Searching for Cyclic-Invariant Fast Matrix Multiplication Algorithms

João Pinheiro, Grey Ballard, Frank Moore

Department of Computer Science

WAKE FOREST UNIVERSITY

Matrix Multiplication Algorithms

Given matrices $A, B \in \mathbb{R}^{n \times n}$ their matrix product $C \in \mathbb{R}^{n \times n}$ is:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Fast Matrix Multiply algorithms attempt to create a recursive algorithm that decreases the number of multiplications performed. Strassen's 1969 algorithm was the first of such group.

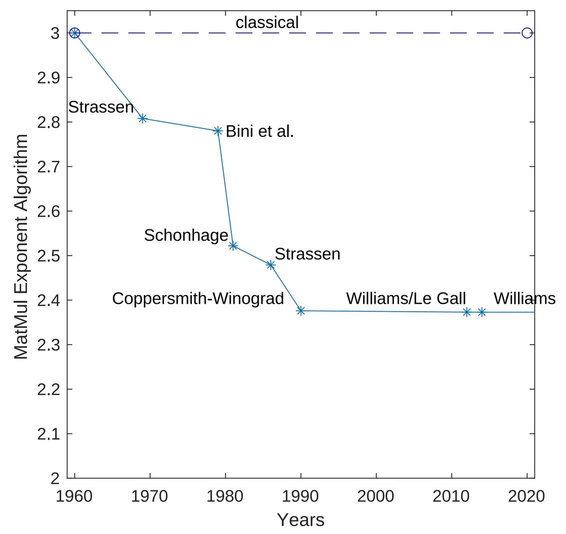


Figure: Matrix Multiply Exponents Through Time

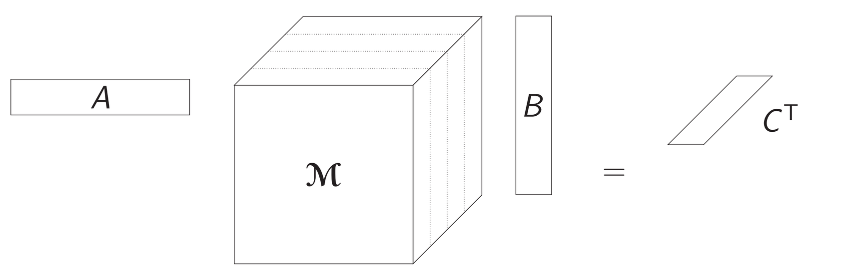
Strassen's Algorithm

$$\begin{aligned}
M_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22}) \cdot B_{11} \\
M_3 &= A_{11} \cdot (B_{12} - B_{22}) \\
M_4 &= A_{22} \cdot (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \cdot B_{22} \\
M_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22})
\end{aligned}$$

$$\begin{aligned}
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 - M_2 + M_3 + M_6
\end{aligned}$$

Matrix Multiplication as Tensors

Matrix Multiplication can be represented by tensors:



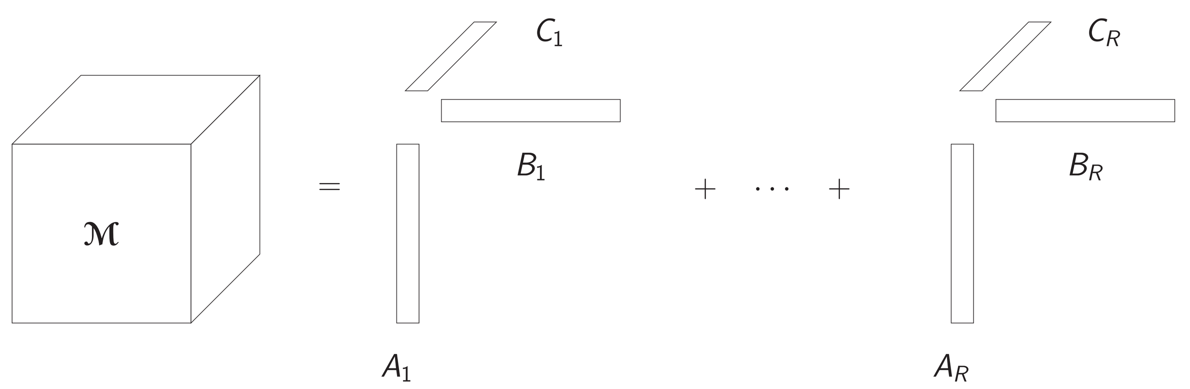
$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad M_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$M_3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad M_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathcal{M} \times_1 vec(A) \times_2 vec(B) = \mathcal{M} \times_1 \begin{bmatrix} A_{11} \\ A_{12} \\ A_{21} \\ A_{22} \end{bmatrix} \times_2 \begin{bmatrix} B_{11} \\ B_{12} \\ B_{21} \\ B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} \\ C_{21} \\ C_{12} \\ C_{22} \end{bmatrix} = vec(C^{\mathsf{T}})$$

CP-Decomposition of Tensors

Tensors can be decomposed into a sum of outer products:



$$= \sum_{i=1}^{R} \underbrace{A_i \circ B_i \circ C_i}_{\text{Outer Products}} = \underbrace{[\![A, B, C]\!]}_{\text{K Tensor}}$$

Since rank (number of columns) represent the number of multiplications, fewer outer products translate to faster algorithms:

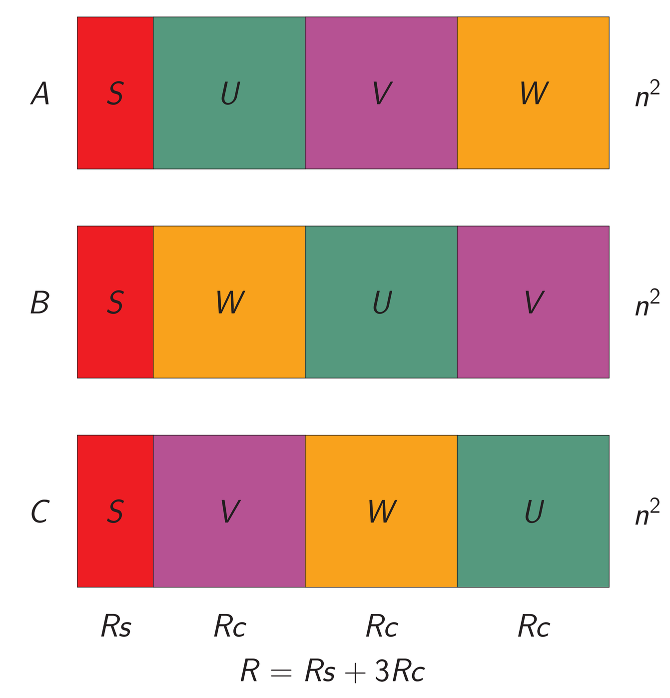
- 2 by 2
  - Rank 8: Classic Algorithm: $O(n^3)$
  - Rank 7: Strassen's Algorithm: $O(n^{\log_2 7}) \approx O(n^{2.81})$
  - Rank 6: Proven to be impossible
- 3 by 3
  - Rank 27: Classic Algorithm: $O(n^3)$
  - Rank 23: Current Best Algorithm: $O(n^{\log_3 23}) \approx O(n^{2.85})$
  - No proven lower bound
- 4 by 4
  - Rank 64: Classic Algorithm: $O(n^3)$
  - Rank 48: Recurse Strassen Twice: $O(n^{\log_4 48}) \approx O(n^{2.80})$
  - No proven lower bound
- 5 by 5
  - Rank 125: Classic Algorithm: $O(n^3)$
  - Rank 97: Flipgraph Algorithm: $O(n^{\log_5 97}) \approx O(n^{2.84})$
  - No proven lower bound

Cyclic Invariance in Matrix Multiplication

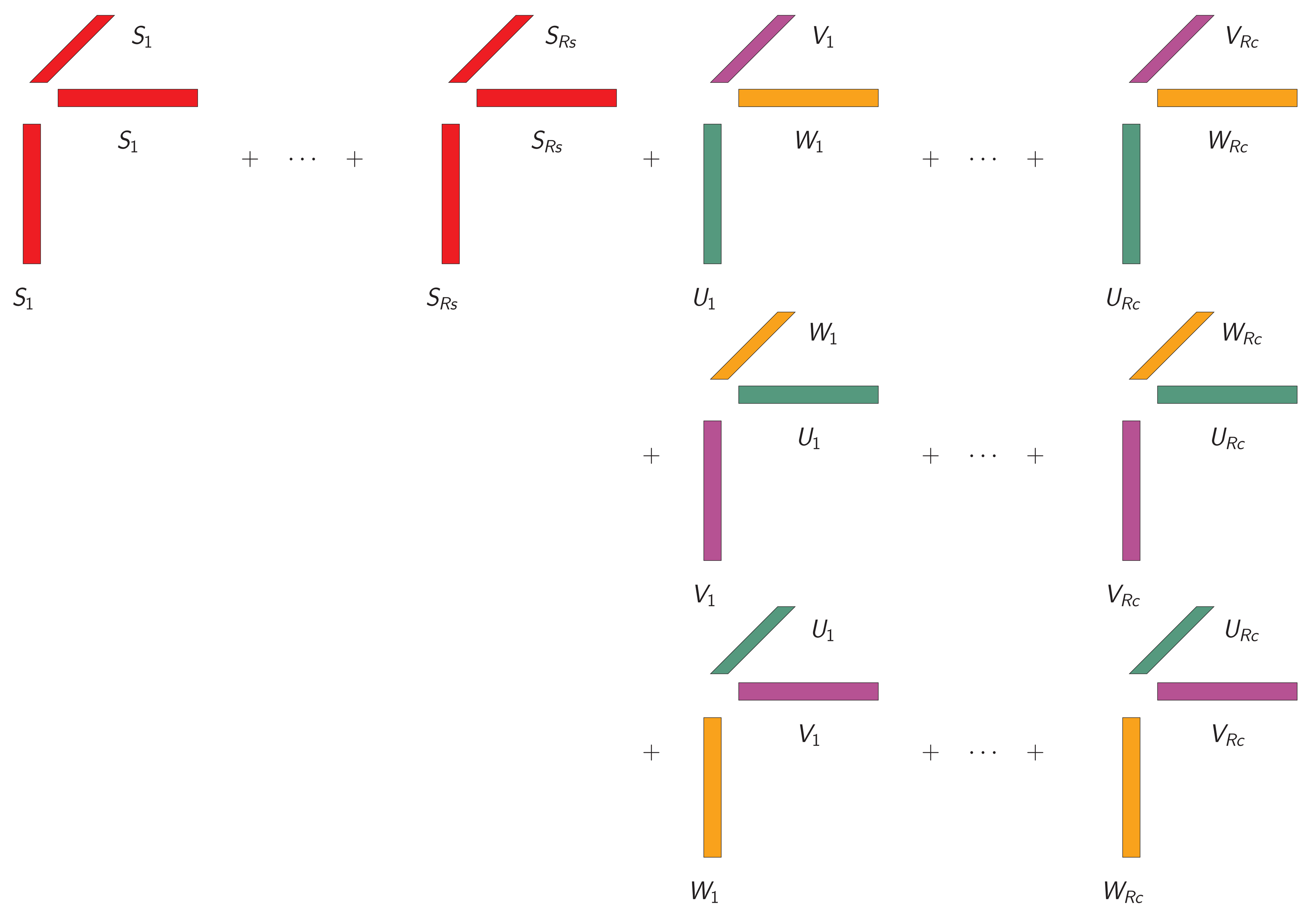
Some algorithms, like Strassen's, are cyclic invariant:

	M_1	M_2	M_3	M_4	M_5	M_6	M_7
A_{11}	1	0	0	0	1	1	−1
A_{12}	0	1	0	0	0	0	1
A_{21}	0	0	0	1	1	0	0
A_{22}	1	1	1	−1	0	0	0
B_{11}	1	1	−1	0	0	0	1
B_{12}	0	0	1	1	0	0	0
B_{21}	0	0	0	0	0	1	1
B_{22}	1	0	0	1	1	−1	0
C_{11}	1	0	1	1	−1	0	0
C_{21}	0	0	0	0	1	1	0
C_{12}	0	1	1	0	0	0	0
C_{22}	1	−1	0	0	0	1	1

We can use cyclic invariance in our favour to search for CP-decompositions of MatMulTensors with structure:



Then our Tensor Decomposition becomes:



Cyclic Invariant CP-Decomposition via Damped Gauss-Newton Optimization

Below is the generic algorithm, the input K is the vectorized version of whatever type of CP-Decomposition we wish to perform. $K = [\![A, B, C]\!]$ if we are performing regular CP-Decomposition, but $K = \{S, U, V, W\}$ if we are performing the Cyclic Invariant variation.

Data: Matrix Multiply Tensor $\mathcal{X}$
Initialize K randomly or through input arguments
Require: Damping Parameter $\lambda \in \mathbb{R}$,
Maximum Iterations $MaxIters \in \mathbb{Z}$,
Convergence Tolerance $\epsilon \in \mathbb{R}$;
Result: Decomposition K

```
for i = 1, ... , MaxIters do
    F_old = ⟵ Compute Function Value ∇F ⟵ Compute Gradient of Function
    M ⟵ Solution to (JᵀJ + λI)K = −∇F
    while Goldstein Conditions Are Satisfied do
        K = Kprev + α M
        F_new = ⟵ Compute Function Value
        α = α/2
    end
    if F_old - F_new < ϵ then
        break
    end
end
```

Figure: Generic Damped Gauss-Newton Algorithm

Modifying CP-DGN Algorithm

We can thus reduce the number of search parameters by 3 by substituting regular CP-Decomposition of factor matrices (A, B, C), with smaller cyclic matrices (S, U, V, W) and substitute all operations accordingly.

Minimizing Functions:

$$min\ f(A, B, C) = \frac{1}{2} \| \mathcal{M} - [\![A, B, C]\!] \|^2$$

$$\Downarrow$$

$$min\ f(S, U, V, W) = \frac{1}{2} \| \mathcal{M} - [\![S, S, S]\!] - [\![U, V, W]\!] - [\![W, U, V]\!] - [\![V, W, U]\!] \|^2$$

Gradient:

$$\nabla \mathbf{f} = [\text{vec}(\frac{\partial f}{\partial A})^{\mathsf{T}}\ \text{vec}(\frac{\partial f}{\partial B})^{\mathsf{T}}\ \text{vec}(\frac{\partial f}{\partial C})^{\mathsf{T}}\ ]^{\mathsf{T}} \qquad \nabla \mathbf{f} = [\text{vec}(\frac{\partial f}{\partial S})^{\mathsf{T}}\ \text{vec}(\frac{\partial f}{\partial U})^{\mathsf{T}}\ \text{vec}(\frac{\partial f}{\partial V})^{\mathsf{T}}\ \text{vec}(\frac{\partial f}{\partial W})^{\mathsf{T}}]^{\mathsf{T}}$$

$$\begin{aligned}
\frac{\partial f}{\partial A} &= -M_{(1)}(C \circ B) + A(C^{\mathsf{T}}C * B^{\mathsf{T}}B) \\
\frac{\partial f}{\partial A} &= -M_{(2)}(C \circ A) + B(C^{\mathsf{T}}B * A^{\mathsf{T}}A) \\
\frac{\partial f}{\partial A} &= -M_{(3)}(B \circ A) + C(B^{\mathsf{T}}B * A^{\mathsf{T}}A)
\end{aligned}
\quad \Longrightarrow \quad
\begin{aligned}
\frac{\partial f}{\partial U} = \ & 3\Big(S\big((S^{\mathsf{T}}V) * (S^{\mathsf{T}}W)\big) + U\big((V^{\mathsf{T}}V) * (W^{\mathsf{T}}W)\big) \\
& + V\big((W^{\mathsf{T}}V) * (U^{\mathsf{T}}W)\big) + W\big((U^{\mathsf{T}}V) * (V^{\mathsf{T}}W)\big)\Big) \\
& - M_{(1)}(V \circ W) - M_{(2)}(W \circ V) - M_{(3)}(V \circ W)
\end{aligned}$$

Jacobian:

$$\mathbf{J} = \begin{bmatrix} J_A & J_B & J_C \end{bmatrix} \in \mathbb{R}^{n^3 \times nR)} \qquad \mathbf{J} = \begin{bmatrix} J_s & J_u & J_v & J_w \end{bmatrix} \in \mathbb{R}^{n^3 \times n(R_s + 3R_c)}$$

$$\begin{aligned}
J_A &= -(C \odot B) \otimes I \\
J_B &= -\Pi_2^{\mathsf{T}} \cdot ((C \odot A) \otimes I) \\
J_C &= -\Pi_3^{\mathsf{T}} \cdot ((B \odot A) \otimes I)
\end{aligned}
\quad \Longrightarrow \quad
\begin{aligned}
J_s &= (S \odot S) \otimes I_n + \Pi_2^{\mathsf{T}} \cdot (S \odot S) \otimes I_n + \Pi_3^{\mathsf{T}} \cdot (S \odot S) \otimes I_n \\
J_u &= (V \odot W) \otimes I_n + \Pi_2^{\mathsf{T}} \cdot (W \odot V) \otimes I_n + \Pi_3^{\mathsf{T}} \cdot (V \odot W) \otimes I_n \\
J_v &= (W \odot U) \otimes I_n + \Pi_2^{\mathsf{T}} \cdot (U \odot W) \otimes I_n + \Pi_3^{\mathsf{T}} \cdot (W \odot U) \otimes I_n \\
J_w &= (U \odot V) \otimes I_n + \Pi_2^{\mathsf{T}} \cdot (V \odot U) \otimes I_n + \Pi_3^{\mathsf{T}} \cdot (U \odot V) \otimes I_n
\end{aligned}$$

Applying $(\mathbf{J}^{\mathsf{T}}\mathbf{J} + \lambda \mathbf{I})$ to vectorized input

$$(J^{\mathsf{T}}J + \lambda I) \cdot vec(K) = \begin{bmatrix} J_A^{\mathsf{T}}J_A + \lambda & J_A^{\mathsf{T}}J_B & J_A^{\mathsf{T}}J_C \\ J_B^{\mathsf{T}}J_A & J_B^{\mathsf{T}}J_A + \lambda & J_B^{\mathsf{T}}J_C \\ J_C^{\mathsf{T}}J_A & J_C^{\mathsf{T}}J_u & J_C^{\mathsf{T}}J_C + \lambda \end{bmatrix} \begin{bmatrix} vec(K_A) \\ vec(K_B) \\ vec(K_C) \end{bmatrix}$$

$$J_B^{\mathsf{T}}J_A vec(K_s) = vec\Big(B(K_A^{\mathsf{T}}A * C^{\mathsf{T}}C)\Big)$$

$$\Downarrow$$

$$(J^{\mathsf{T}}J + \lambda I) \cdot vec(K) = \begin{bmatrix} J_s^{\mathsf{T}}J_s + \lambda & J_s^{\mathsf{T}}J_u & J_s^{\mathsf{T}}J_v & J_s^{\mathsf{T}}J_w \\ J_u^{\mathsf{T}}J_s & J_u^{\mathsf{T}}J_u + \lambda & J_u^{\mathsf{T}}J_v & J_u^{\mathsf{T}}J_w \\ J_v^{\mathsf{T}}J_s & J_v^{\mathsf{T}}J_u & J_v^{\mathsf{T}}J_v + \lambda & J_v^{\mathsf{T}}J_w \\ J_w^{\mathsf{T}}J_s & J_w^{\mathsf{T}}J_u & J_w^{\mathsf{T}}J_v & J_w^{\mathsf{T}}J_w + \lambda \end{bmatrix} \begin{bmatrix} vec(K_s) \\ vec(K_u) \\ vec(K_v) \\ vec(K_w) \end{bmatrix}$$

$$J_u^{\mathsf{T}}J_s vec(K_s) = 3vec\Big(K_s(S^{\mathsf{T}}V) * (S^{\mathsf{T}}W) + S\big((K_s^{\mathsf{T}}W) * (S^{\mathsf{T}}V) + (K_s^{\mathsf{T}}V) * (S^{\mathsf{T}}W)\big)\Big)$$

Results

Solutions We have found so far

- 2 by 2 - Rank 7
  - (Rs = 4, Rc = 1)
- 3 by 3 - Rank 23
  - (Rs = 11, Rc = 4)
  - (Rs = 8. Rc = 5) - Evidence for border rank solution
  - (Rs = 5, Rc = 6)
  - (Rs = 2, Rc = 7)
- 4 by 4 - Rank 49
  - (Rs = 16, Rc = 11)
  - (Rs = 1, Rc = 26)
- 5 by 5 - Rank 99
  - (Rs = 18, Rc = 27) - Evidence for border rank solution

Future Work

- Only began decomposing MatMul 5 recently, so there is more to explore
- Explore new auxiliary functions that penalize search based on input heuristic

References & Github Repository

1. Rouse, Kathryn Z., and Grey M. Ballard. "On the Efficiency of Algorithms for Tensor Decompositions and Their Applications." Wake Forest University, 2018. Print.
2. G. Ballard, C. Ikenmeyer, J. Landsberg and N. Ryder, The geometry of rank decompositions of matrix multiplication II: 3x3 matrices, Journal of Pure and Applied Algebra, Volume 223, Number 8, pp. 3205 - 3224, 2018.
3. https://github.com/Jv7Pinheiro/FastMatrixMultiplyAlgorithmsSearch