SPECTRUM APPROXIMATION BEYOND FAST MATRIX MULTIPLICATION: ALGORITHMS AND HARDNESS

Cameron Musco (MIT), Praneeth Netrapalli (MSR), Aaron Sidford (Stanford), Shashanka Ubaru (UMN), David Woodruff (CMU)

ITCS 2018

Despite tons of algorithmic work, we don't understand the fundamental complexity of many linear algebraic problems.

Despite tons of algorithmic work, we don't understand the fundamental complexity of many linear algebraic problems.

- E.g. linear system solving, eigenvector/value computation, determinant computation, and low-rank approximation.

MOTIVATING QUESTION

Despite tons of algorithmic work, we don't understand the fundamental complexity of many linear algebraic problems.

- E.g. linear system solving, eigenvector/value computation, determinant computation, and low-rank approximation.
- Most can be solved with 'heavy hammers' like matrix inversion, full eigendecomposition, and SVD. These computations are known to be equivalent in difficulty to matrix multiplication and take $O(n^\omega)$ time for $n \times n$ matrices, $O(n^3)$ time in practice.

Despite tons of algorithmic work, we don't understand the fundamental complexity of many linear algebraic problems.

- E.g. linear system solving, eigenvector/value computation, determinant computation, and low-rank approximation.
- Most can be solved with 'heavy hammers' like matrix inversion, full eigendecomposition, and SVD. These computations are known to be equivalent in difficulty to matrix multiplication and take $O(n^\omega)$ time for $n \times n$ matrices, $O(n^3)$ time in practice.

Which natural linear algebraic problems can be solved in $o(n^\omega)$ time for general matrices? Conversely, which are truly as hard as matrix multiplication?

Despite tons of algorithmic work, we don't understand the fundamental complexity of many linear algebraic problems.

- E.g. linear system solving, eigenvector/value computation, determinant computation, and low-rank approximation.
- Most can be solved with 'heavy hammers' like matrix inversion, full eigendecomposition, and SVD. These computations are known to be equivalent in difficulty to matrix multiplication and take $O(n^\omega)$ time for $n \times n$ matrices, $O(n^3)$ time in practice.

Which natural linear algebraic problems can be solved in $o(n^\omega)$ time for general matrices? Conversely, which are truly as hard as matrix multiplication?[1]

[1] Important even if $\omega = 2$.

$o(n^\omega)$ Time Algorithms – Two Main Approaches:

$o(n^\omega)$ Time Algorithms – Two Main Approaches:

- **Assumptions** like bounded condition number or eigengaps yield accurate algorithms for linear systems, more general matrix functions, and eigenvector computation. Algorithms are typically iterative and linearly convergent (i.e., with $\log(1/\epsilon)$ error dependence).

$o(n^\omega)$ Time Algorithms – Two Main Approaches:

- Assumptions like bounded condition number or eigengaps yield accurate algorithms for linear systems, more general matrix functions, and eigenvector computation. Algorithms are typically iterative and linearly convergent (i.e., with $\log(1/\epsilon)$ error dependence).
- Coarser approximation methods give general solutions e.g. for linear systems, eigenvalue computation, and low-rank approximation, with $\text{poly}(1/\epsilon)$ dependence.

$o(n^\omega)$ Time Algorithms – Two Main Approaches:

- **Assumptions** like bounded condition number or eigengaps yield accurate algorithms for linear systems, more general matrix functions, and eigenvector computation. Algorithms are typically iterative and linearly convergent (i.e., with $\log(1/\epsilon)$ error dependence).
- **Coarser approximation** methods give general solutions e.g. for linear systems, eigenvalue computation, and low-rank approximation, with $\mathrm{poly}(1/\epsilon)$ dependence.
- **Essentially nothing is known beyond these techniques.**

$o(n^\omega)$ Time Algorithms – Two Main Approaches:

- **Assumptions** like bounded condition number or eigengaps yield accurate algorithms for linear systems, more general matrix functions, and eigenvector computation. Algorithms are typically iterative and linearly convergent (i.e., with $\log(1/\epsilon)$ error dependence).
- **Coarser approximation** methods give general solutions e.g. for linear systems, eigenvalue computation, and low-rank approximation, with $\text{poly}(1/\epsilon)$ dependence.
- **Essentially nothing is known beyond these techniques.**
- All known linear algebraic algorithms which work with high accuracy on general matrices require full $n \times n \times n$ matrix multiplication (i.e. $O(n^\omega)$ time). Why?

Lower Bounds (much less work here):

Lower Bounds (much less work here):

- [Baur, Strassen '83] shows that an arithmetic circuit for the determinant with M gates gives a circuit for matrix inversion/multiplication with $O(M)$ gates.

Lower Bounds (much less work here):

- [Baur, Strassen '83] shows that an arithmetic circuit for the determinant with M gates gives a circuit for matrix inversion/multiplication with $O(M)$ gates.
- No reduction known for uniform computation.

Lower Bounds (much less work here):

- [Baur, Strassen '83] shows that an arithmetic circuit for the determinant with M gates gives a circuit for matrix inversion/multiplication with $O(M)$ gates.
- No reduction known for uniform computation.
- An emerging line of work on reductions and hardness for linear algebraic problems [Kyng, Zhang '17], [Backurs, Indyk, Schmidt '17], [Musco, Woodruff '17].

THIS WORK

Algorithms: Give $o(n^\omega)$ time approximation algorithms (poly($1/\epsilon$) dependence) for a many spectral summarization tasks, like trace norm computation, SVD entropy, etc.

Algorithms: Give $o(n^\omega)$ time approximation algorithms (poly$(1/\epsilon)$ dependence) for a many spectral summarization tasks, like trace norm computation, SVD entropy, etc.

Lower Bounds: Show that it may be hard to find highly accurate $o(n^\omega)$ time methods for many of these tasks by giving reductions from matrix multiplication.

Algorithms: Give $o(n^{\omega})$ time approximation algorithms (poly$(1/\epsilon)$ dependence) for a many spectral summarization tasks, like trace norm computation, SVD entropy, etc.

Lower Bounds: Show that it may be hard to find highly accurate $o(n^{\omega})$ time methods for many of these tasks by giving reductions from matrix multiplication.

- Bounds extend to many natural problems like determinant, trace inverse, effective resistance computation, etc.

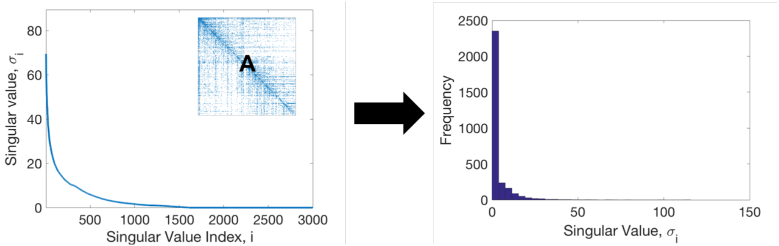
Basic Question: Given a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, can we approximate, in some useful way, its singular value spectrum $\sigma_1 > .... > \sigma_n \geq 0$ without performing a full SVD. I.e., in $o(n^\omega)$ time, for the current value of ω?

Basic Question: Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, can we approximate, in some useful way, its singular value spectrum $\sigma_1 > .... > \sigma_n \geq 0$ without performing a full SVD. I.e., in $o(n^\omega)$ time, for the current value of ω?

Our algorithmic contribution: Show how to efficiently compute an approximate histogram of the spectrum.

Basic Question: Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, can we approximate, in some useful way, its singular value spectrum $\sigma_1 > .... > \sigma_n \geq 0$ without performing a full SVD. I.e., in $o(n^\omega)$ time, for the current value of ω?

Our algorithmic contribution: Show how to efficiently compute an approximate histogram of the spectrum.



APPLICATION: $o(n^\omega)$ TIME MATRIX NORMS

Use our histogram primitive to give the first algorithms for approximating many matrix norms in $o(n^\omega)$ time.

Use our histogram primitive to give the first algorithms for approximating many matrix norms in $o(n^\omega)$ time.

- $\tilde{O}(n^{2.18}/\epsilon^3)$ time algorithm for approximating the nuclear norm to $1+\epsilon$ relative error. $\tilde{O}(n^{2.33}/\epsilon^3)$ time without fast matrix mult.

$$\|\mathbf{A}\|_* = \sum_{i=1}^{n} \sigma_i.$$

APPLICATION: $o(n^\omega)$ TIME MATRIX NORMS

Use our histogram primitive to give the first algorithms for approximating many matrix norms in $o(n^\omega)$ time.

- $\tilde{O}(n^{2.18}/\epsilon^3)$ time algorithm for approximating the nuclear norm to $1+\epsilon$ relative error. $\tilde{O}(n^{2.33}/\epsilon^3)$ time without fast matrix mult.

$$\|\mathbf{A}\|_* = \sum_{i=1}^{n} \sigma_i.$$

- $\tilde{O}(n^2 \cdot p/\epsilon^3)$ time algorithm for approximating the Schatten p-norm for any real $p > 2$:

$$\|\mathbf{A}\|_p = \left(\sum_{i=1}^{n} \sigma_i^p\right)^{1/p}.$$

APPLICATION: $o(n^\omega)$ TIME MATRIX NORMS

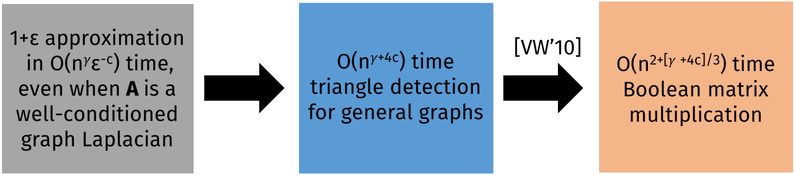
Use our histogram primitive to give the first algorithms for approximating many matrix norms in $o(n^\omega)$ time.

- $\tilde{O}(n^{2.18}/\epsilon^3)$ time algorithm for approximating the nuclear norm to $1+\epsilon$ relative error. $\tilde{O}(n^{2.33}/\epsilon^3)$ time without fast matrix mult.
$$\|\mathbf{A}\|_* = \sum_{i=1}^{n} \sigma_i.$$
- $\tilde{O}(n^2 \cdot p/\epsilon^3)$ time algorithm for approximating the Schatten p-norm for any real $p > 2$:
$$\|\mathbf{A}\|_p = \left(\sum_{i=1}^{n} \sigma_i^p\right)^{1/p}.$$
- Results for general Schatten p-norms, SVD entropy, and more general matrix norms of the form $\sum_{i=1}^{n} g(\sigma_i)$.

In contrast, we prove that higher accuracy approximation is in some sense as hard as matrix multiplication.

In contrast, we prove that higher accuracy approximation is in some sense as hard as matrix multiplication.

- For $\|\mathbf{A}\|_p$ for any $p \neq 2$, SVD entropy, $\mathrm{tr}(\mathbf{A}^{-1})$, $\mathrm{tr}(\exp(\mathbf{A}))$, $\det(\mathbf{A})$, $\log(\det(\mathbf{A}))$, all pairs effective resistances:



In contrast, we prove that higher accuracy approximation is in some sense as hard as matrix multiplication.

- For $\|\mathbf{A}\|_p$ for any $p \neq 2$, SVD entropy, $\mathrm{tr}(\mathbf{A}^{-1})$, $\mathrm{tr}(\exp(\mathbf{A}))$, $\det(\mathbf{A})$, $\log(\det(\mathbf{A}))$, all pairs effective resistances:

1+ε approximation in $O(n^{\gamma}\varepsilon^{-c})$ time, even when A is a well-conditioned graph Laplacian

$\Rightarrow$

$O(n^{\gamma+4c})$ time triangle detection for general graphs

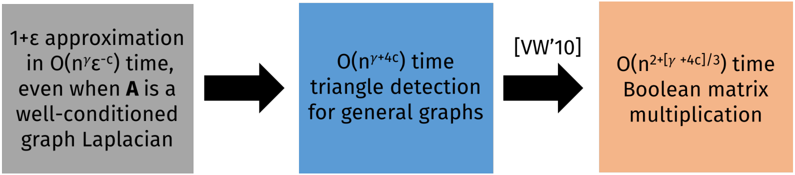
$\Rightarrow$ [VW'10]

$O(n^{2+[\gamma+4c]/3})$ time Boolean matrix multiplication

- Our $\tilde{O}(n^2/\epsilon^3)$ time algorithm for $\|\mathbf{A}\|_3$ would give faster triangle detection if the ϵ dependence was $\approx \frac{1}{\epsilon^{1/10}}$.

In contrast, we prove that higher accuracy approximation is in some sense as hard as matrix multiplication.

- For $\|\mathbf{A}\|_p$ for any $p \neq 2$, SVD entropy, $\mathrm{tr}(\mathbf{A}^{-1})$, $\mathrm{tr}(\exp(\mathbf{A}))$, $\det(\mathbf{A})$, $\log(\det(\mathbf{A}))$, all pairs effective resistances:



- Our $\tilde{O}(n^2/\epsilon^3)$ time algorithm for $\|\mathbf{A}\|_3$ would give faster triangle detection if the ϵ dependence was $\approx \frac{1}{\epsilon^{1/10}}$.
- An $O(n^{3-\delta} \cdot \log(1/\epsilon))$ time algorithm gives $\tilde{O}(n^{3-\delta})$ time triangle detection and $\tilde{O}(n^{3-\delta/3})$ time matrix multiplication.

HIGH LEVEL VIEW

Slow: $\Theta(n^\omega)$

Matrix multiplication
Matrix inversion
Eigendecomposition
Full SVD

Fast, Approximate:
$o(n^\omega)/\varepsilon^c$ for $c \geq 1/2$

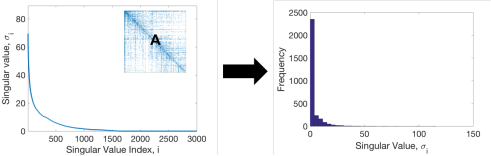
Linear systems
Top eigenvalue
Low-rank approximation
Schatten norms
SVD entropy

Fast, With Assumptions:

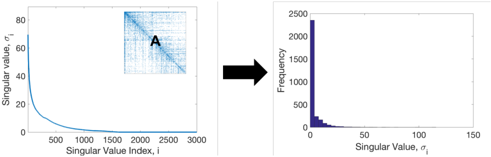
Linear systems
Eigenvectors/values
Low-rank approximation
$\exp(\mathbf{A})$, $\mathbf{A}^{1/2}$, etc.

Fast, Accurate, No Assumptions:
$o(n^\omega)/\varepsilon^c$ for small c
$o(n^\omega \log(1/\varepsilon))$

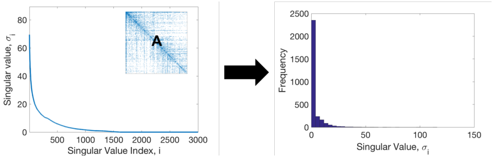
Anything?
Our results give negative evidence for many candidate problems.



Singular value, σ_i
Singular Value Index, i
A
Frequency
Singular Value, σ_i

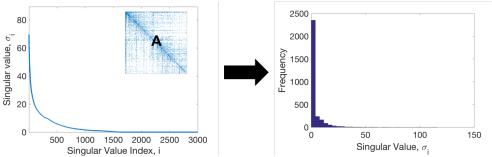


Key Primitive: Approximately count the number of singular values in each bucket.



Key Primitive: Approximately count the number of singular values in each bucket.

- Combine randomized trace estimation, stochastic optimization, polynomial approximation, and preconditioning.



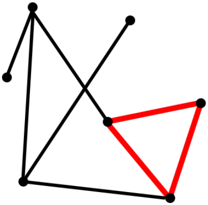
Key Primitive: Approximately count the number of singular values in each bucket.

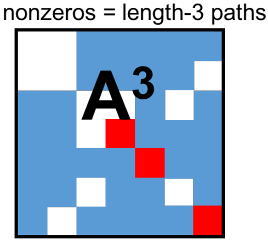
- Combine randomized trace estimation, stochastic optimization, polynomial approximation, and preconditioning.
- Leverage stochastic gradient based system solvers, which give better guarantees than the conjugate gradient method for certain spectrums. Use these guarantees to give generic speed ups. E.g., $O(n^{2.5}) \to O(n^{2.33})$ for $\|\mathbf{A}\|_*$.

Triangle detection naturally reduces to Schatten-3 norm estimation (almost).

Triangle detection naturally reduces to Schatten-3 norm estimation (almost).

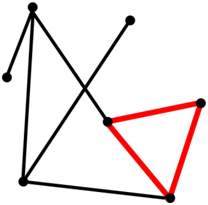
- Detecting a triangle in a graph is equivalent to testing if $\mathrm{tr}(\mathbf{A}^3) > 0$, where $\mathbf{A}$ is the adjacency matrix. I.e., relative error approximation to $\mathrm{tr}(\mathbf{A}^3)$ gives triangle detection.



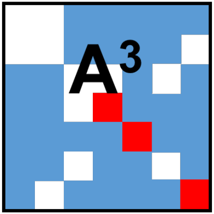


Triangle detection naturally reduces to Schatten-3 norm estimation (almost).

- Detecting a triangle in a graph is equivalent to testing if $\mathrm{tr}(\mathbf{A}^3) > 0$, where $\mathbf{A}$ is the adjacency matrix. I.e., relative error approximation to $\mathrm{tr}(\mathbf{A}^3)$ gives triangle detection.



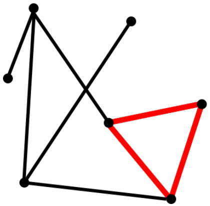
nonzeros = length-3 paths

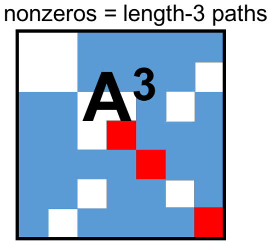


- $\mathrm{tr}(\mathbf{A}^3) = \sum_{i=1}^{n} \lambda_i(\mathbf{A}^3) = \sum_{i=1}^{n} \lambda_i(\mathbf{A})^3$

Triangle detection naturally reduces to Schatten-3 norm estimation (almost).

- Detecting a triangle in a graph is equivalent to testing if $\mathrm{tr}(\mathbf{A}^3) > 0$, where $\mathbf{A}$ is the adjacency matrix. I.e., relative error approximation to $\mathrm{tr}(\mathbf{A}^3)$ gives triangle detection.





- $\mathrm{tr}(\mathbf{A}^3) = \sum_{i=1}^{n} \lambda_i(\mathbf{A}^3) = \sum_{i=1}^{n} \lambda_i(\mathbf{A})^3 \neq \sum_{i=1}^{n} \sigma_i(\mathbf{A})^3 \overset{\text{def}}{=} \|\mathbf{A}\|_3^3$.

Idea: Replace adjacency matrix with PSD Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

Idea: Replace adjacency matrix with PSD Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

- $\lambda_i(\mathbf{L}) \geq 0$ for all i, so $\lambda_i(\mathbf{L}) = \sigma_i(\mathbf{L})$.

Idea: Replace adjacency matrix with PSD Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

- $\lambda_i(\mathbf{L}) \geq 0$ for all i, so $\lambda_i(\mathbf{L}) = \sigma_i(\mathbf{L})$.

$$\|\mathbf{L}\|_3^3 = \sum_{i=1}^{n} \lambda_i(\mathbf{L})^3 = \mathrm{tr}(\mathbf{L}^3) = \mathrm{tr}(\mathbf{D}^3) - 3\,\mathrm{tr}(\mathbf{D}^2\mathbf{A}) + 3\,\mathrm{tr}(\mathbf{D}\mathbf{A}^2) - \mathrm{tr}(\mathbf{A}^3)$$

Idea: Replace adjacency matrix with PSD Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

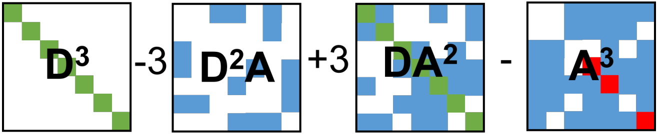
- $\lambda_i(\mathbf{L}) \geq 0$ for all i, so $\lambda_i(\mathbf{L}) = \sigma_i(\mathbf{L})$.

$$\|\mathbf{L}\|_3^3 = \sum_{i=1}^{n} \lambda_i(\mathbf{L})^3 = \operatorname{tr}(\mathbf{L}^3) = \operatorname{tr}(\mathbf{D}^3) - 3\operatorname{tr}(\mathbf{D}^2\mathbf{A}) + 3\operatorname{tr}(\mathbf{D}\mathbf{A}^2) - \operatorname{tr}(\mathbf{A}^3)$$

Idea: Replace adjacency matrix with PSD Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

- $\lambda_i(\mathbf{L}) \geq 0$ for all i, so $\lambda_i(\mathbf{L}) = \sigma_i(\mathbf{L})$.

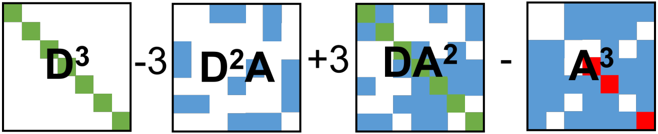
$$\|\mathbf{L}\|_3^3 = \sum_{i=1}^{n} \lambda_i(\mathbf{L})^3 = \operatorname{tr}(\mathbf{L}^3) = \operatorname{tr}(\mathbf{D}^3) - 3\operatorname{tr}(\mathbf{D}^2\mathbf{A}) + 3\operatorname{tr}(\mathbf{D}\mathbf{A}^2) - \operatorname{tr}(\mathbf{A}^3)$$



Idea: Replace adjacency matrix with PSD Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

- $\lambda_i(\mathbf{L}) \geq 0$ for all i, so $\lambda_i(\mathbf{L}) = \sigma_i(\mathbf{L})$.

$$\|\mathbf{L}\|_3^3 = \sum_{i=1}^{n} \lambda_i(\mathbf{L})^3 = \mathrm{tr}(\mathbf{L}^3) = \mathrm{tr}(\mathbf{D}^3) - 3\,\mathrm{tr}(\mathbf{D}^2\mathbf{A}) + 3\,\mathrm{tr}(\mathbf{D}\mathbf{A}^2) - \mathrm{tr}(\mathbf{A}^3)$$

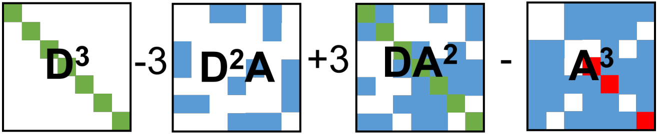


- $\mathrm{tr}(\mathbf{D}^3)$ exactly computable in $O(n^2)$ time.

Idea: Replace adjacency matrix with PSD Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

- $\lambda_i(\mathbf{L}) \geq 0$ for all i, so $\lambda_i(\mathbf{L}) = \sigma_i(\mathbf{L})$.

$$\|\mathbf{L}\|_3^3 = \sum_{i=1}^{n} \lambda_i(\mathbf{L})^3 = \mathrm{tr}(\mathbf{L}^3) = \mathrm{tr}(\mathbf{D}^3) - 3\,\mathrm{tr}(\mathbf{D}^2\mathbf{A}) + 3\,\mathrm{tr}(\mathbf{D}\mathbf{A}^2) - \mathrm{tr}(\mathbf{A}^3)$$

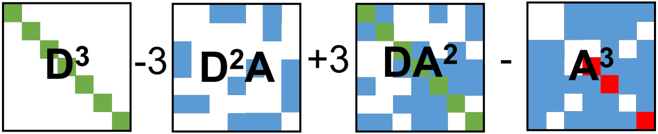


- $\mathrm{tr}(\mathbf{D}^3)$ exactly computable in $O(n^2)$ time.
- $\mathrm{tr}(\mathbf{D}^2\mathbf{A}) = 0$.

Idea: Replace adjacency matrix with PSD Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

- $\lambda_i(\mathbf{L}) \geq 0$ for all i, so $\lambda_i(\mathbf{L}) = \sigma_i(\mathbf{L})$.

$$\|\mathbf{L}\|_3^3 = \sum_{i=1}^{n} \lambda_i(\mathbf{L})^3 = \mathrm{tr}(\mathbf{L}^3) = \mathrm{tr}(\mathbf{D}^3) - 3\,\mathrm{tr}(\mathbf{D}^2\mathbf{A}) + 3\,\mathrm{tr}(\mathbf{D}\mathbf{A}^2) - \mathrm{tr}(\mathbf{A}^3)$$



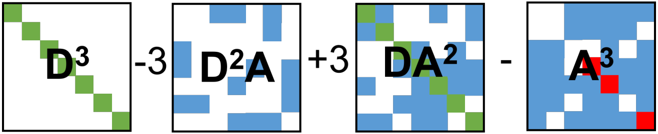
- $\mathrm{tr}(\mathbf{D}^3)$ exactly computable in $O(n^2)$ time.
- $\mathrm{tr}(\mathbf{D}^2\mathbf{A}) = 0$.
- $\mathrm{tr}(\mathbf{D}\mathbf{A}^2) = \mathrm{tr}(\mathbf{D}^2)$, exactly computable in $O(n^2)$ time.

REDUCTION SKETCH

Idea: Replace adjacency matrix with PSD Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

- $\lambda_i(\mathbf{L}) \geq 0$ for all i, so $\lambda_i(\mathbf{L}) = \sigma_i(\mathbf{L})$.

$$\|\mathbf{L}\|_3^3 = \sum_{i=1}^{n} \lambda_i(\mathbf{L})^3 = \mathrm{tr}(\mathbf{L}^3) = \mathrm{tr}(\mathbf{D}^3) - 3\,\mathrm{tr}(\mathbf{D}^2\mathbf{A}) + 3\,\mathrm{tr}(\mathbf{D}\mathbf{A}^2) - \mathrm{tr}(\mathbf{A}^3)$$



- $\mathrm{tr}(\mathbf{D}^3)$ exactly computable in $O(n^2)$ time.
- $\mathrm{tr}(\mathbf{D}^2\mathbf{A}) = 0$.
- $\mathrm{tr}(\mathbf{D}\mathbf{A}^2) = \mathrm{tr}(\mathbf{D}^2)$, exactly computable in $O(n^2)$ time.

Thus, additive $\delta < 1$ approximation to $\|\mathbf{L}\|_3^3$ gives additive $\delta < 1$ approximation to $\mathrm{tr}(\mathbf{A}^3)$ and triangle detection.

Additive $\delta < 1$ approximation to $\|\mathbf{L}\|_3^3$ gives additive $\delta < 1$ approximation to $\mathrm{tr}(\mathbf{A}^3)$ and triangle detection.

Additive $\delta < 1$ approximation to $\|\mathbf{L}\|_3^3$ gives additive $\delta < 1$ approximation to $\mathrm{tr}(\mathbf{A}^3)$ and triangle detection.

- $\|\mathbf{L}\|_3^3 \leq 8n^4$ for unweighted graphs.

Additive $\delta < 1$ approximation to $\|\mathbf{L}\|_3^3$ gives additive $\delta < 1$ approximation to $\mathrm{tr}(\mathbf{A}^3)$ and triangle detection.

- $\|\mathbf{L}\|_3^3 \leq 8n^4$ for unweighted graphs.
- So multiplicative $(1+\epsilon)$ approximation for $\epsilon < \frac{1}{8n^4}$ gives triangle detection.

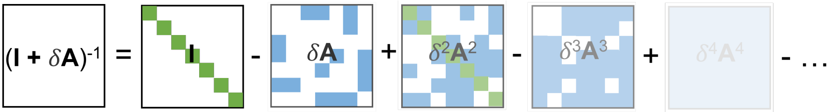
Additive $\delta < 1$ approximation to $\|\mathbf{L}\|_3^3$ gives additive $\delta < 1$ approximation to $\mathrm{tr}(\mathbf{A}^3)$ and triangle detection.

- $\|\mathbf{L}\|_3^3 \leq 8n^4$ for unweighted graphs.
- So multiplicative $(1+\epsilon)$ approximation for $\epsilon < \frac{1}{8n^4}$ gives triangle detection.
- Computing $(1 \pm \epsilon)\|\mathbf{L}\|_3^3$ in $O(n^\gamma \cdot \epsilon^{-c})$ time gives triangle detection in $O(n^{\gamma+4c})$ time.

Extend reduction to other matrix norms, logdet, trace inverse, trace exponential, etc. via Taylor expansion.

Extend reduction to other matrix norms, logdet, trace inverse, trace exponential, etc. via Taylor expansion.

- $\frac{1}{1+x} = \sum_{i=0}^{\infty}(-x)^i$.



Extend reduction to other matrix norms, logdet, trace inverse, trace exponential, etc. via Taylor expansion.

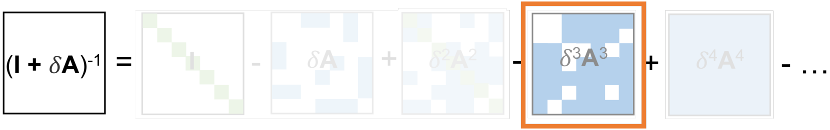
- $\frac{1}{1+x} = \sum_{i=0}^{\infty}(-x)^i$.

- Traces of large terms can again be computed exactly.

EXTENDING TO OTHER SPECTRAL SUMMARIES

Extend reduction to other matrix norms, logdet, trace inverse, trace exponential, etc. via Taylor expansion.

- $\frac{1}{1+x} = \sum_{i=0}^{\infty} (-x)^i$.

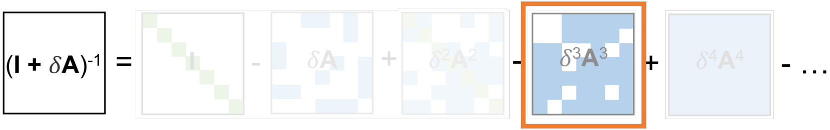


- Traces of large terms can again be computed exactly.
- After subtracting out these terms, $\mathrm{tr}(\delta^3 \mathbf{A}^3)$ dominates $\mathrm{tr}((\mathbf{I} + \delta \mathbf{A})^{-1})$ (we set $\delta = 1/\operatorname{poly}(n)$). So fine enough approximation gives triangle detection.

EXTENDING TO OTHER SPECTRAL SUMMARIES

Extend reduction to other matrix norms, logdet, trace inverse, trace exponential, etc. via Taylor expansion.

- $\frac{1}{1+x} = \sum_{i=0}^{\infty}(-x)^i$.



- Traces of large terms can again be computed exactly.
- After subtracting out these terms, $\mathrm{tr}(\delta^3\mathbf{A}^3)$ dominates $\mathrm{tr}((\mathbf{I}+\delta\mathbf{A})^{-1})$ (we set $\delta = 1/\operatorname{poly}(n)$). So fine enough approximation gives triangle detection.
- Bound for determinant is similar, by expanding out $\det(\mathbf{I}+\delta\mathbf{A}) = \prod_{i=1}^{n}(1+\delta\lambda_i(\mathbf{A}))$.

OPEN QUESTIONS (A SMALL SUBSET)

- Can any natural linear algebraic problem be solved in $o(n^{\omega})$ time for general matrices with high accuracy? (e.g. $\log(1/\epsilon)$ dependence on the error).
- Can we compute even a constant factor approximation to $\det(\mathbf{A})$ in $o(n^{\omega})$ time?
- Can the our reductions from matrix multiplication (through triangle detection) be tightened?

- Can any natural linear algebraic problem be solved in $o(n^\omega)$ time for general matrices with high accuracy? (e.g. $\log(1/\epsilon)$ dependence on the error).
- Can we compute even a constant factor approximation to $\det(\mathbf{A})$ in $o(n^\omega)$ time?
- Can the our reductions from matrix multiplication (through triangle detection) be tightened?
- Can we deterministically approximate $\sigma_1(\mathbf{A})$ without computing a full SVD (i.e. in $o(n^\omega)$ time).
- If we could solve any PSD linear system in $O(n^2)$ time, would this imply anything about how fast we can solve matrix multiplication?

OPEN QUESTIONS (A SMALL SUBSET)

- Can any natural linear algebraic problem be solved in $o(n^\omega)$ time for general matrices with high accuracy? (e.g. $\log(1/\epsilon)$ dependence on the error).
- Can we compute even a constant factor approximation to $\det(\mathbf{A})$ in $o(n^\omega)$ time?
- Can the our reductions from matrix multiplication (through triangle detection) be tightened?
- Can we deterministically approximate $\sigma_1(\mathbf{A})$ without computing a full SVD (i.e. in $o(n^\omega)$ time).
- If we could solve any PSD linear system in $O(n^2)$ time, would this imply anything about how fast we can solve matrix multiplication?

Thanks! Questions?