ For a part of $\text{dCov}^2(X, Y)$: $\frac{(n-4)!}{n!} \sum_{i \neq j \neq q \neq r} a_{ij} b_{qr}$

$$\mathcal{A}^{\text{dCov},1} = ((1, 2), (3, 4)), T^{(1)} = a, T^{(2)} = b.$$

- ▶ For $C(R_1) = \frac{1}{2} \sum_{i_1 \neq i_2 \neq i_3} A_{i_1 i_2} A_{i_2 i_3} B_{i_3 i_1}$

$$\mathcal{A}_3^{\text{Motif}} = ((1, 2), (2, 3), (3, 1)), T^{(1)} = T^{(2)} = A, T^{(3)} = B.$$

Algorithm Framework : Sparsification

- For $h_3^{\text{HOIF}}(\mathbf{X}) = f_1(X_1, X_2)f_2(X_2, X_3)$, let

$$T_{ij}^{(k)} = f_1(X_i, X_j), \quad k = 1, 2.$$

$$\tilde{T}_{ij}^{(k)} = \begin{cases} T_{ij}^{(1)} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}, \quad k = 1, 2.$$

- Then we have

$$\begin{aligned} & \mathbb{U}[h_3^{\text{HOIF}}] \\ &= \sum_{i_1 \neq i_2 \neq i_3} T_{i_1 i_2}^{(1)} T_{i_2 i_3}^{(2)} \\ &= \sum_{i_1 \neq i_2 \neq i_3} \tilde{T}_{i_1 i_2}^{(1)} \tilde{T}_{i_2 i_3}^{(2)} \\ &= \left(\sum_{i_1, i_2, i_3} - \sum_{(i_1=i_2), i_3} - \sum_{(i_1=i_3), i_2} - \sum_{(i_2=i_3), i_1} + 2 \sum_{i_1=i_2=i_3} \right) \tilde{T}_{i_1 i_2}^{(1)} \tilde{T}_{i_2 i_3}^{(2)} \\ &= \left(\sum_{i_1, i_2, i_3} - \cancel{\sum_{(i_1=i_2), i_3}} - \sum_{(i_1=i_3), i_2} - \cancel{\sum_{(i_2=i_3), i_1}} + 2 \cancel{\sum_{i_1=i_2=i_3}} \right) \tilde{T}_{i_1 i_2}^{(1)} \tilde{T}_{i_2 i_3}^{(2)} \end{aligned}$$

Algorithm Framework : Sparsification

- In general, let

$$\Pi_m^{\mathcal{A}} = \{\pi \in \Pi_m \mid \forall Q \in \pi, \forall A \in \mathcal{A}, |Q \cap \text{set}[A]| < 2\}.$$

- We just need to sum over $\pi \in \Pi_m^{\mathcal{A}}$.

| Order (m) | V-stat terms (Bell number) | V-stat terms (Sparsification) | Ratio |
|---------------|-------------------------------|----------------------------------|-------|
| 3 | 5 | 2 | 0.4 |
| 4 | 15 | 5 | 0.333 |
| 5 | 52 | 15 | 0.288 |
| 6 | 203 | 52 | 0.256 |
| 7 | 877 | 203 | 0.231 |
| 8 | 4140 | 877 | 0.212 |
| 9 | 21147 | 4140 | 0.196 |
| 10 | 115975 | 21147 | 0.182 |
| 11 | 678570 | 115975 | 0.171 |
| 12 | 4213597 | 678570 | 0.161 |

Table: The Sparsification trick on HOIF

Algorithm Framework & Implementation

Updated Algorithm Framework

Input: decomposition signature \mathcal{A} , corresponding **diagonal-excluded** tensors $\tilde{T}^{(1)}, \tilde{T}^{(2)}, \dots$

Output: U-statistic $\mathbb{U}[h]$

1. Initialize $\mathbb{U}[h] \leftarrow 0$
2. Get m from \mathcal{A}
3. For each partition $\pi \in \Pi_m^{\mathcal{A}}$:
 - 3.1 Compute coefficient μ_{π}
 - 3.2 Compute V-statistic $\mathbb{V}[\pi](h)$ via Einsum with $\tilde{T}^{(1)}, \tilde{T}^{(2)}, \dots$
 - 3.3 $\mathbb{U}[h] \leftarrow \mathbb{U}[h] + \mu_{\pi} \cdot \mathbb{V}[\pi](h)$
4. Return $\mathbb{U}[h]$

Our new package `u-stats` for computing U-statistics is available on PyPI:

<https://pypi.org/project/u-stats/>

Install via:

```
pip install u-stats
```


Outline

Introduction

On Computing U-Statistics

On Complexity of Computing U-Statistics

Applications

Conclusions

Complexity: V-statistics

For **Complexity of V-statistics**:

► For $h_3^{\text{HOIF}}(\mathbf{X}) = f_1(X_1, X_2)f_2(X_2, X_3)$, recall

$$\begin{aligned} \mathbb{U}[h_3^{\text{HOIF}}] \\ = \sum_{i_1, i_2, i_3} \tilde{T}_{i_1 i_2}^{(1)} \tilde{T}_{i_2 i_3}^{(2)} - \sum_{i_1, i_2} \tilde{T}_{i_1 i_2}^{(1)} \tilde{T}_{i_2 i_1}^{(2)} \end{aligned}$$

► Consider the first V-statistic, with $\mathcal{A}_{\pi_1} = ((1, 2), (2, 3))$.

Order $(1 \rightarrow 2 \rightarrow 3)$:

Order $(2 \rightarrow 1 \rightarrow 3)$:

$$\sum_{i_1, i_2, i_3} \tilde{T}_{i_1 i_2}^{(1)} \tilde{T}_{i_2 i_3}^{(2)} \xrightarrow{\text{sum over } i_1} \sum_{i_2, i_3} \tilde{T}_{i_2}^{(3)} \tilde{T}_{i_2 i_3}^{(2)}$$

$$\sum_{i_1, i_2, i_3} \tilde{T}_{i_1 i_2}^{(1)} \tilde{T}_{i_2 i_3}^{(2)} \xrightarrow{\text{sum over } i_2} \sum_{i_1, i_3} \tilde{T}_{i_1 i_3}^{(5)}$$

$$\xrightarrow{\text{sum over } i_2} \sum_{i_3} \tilde{T}_{i_3}^{(4)}$$

$$\xrightarrow{\text{sum over } i_1} \sum_{i_3} \tilde{T}_{i_3}^{(6)}$$

$$\xrightarrow{\text{sum over } i_3} \text{Final Result}$$

$$\xrightarrow{\text{sum over } i_3} \text{Final Result}$$

$$\Rightarrow \max \text{ complexity} = O(n^2)$$

$$\Rightarrow \max \text{ complexity} = O(n^3)$$

Complexity: V-statistics

- In general, the complexity will depend on the computational ordering of indexes.
- A simple graph $G_{\mathcal{A}}$ captures the decomposable structure \mathcal{A} .

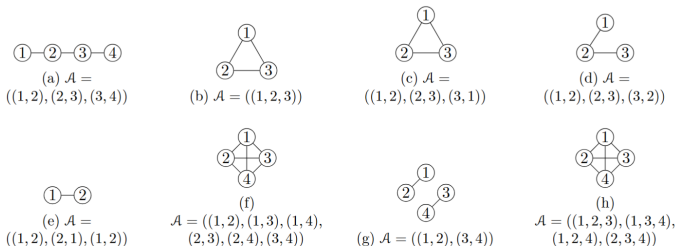
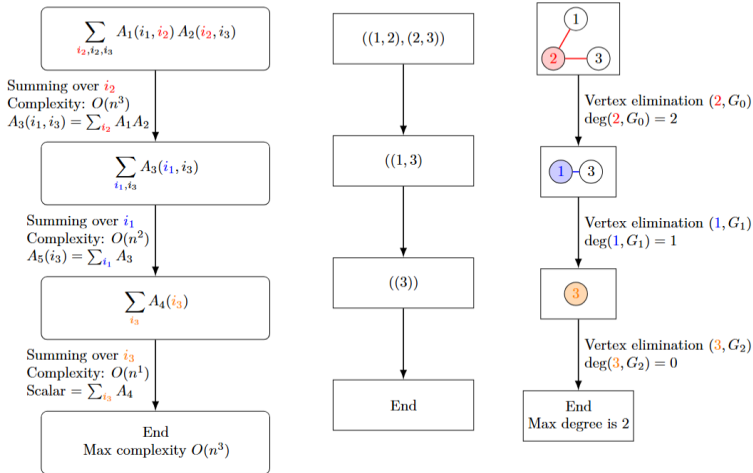


Figure 1: Examples of decomposition graphs associated with different decomposition signatures.

- Summing over one index corresponding to **eliminate** the vertex in graph.

Complexity: V-statistics



Procedure of Computing a 3-th order V-statistic with $\mathcal{A} = ((1, 2), (2, 3))$

Complexity: V-statistics

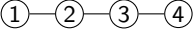
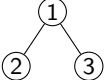
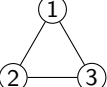
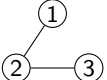

| Decomposition Graph | Elimination Ordering | Treewidth | Complexity |
|---|---|-----------|------------|
|  | $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ | 1 | $O(n^2)$ |
|  | $2 \rightarrow 3 \rightarrow 1$ | 1 | $O(n^2)$ |
|  | $1 \rightarrow 2 \rightarrow 3$ | 2 | $O(n^3)$ |
|  | $1 \rightarrow 3 \rightarrow 2$ | 1 | $O(n^2)$ |
|  | $1 \rightarrow 2$ | 1 | $O(n^2)$ |

Table: Decomposition graphs, elimination orderings, and treewidths.

treewidth of a simple graph G is denoted by $\text{tw}(G)$. Let $V(G) = [m] = \{1, 2, \dots, m\}$, then:

$$\text{tw}(G) = \min_{\sigma \in \text{Perm}([m])} \max_{i \in [m]} \deg(i, G_{i-1}), \quad G_0 = G, \quad G_i = \text{elim}_i(G_{i-1}).$$

Complexity: V-statistics

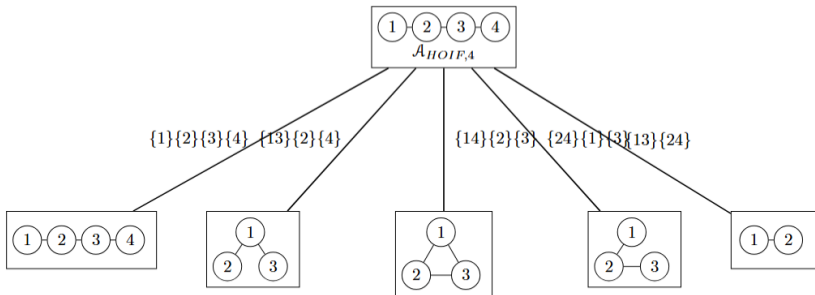
Lemma 2 (C., Zhang, Liu, 25)

Let \mathbb{X} be a non-empty set and $h : \mathbb{X}^m \rightarrow \mathbb{R}$ be a kernel function with a multiplicative decomposable \mathcal{A} and the complexity of evaluation of h is $o(n)$. Then for any $\mathbf{X} \in \mathbb{X}^n : n \geq m$,

$$\text{Complexity}(\mathbb{V}[h]) = O(n^{\text{tw}(G_{\mathcal{A}})+1})$$

- ▶ This result is an extension of [Markov and Shi \(2008\)](#).
- ▶ Finding the optimal ordering (treewidth) is known to be NP-hard. ([Arnborg et al., 1987](#))
- ▶ In practice, heuristic algorithm such as greedy algorithm is used. It is well studied in **tensor network** community [Gray and Kourtis \(2021\)](#).
- ▶ Our **u-stats** use **opt-einsum** to choose ordering.

Complexity: U -statistics



Quotient graphs induced by U -statistics with $\mathcal{A} = ((1, 2), (2, 3), (3, 4))$

Complexity: U -statistics

Theorem 1 (C., Zhang, Liu, 25)

Let \mathbb{X} be a non-empty set and $h : \mathbb{X}^m \rightarrow \mathbb{R}$ be a kernel function with a multiplicative decomposable \mathcal{A} and the complexity of evaluation of h is $o(n)$. Then for any $\mathbf{X} \in \mathbb{X}^n : n \geq m$,

$$\text{Complexity}(\mathbb{U}[h]) = \max_{\pi \in \Pi_m^{\mathcal{A}}} O(n^{\text{tw}(G_{\mathcal{A}\pi})+1})$$

- ▶ All graph can be determined by **quotient graph**, for U -statistic with kernel function satisfying graph $G_{\mathcal{A}}$:
- ▶ The treewidth of quotient graph is less studied.

$$\mathbb{V}[\pi](h) \leftrightarrow G_{\mathcal{A}\pi} \leftrightarrow G_{\mathcal{A}}/\pi$$

Outline

Introduction

On Computing U-Statistics

On Complexity of Computing U-Statistics

Applications

Conclusions

Application 2: Motif Counts

- ▶ We compare the performance of our `u-stats` with `igraph` and `cugraph`. `cugraph` is developed by NVIDIA.
- ▶ On Erdős-Rényi graphs $G(n, p)$:
 - ▶ n is the number of vertices.
 - ▶ p is the probability of edge existence.
- ▶ CPU parallel with 64 cores via `pytorch.einsum` for `u-stats` compared with `igraph`
 - ▶ Count all types of 3-node or 4-node motifs.
- ▶ GPU parallel with RTX 4090 via `pytorch.einsum` for `u-stats` compared with `cugraph`.
 - ▶ Count only triangles.

Application 2: Motif Counts

- For the 3-node motifs,

Table: Runtime comparison of exact all 3-node motif counts using u-stats and igraph on Erdős-Rényi graphs $G(n, p)$ with $n = 5623$. Experiments were run on Intel Xeon ICX Platinum 8358 CPUs (2.6GHz, 64 total cores) with 512 GB of memory. "Speedup ratio" compares igraph to our parallel method; "single core speedup" assumes single-core execution.

| Edge Prob. p | u-stats Time (s) | igraph Time (s) | Speedup Ratio | Single-Core Speedup |
|----------------|------------------|-----------------|---------------|---------------------|
| 0.001 | 0.617 | 0.025 | 0.040 | 0.001 |
| 0.005 | 0.649 | 0.353 | 0.544 | 0.008 |
| 0.010 | 0.692 | 1.795 | 2.595 | 0.041 |
| 0.020 | 0.783 | 10.594 | 13.529 | 0.211 |
| 0.050 | 1.038 | 136.272 | 131.288 | 2.051 |
| 0.080 | 1.298 | 514.702 | 396.415 | 6.194 |
| 0.100 | 1.453 | 960.299 | 660.690 | 10.323 |
| 0.150 | 1.864 | 3003.490 | 1611.389 | 25.178 |
| 0.200 | 2.275 | 6725.515 | 2955.656 | 46.182 |

Application 2: Motif Counts

- For the 4-node motifs,

Table: Runtime comparison of exact all 4-node motif counts using u-stats and igraph on Erdős-Rényi graphs $G(n, p)$ with $n = 2000$. Experiments were run on Intel Xeon ICX Platinum 8358 CPUs (2.6GHz, 64 total cores) with 512 GB of memory. "Speedup ratio" compares igraph to our parallel method; "single core speedup" assumes single-core execution.

| Edge Prob. p | u-stats Time (s) | igraph Time (s) | Speedup Ratio | Single-Core Speedup |
|----------------|------------------|-----------------|---------------|---------------------|
| 0.001 | 70.959 | 0.004 | 0. | 0. |
| 0.005 | 71.947 | 0.168 | 0.002 | 0. |
| 0.010 | 68.463 | 1.453 | 0.021 | 0. |
| 0.020 | 65.634 | 15.573 | 0.237 | 0.004 |
| 0.050 | 65.786 | 394.532 | 5.997 | 0.094 |
| 0.080 | 66.866 | 2371.373 | 35.465 | 0.554 |
| 0.100 | 68.219 | 5396.057 | 79.099 | 1.236 |
| 0.150 | 65.969 | 23003.041 | 348.692 | 5.448 |
| 0.200 | 65.259 | 61796.760 | 946.940 | 14.796 |

Application 2: Motif Counts

Table: Runtime comparison (seconds) of exact triangle counting using u-stats and cugraph with a single **GPU (NVIDIA RTX 4090, 24GB)**, across varying edge probability and $n = 10000$. Here "OOM" indicates Out Of Memory error

| Edge Probability (p) | u-stats | cugraph |
|--------------------------|---------|---------|
| 0.001 | 2.8014 | 0.1536 |
| 0.005 | 2.9564 | 0.5664 |
| 0.010 | 3.3363 | 1.0379 |
| 0.020 | 3.7830 | 1.9797 |
| 0.050 | 5.0856 | 4.9804 |
| 0.080 | 6.3764 | 8.1346 |
| 0.100 | 7.2829 | 10.3661 |
| 0.150 | 9.6873 | 15.3828 |
| 0.200 | 11.7283 | OOM |
| 0.800 | 37.6880 | OOM |

Application 3: Higher-Order Influence Functions

- ▶ HOIFs involve computing a class of U-statistics like

$$h_m^{\text{HOIF}}(X_1, \dots, X_m) = f_1(X_1, X_2) f_2(X_2, X_3) \cdots f_{m-1}(X_{m-1}, X_m)$$

- ▶ We test **u-stats** on computing HOIFs on platforms both in CPU and GPU (with **pytorch**).
- ▶ For CPU

Table: Average runtime (in seconds) using CPU parallel computation. Experiments were conducted on Intel Xeon ICX Platinum 8358 CPUs (2.6GHz, 64 total cores) with 512 GB of memory, evaluated across varying sample sizes and orders of HOIF-type U-statistics.

| $m \backslash n$ | 1000 | 2000 | 4000 | 8000 | 10000 |
|------------------|---------|---------|----------|-----------|-----------|
| 2 | 0.64 | 0.00141 | 0.01298 | 0.03174 | 0.04746 |
| 3 | 0.00396 | 0.01518 | 0.07392 | 0.26849 | 0.45976 |
| 4 | 0.02321 | 0.07313 | 0.32765 | 2.09545 | 2.36766 |
| 5 | 0.09853 | 0.36239 | 1.71419 | 9.11349 | 14.21350 |
| 6 | 0.38878 | 1.47719 | 7.44444 | 40.22143 | 58.85063 |
| 7 | 1.91677 | 6.75947 | 34.08805 | 195.31414 | 290.54295 |

Application 3: Higher-Order Influence Functions

► For GPU

Table: Average runtime (in seconds) using a single GPU (NVIDIA RTX 4090, 24GB) with parallel computation, across varying sample sizes and orders of HOIF-type U-statistics.

| $m \backslash n$ | 1000 | 2000 | 4000 | 8000 | 10000 |
|------------------|---------|---------|---------|---------|----------|
| 2 | 0.00184 | 0.00151 | 0.00155 | 0.00238 | 0.00339 |
| 3 | 0.00172 | 0.00147 | 0.00193 | 0.00577 | 0.00745 |
| 4 | 0.00305 | 0.00353 | 0.00707 | 0.03612 | 0.06245 |
| 5 | 0.00835 | 0.01089 | 0.03456 | 0.19807 | 0.36160 |
| 6 | 0.02608 | 0.03906 | 0.16981 | 1.12072 | 2.13328 |
| 7 | 0.12031 | 0.19132 | 0.91884 | 6.45225 | 12.21350 |

Outline

Introduction

On Computing U-Statistics

On Complexity of Computing U-Statistics

Applications

Conclusions

Conclusions

- ▶ Summary
 - ▶ Developed a **novel algorithm** for the exact computation of U -statistics.
 - ▶ Created a Python package named **u-stats** that leverages the highly-optimized Einsum function within `pytorch.einsum` and `numpy.einsum` for efficient computation.
 - ▶ Find an **optimal upper bound** for the computational complexity of exactly computing U -statistics.
- ▶ Limitation and future work:
 - ▶ Memory usage is high: for HOIFs, it's not available for sample size larger than 2000 and order larger than 8 – Hybrid between for-loop & Einsum (or we can wait for better GPUs)?
 - ▶ Interface for R.
 - ▶ Non-scalar-valued U -statistics.

References

- Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k -tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.
- Anirban Chatterjee, Soham Dan, and Bhaswar B Bhattacharya. Higher-order graphon theory: Fluctuations, degeneracies, and inference. *arXiv preprint arXiv:2404.13822*, 2024.
- Johnnie Gray and Stefanos Kourtis. Hyper-optimized tensor network contraction. *Quantum*, 5:410, 2021.
- Igor L Markov and Yaoyun Shi. Simulating quantum computation by contracting tensor networks. *SIAM Journal on Computing*, 38(3):963–981, 2008.
- James Robins, Lingling Li, Eric Tchetgen Tchetgen, and Aad van der Vaart. Higher order influence functions and minimax estimation of nonlinear functionals. In *Probability and Statistics: Essays in Honor of David A. Freedman*, pages 335–421. Institute of Mathematical Statistics, 2008.
- James Robins, Lingling Li, Eric Tchetgen Tchetgen, and Aad van der Vaart. Technical report: Higher order influence functions and minimax estimation of nonlinear functionals. *arXiv preprint arXiv:1601.05820*, 2016.
- Meijia Shao, Dong Xia, and Yuan Zhang. U-statistic reduction: Higher-order accurate risk control and statistical-computational trade-off. *Journal of the American Statistical Association*, pages 1–14, 2025.
- Volker Strassen. The asymptotic spectrum of tensors and the exponent of matrix multiplication. In *27th Annual Symposium on Foundations of*

Thank You!

For more details:

- ▶ Paper draft (arXiv):
<https://arxiv.org/abs/2508.12627>
- ▶ Software (GitHub):
github.com/zrq1706/U-Statistics-python
- ▶ Personal website:
<https://cxy0714.github.io/>