Fast Matrix Multiplication Without Tears: A Constraint Programming Approach

Arnaud Deza, Chang Liu, Pashootan Vaezipoor, Elias B. Khalil

Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

CP 2023
TORONTO, CANADA

Standard matrix multiplication

*Multiplication of 2×2 matrices with **8** scalar multiplications*

$$\overset{\textbf{C}}{\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix}} = \overset{\textbf{A}}{\begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix}} \cdot \overset{\textbf{B}}{\begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}}$$

$$\begin{aligned}
m_1 &= a_1 \cdot b_1 \\
m_2 &= a_2 \cdot b_3 \\
m_3 &= a_1 \cdot b_2 \\
m_4 &= a_2 \cdot b_4 \\
m_5 &= a_3 \cdot b_1 \\
m_6 &= a_4 \cdot b_3 \\
m_7 &= a_3 \cdot b_2 \\
m_8 &= a_4 \cdot b_4
\end{aligned}$$

$$\boxed{\begin{aligned}
c_1 &= m_1 + m_2 \\
c_2 &= m_3 + m_4 \\
c_3 &= m_5 + m_6 \\
c_4 &= m_7 + m_8
\end{aligned}}$$

Fast Matrix Multiplication

*Strassen (1969): Multiplication of 2×2 matrices in **7** scalar multiplications (Optimal)*

$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$

$$\begin{aligned}
m_1 &= (a_1 + a_4) \cdot (b_1 + b_4) \\
m_2 &= (a_3 + a_4) \cdot (b_1) \\
m_3 &= (a_1) \cdot (b_2 - b_4) \\
m_4 &= (a_4) \cdot (b_3 - b_1) \\
m_5 &= (a_1 + a_2) \cdot (b_4) \\
m_6 &= (a_3 - a_1) \cdot (b_1 + b_2) \\
m_7 &= (a_2 - a_4) \cdot (b_3 + b_4)
\end{aligned}$$

$$\begin{aligned}
c_1 &= m_1 + m_4 - m_5 + m_7 \\
c_2 &= m_3 + m_5 \\
&= (a_1)(b_2 - b_4) + (a_1 + a_2)(b_4) \\
&= a_1 b_2 - \cancel{a_1 b_4} + \cancel{a_1 b_4} + a_2 b_4 \\
&= a_1 b_2 + a_2 b_4 \\
c_3 &= m_2 + m_4 \\
c_4 &= m_1 - m_2 + m_3 + m_6
\end{aligned}$$

Standard vs. fast MM

Standard Matrix Multiplication

$$m_1 = a_1 \cdot b_1$$
$$m_2 = a_2 \cdot b_3$$
$$m_3 = a_1 \cdot b_2$$
$$m_4 = a_2 \cdot b_4$$
$$m_5 = a_3 \cdot b_1$$
$$m_6 = a_4 \cdot b_3$$
$$m_7 = a_3 \cdot b_2$$
$$m_8 = a_4 \cdot b_4$$

$$c_1 = m_1 + m_2$$
$$c_2 = m_3 + m_4$$
$$c_3 = m_5 + m_6$$
$$c_4 = m_7 + m_8$$

Strassen's Algorithm

$$m_1 = (a_1 + a_4) \cdot (b_1 + b_4)$$
$$m_2 = (a_3 + a_4) \cdot (b_1)$$
$$m_3 = (a_1) \cdot (b_2 - b_4)$$
$$m_4 = (a_4) \cdot (b_3 - b_1)$$
$$m_5 = (a_1 + a_2) \cdot (b_4)$$
$$m_6 = (a_3 - a_1) \cdot (b_1 + b_2)$$
$$m_7 = (a_2 - a_4) \cdot (b_3 + b_4)$$

$$c_1 = m_1 + m_4 - m_5 + m_7$$
$$c_2 = m_3 + m_5$$
$$c_3 = m_2 + m_4$$
$$c_4 = m_1 - m_2 + m_3 + m_6$$

Multiplication is a ***more expensive*** *arithmetic operation than addition, making Strassen's algorithm more desirable.*

Why are we studying this problem?

Matrix multiplication is a fundamental operation in linear algebra with applications in essentially all computations.

- ***Mathematically***, there are still open problems:
  - To multiply two 3×3 matrices, the best known algorithm requires $R = \mathbf{23}$ multiplications (compared to the naive 27) but a theoretical lower bound of $R = \mathbf{19}$ exists. Are there any algorithms with $19 \leq R \leq 22$ or is $R = \mathbf{23}$ optimal?

- ***Computationally***, while our focus is on small matrices, faster multiplication algorithms for larger matrices can be obtained by recursive application of fast small matrix multiplication.
  - Note that this may have numerical stability issues[1].

[1] *https://nhigham.com/2022/09/13/what-is-fast-matrix-multiplication/*

Tensor representation of matrix multiplication

$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$

Binary matrix representing the $a_i b_j$ terms that appear in c_1

$$T_{:,:,1} = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{array} \begin{array}{c} \begin{array}{cccc} b_1 & b_2 & b_3 & b_4 \end{array} \\ \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$c_1 = a_1 b_1 + a_2 b_3$$

$$T_{:,:,2} = \begin{pmatrix} 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad T_{:,:,3} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \end{pmatrix} \quad T_{:,:,4} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} \end{pmatrix}$$

Strassen's algorithm, tensorized

$$m_1 = (a_1 + a_4) \cdot (b_1 + b_4)$$
$$m_2 = (a_3 + a_4) \cdot (b_1)$$
$$m_3 = (a_1) \cdot (b_2 - b_4)$$
$$m_4 = (a_4) \cdot (b_3 - b_1)$$
$$m_5 = (a_1 + a_2) \cdot (b_4)$$
$$m_6 = (a_3 - a_1) \cdot (b_1 + b_2)$$
$$m_7 = (a_2 - a_4) \cdot (b_3 + b_4)$$

$$c_1 = m_1 + m_4 - m_5 + m_7$$
$$c_2 = m_3 + m_5$$
$$c_3 = m_2 + m_4$$
$$c_4 = m_1 - m_2 + m_3 + m_6$$

Factor Matrices

$$U = \begin{array}{c} \begin{array}{ccccccc} m_1 & m_2 & m_3 & m_4 & m_5 & m_6 & m_7 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \end{array} \begin{array}{c} a_1 \\ a_2 \\ a_3 \\ a_4 \end{array}$$

$$V = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{array}{c} b_1 \\ b_2 \\ b_3 \\ b_4 \end{array}$$

$$W = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{array}{c} c_1 \\ c_2 \\ c_3 \\ c_4 \end{array}$$

Matrix multiplication algorithm ≡ low-rank decomposition of tensor T

Strassen's Algorithm

Factor Matrices

$$U = \begin{pmatrix} \boxed{1} & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$V = \begin{pmatrix} \boxed{1} & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$W = \begin{pmatrix} \boxed{1} & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$T_{:,:,1} = \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$T_{1,1,1} = 1$$

$$= \sum_{r=1}^{7} u_{1,r} \cdot v_{1,r} \cdot w_{1,r}$$

$$= 1 + 0 + 0 + 0 + 0 + 0 + 0$$

Matrix multiplication algorithm ≡ low-rank decomposition of tensor T

Strassen's Algorithm

Factor Matrices

$$U = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ \boxed{1} & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$V = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ \boxed{1} & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$W = \begin{pmatrix} \boxed{1} & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$T_{:,:,1} = \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$T_{4,4,1} = 0$$

$$= \sum_{r=1}^{7} u_{4,r} \cdot v_{4,r} \cdot w_{1,r}$$

$$= 1 + 0 + 0 + 0 + 0 + 0 - 1$$

Matrix multiplication algorithm ≡ low-rank decomposition of tensor T

Strassen's Algorithm

Factor Matrices

$$U = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$V = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$W = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$T_{:,:,1} = \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$T_{1,1,1} = 1$$

$$= \sum_{r=1}^{7} u_{1,r} \cdot v_{1,r} \cdot w_{1,r}$$

In general,

$$T_{i,j,k} = \sum_{r=1}^{7} u_{i,r} \cdot v_{j,r} \cdot w_{k,r}$$

Fast Matrix Multiplication ≡ **Lowest**-rank decomposition of tensor T

≡ *Construct U, V, W matrices with as few columns as possible!*

Strassen's Algorithm

Factor Matrices

$$U = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$V = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$W = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$T_{:,:,1} = \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$T_{1,1,1} = 1$$

$$= \sum_{r=1}^{7} u_{1,r} \cdot v_{1,r} \cdot w_{1,r}$$

In general,

$$T_{i,j,k} = \sum_{r=1}^{7} u_{i,r} \cdot v_{j,r} \cdot w_{k,r}$$

Constraint Programming model

A multiplication of two matrices **A** and **B** of sizes $N \times M$ and $M \times P$, respectively, results in a product matrix **C** of size $N \times P$. With a given rank R, we have the following CP model:

$$\sum_{r=1}^{R} (u_{i,r} \cdot v_{j,r} \cdot w_{k,r}) = T_{i,j,k}\,, \qquad \forall i \in \mathscr{U}, j \in \mathscr{V}, k \in \mathscr{W}$$

$$u_{i,r}\,, v_{j,r}\,, w_{k,r} \in \{-1,0,1\} \qquad \forall i \in \mathscr{U}, j \in \mathscr{V}, k \in \mathscr{W}, r \in \mathscr{R}$$

where, $\mathscr{U} := \{1,\ldots,N \cdot M\}, \mathscr{V} := \{1,\ldots,M \cdot P\}, \mathscr{W} := \{1,\ldots,N \cdot P\}$ and $\mathscr{R} := \{1,\ldots,R\}$

- For (N, M, P, R) = (2, 2, 2, 7), there are 84 variables and the solution space has size 1.19×10^{40}
- For (N, M, P, R) = (3, 3, 3, 23), there are 621 variables and the solution space has size 1.96×10^{296}

Note: {-1,0,1} is a common domain that is searched for, some search for {-2,..,2}

Existing *inexact* approaches

Deep Reinforcement Learning (AlphaTensor)

- constructing factor matrices as a sequential game; reward for matching new entries of tensor T
- first to directly incorporate reinforcement learning into the search in FMM
- discovery of new minimal ranks (largest is for 5×5 matrices, $R = 98$)

Limitations:

- undocumented/large training/running time
- has likely reached its limits

Alhussein Fawzi, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. Nature, 610(7930):47–53, 2022.

Continuous Local Search

$$\min \sum_{i \in \mathcal{U}} \sum_{j \in \mathcal{V}} \sum_{k \in \mathcal{W}} \left(\sum_{r \in \mathcal{R}} u_{i,r} \cdot v_{j,r} \cdot w_{k,r} - T_{i,j,k} \right)^2$$

- most successful in finding fast algorithms whilst remaining computationally tractable, scaled up to (3×4) by (4×5), $R = 48$

Limitations:

- getting stuck at local minima
- continuous variables → need to round them

Alexey V. Smirnov. The bilinear complexity and practical algorithms for matrix multiplication. Computational Mathematics and Mathematical Physics, 53:1781 – 1795, 2013.

Existing *exact* approaches

Integer Programming

- focus on presenting the MILP formulation for square matrices
- linearize the trilinear products in the low-rank decomposition of T_n to a MILP
- the MILP formulations for $N \in \{2,3\}$ are benchmark problems in MIPLIB 2017

Limitations:

- no computational experiments reported
- linearization via Big-M → weak LP relaxation

Laurent Sorber and Marc Van Barel. A mixed-integer linear program formulation for fast matrix multiplication, 2017.

SAT-based methods

- transform the multiplication to "*and*" clauses and the addition to "*xor*" clauses
- find new general solutions from previously found solutions using local search
- found many thousands new general algorithms for 3×3 matrices with rank 23

Limitations:

- difficult for complete SAT-solvers

Marijn JH Heule, Manuel Kauers, and Martina Seidl. New ways to multiply 3×3-matrices. Journal of Symbolic Computation, 2021.

Experimental setup

Base CP Model

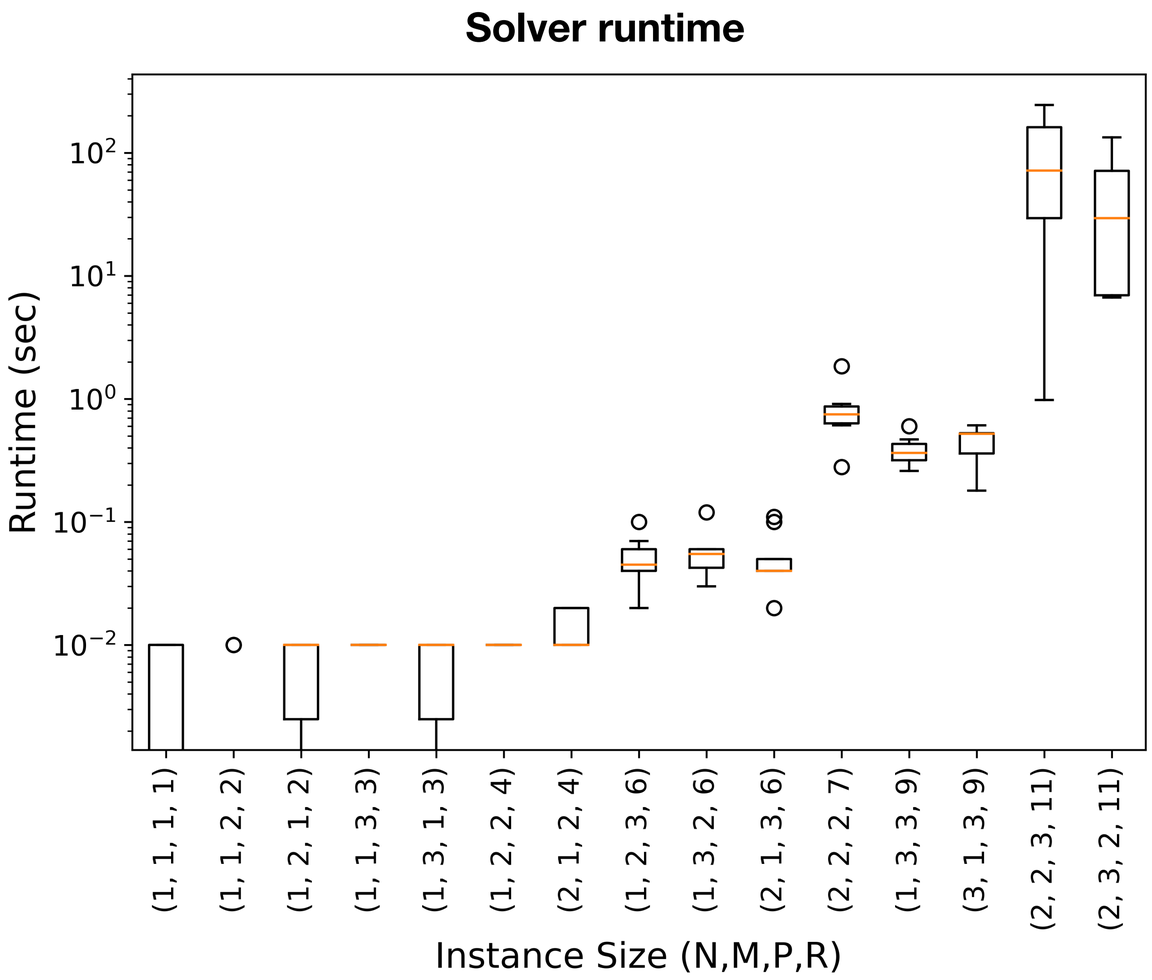
$$\sum_{r=1}^{R} (u_{i,r} \cdot v_{j,r} \cdot w_{k,r}) = T_{i,j,k}\,, \qquad \forall i \in \mathscr{U}, j \in \mathscr{V}, k \in \mathscr{W}$$

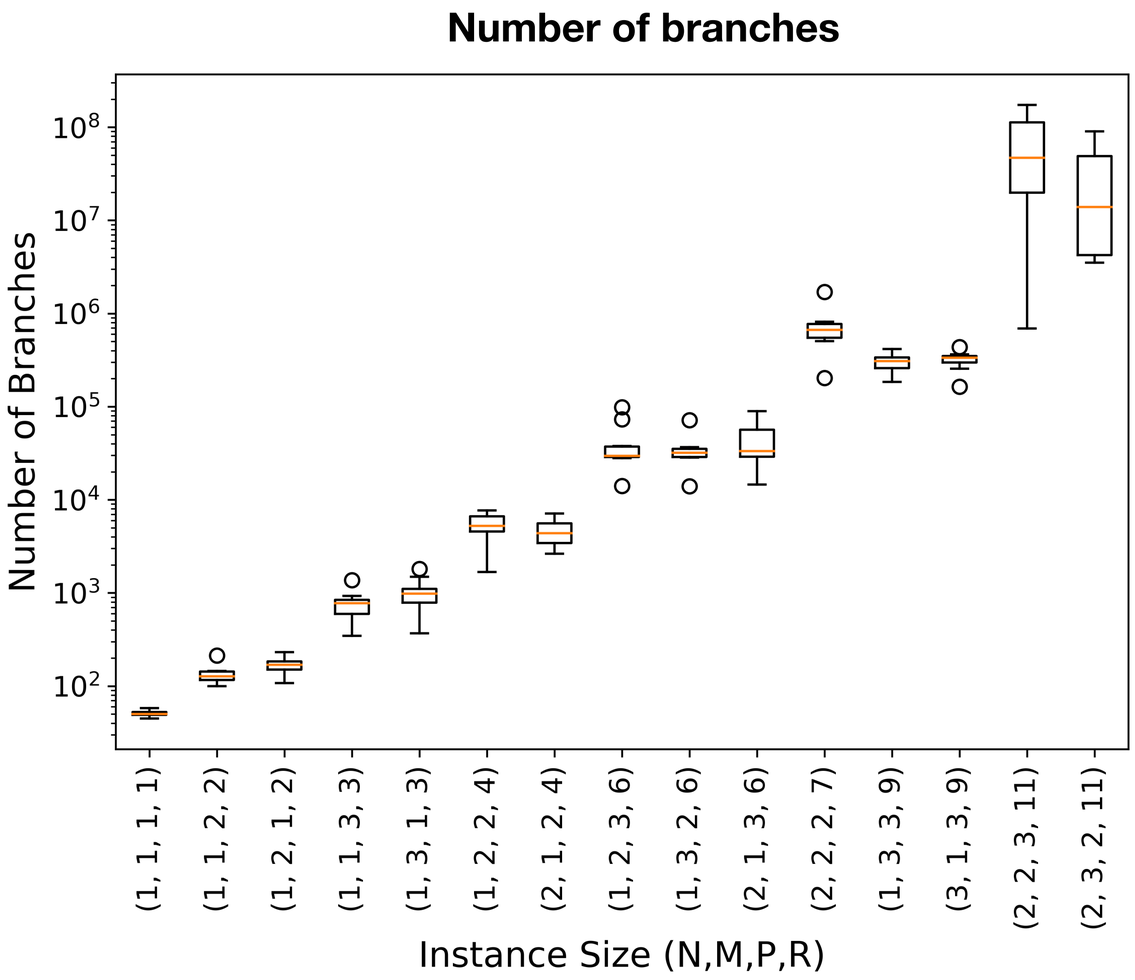
$$u_{i,r}\,, v_{j,r}\,, w_{k,r} \in \{-1,0,1\} \qquad \forall i \in \mathscr{U}, j \in \mathscr{V}, k \in \mathscr{W}, r \in \mathscr{R}$$

where, $\mathscr{U} := \{1,\ldots,N \cdot M\}, \mathscr{V} := \{1,\ldots,M \cdot P\}, \mathscr{W} := \{1,\ldots,N \cdot P\}$ and $\mathscr{R} := \{1,\ldots,R\}$

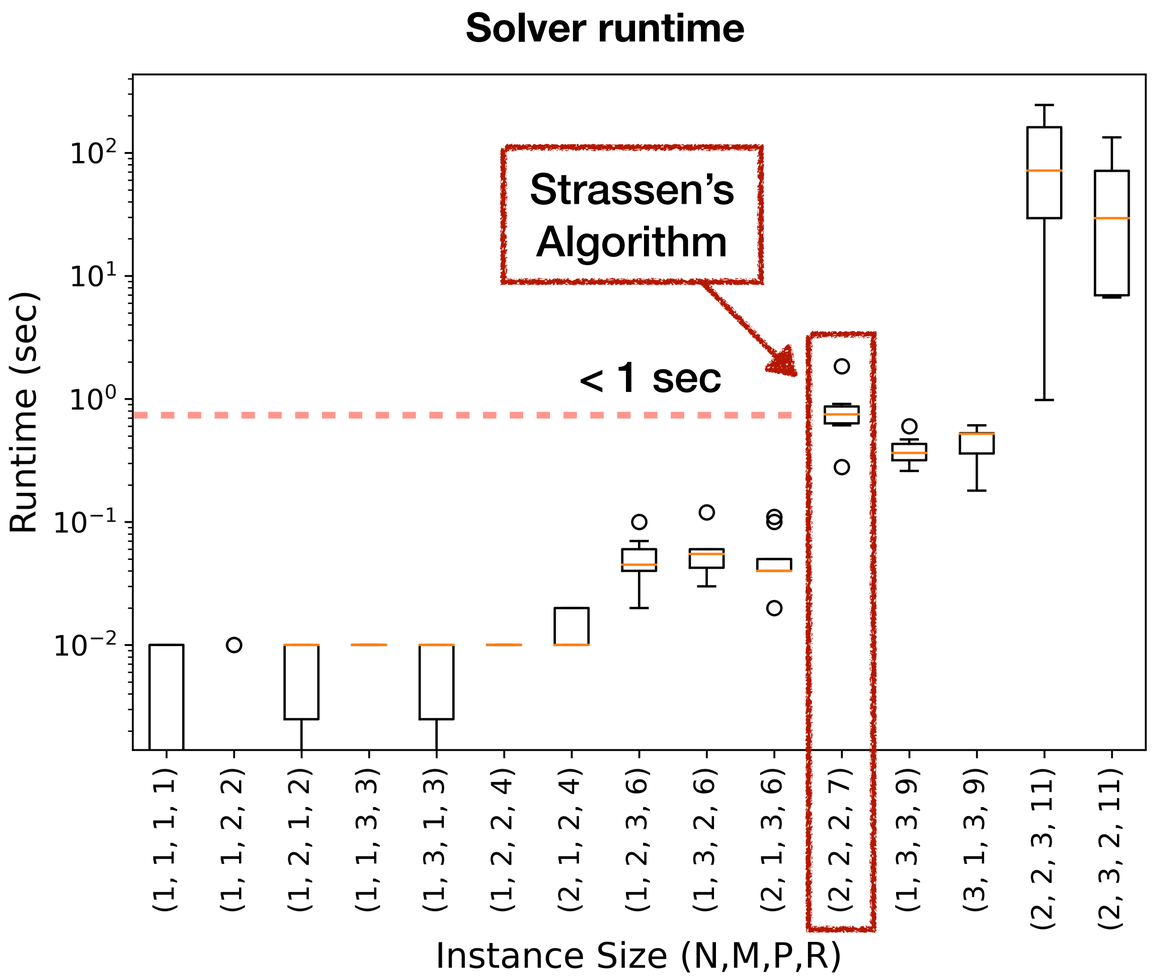
- constraints involve sum of products of 3 integer variables
- solved with **IBM ILOG CP Optimizer (CPO) 22.1.0**
- ran our experiments with **10 different seeds**
- time limit of **2 hours** (for the most part)

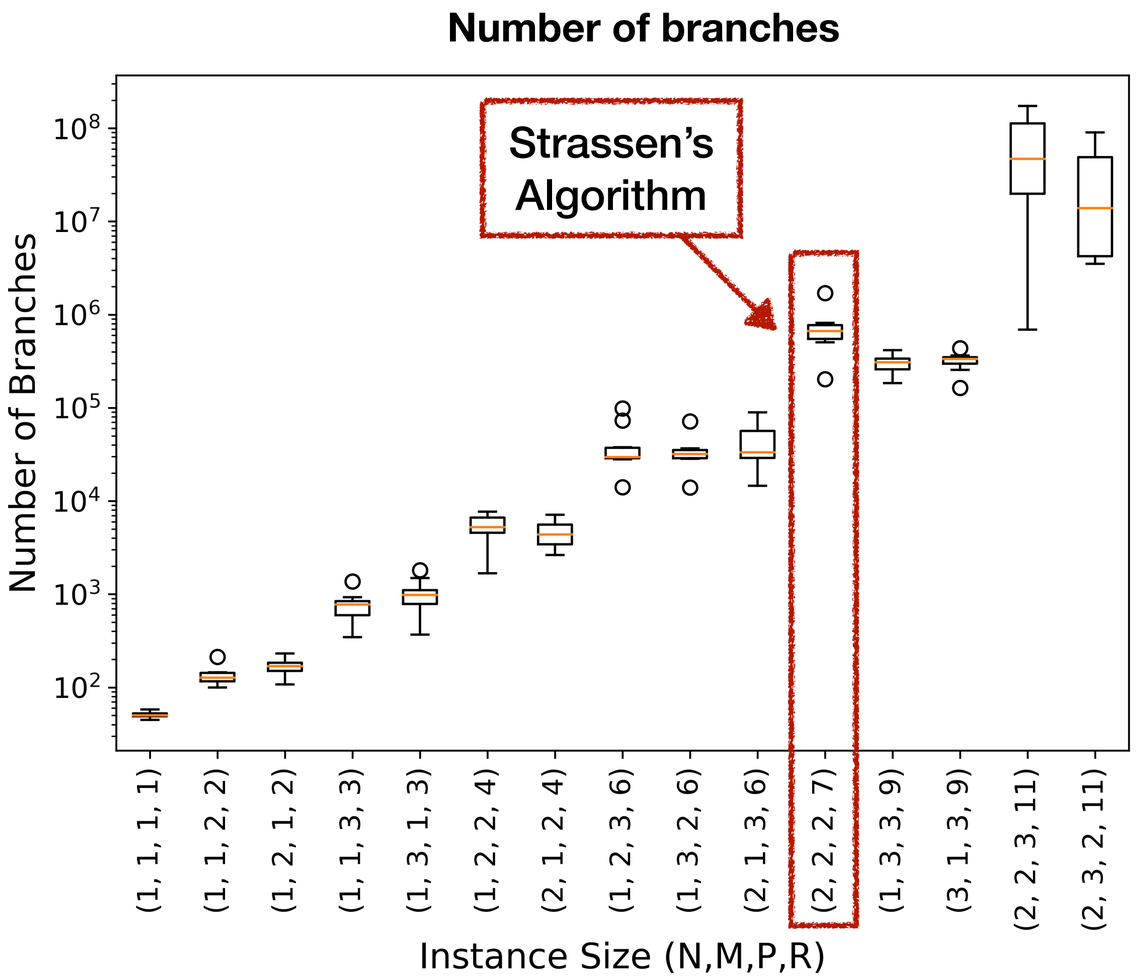
Experimental results for feasible cases



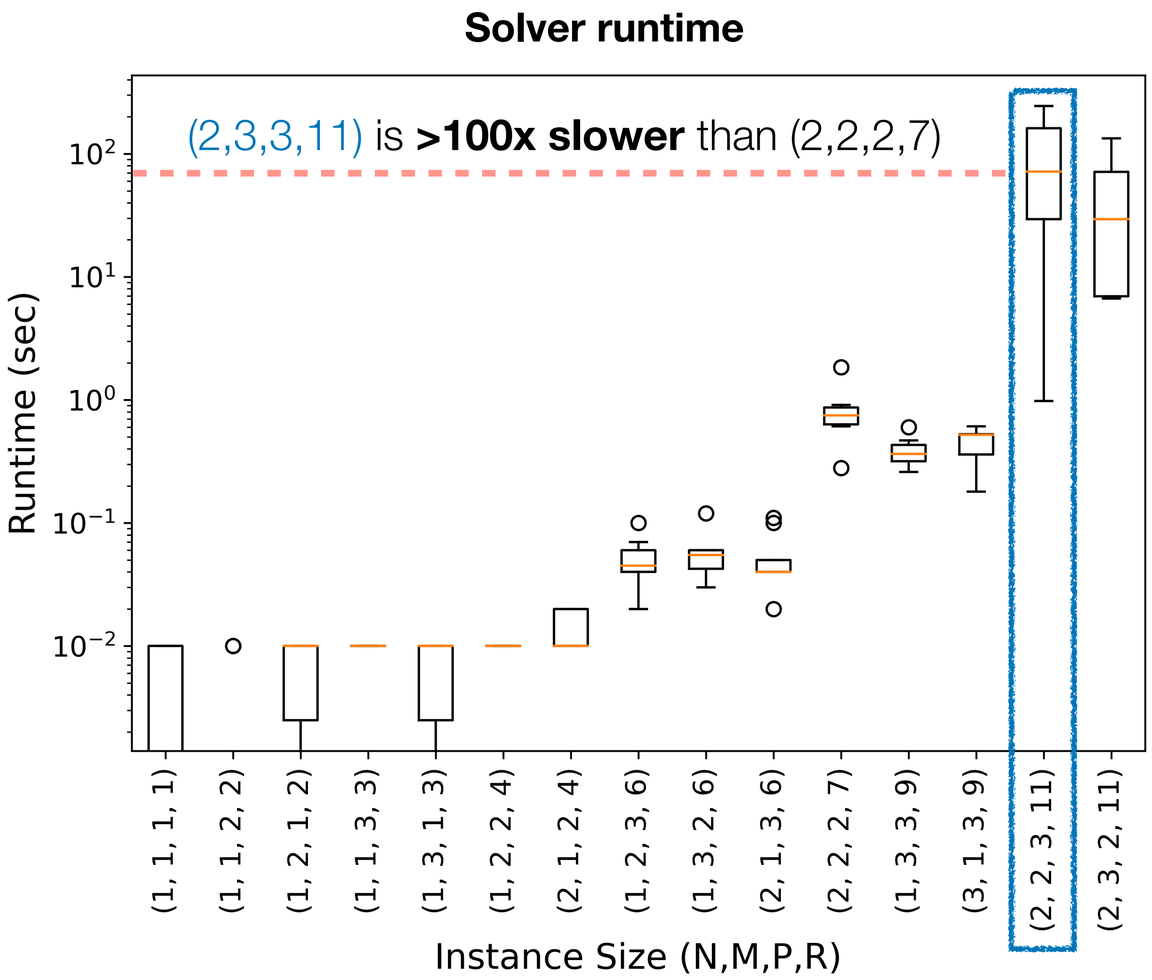


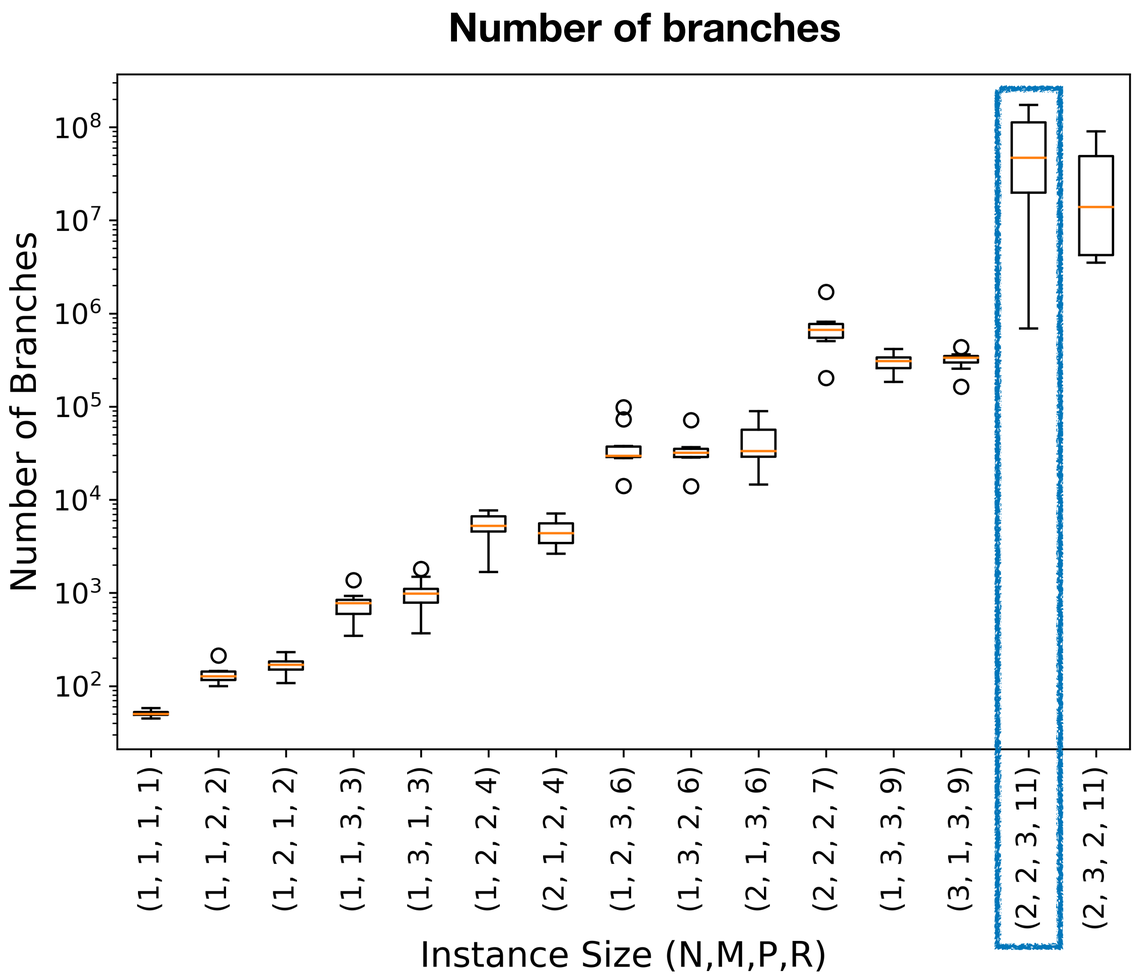
Experimental results for feasible cases





Experimental results for feasible cases





Solving larger instances: a sparsity-based decomposition

Strassen's Algorithm

$$U = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$V = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$W = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$m_1 = (a_1 + a_4) \cdot (b_1 + b_4)$$
$$m_2 = (a_3 + a_4) \cdot (b_1)$$
$$m_3 = (a_1) \cdot (b_2 - b_4)$$
$$m_4 = (a_4) \cdot (b_3 - b_1)$$
$$m_5 = (a_1 + a_2) \cdot (b_4)$$
$$m_6 = (a_3 - a_1) \cdot (b_1 + b_2)$$
$$m_7 = (a_2 - a_4) \cdot (b_3 + b_4)$$
$$c_1 = m_1 + m_4 - m_5 + m_7$$
$$c_2 = m_3 + m_5$$
$$c_3 = m_2 + m_4$$
$$c_4 = m_1 - m_2 + m_3 + m_6$$

Solving larger instances: a sparsity-based decomposition

Strassen's Algorithm

$$U = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$V = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$W = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$m_1 = (a_1 + a_4) \cdot (b_1 + b_4)$$
$$m_2 = (a_3 + a_4) \cdot (b_1)$$
$$m_3 = (a_1) \cdot (b_2 - b_4)$$
$$m_4 = (a_4) \cdot (b_3 - b_1)$$
$$m_5 = (a_1 + a_2) \cdot (b_4)$$
$$m_6 = (a_3 - a_1) \cdot (b_1 + b_2)$$
$$m_7 = (a_2 - a_4) \cdot (b_3 + b_4)$$
$$c_1 = m_1 + m_4 - m_5 + m_7$$
$$c_2 = m_3 + m_5$$
$$c_3 = m_2 + m_4$$
$$c_4 = m_1 - m_2 + m_3 + m_6$$

Many zeros in the solution

Solving larger instances: a sparsity-based decomposition

$$U = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$V = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$W = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Many zeros in the solution

No m term uses more than 2 out of 4 of the a or b terms

$$\begin{aligned}
m_1 &= \overbrace{(a_1 + a_4)} \cdot \overbrace{(b_1 + b_4)} \\
m_2 &= (a_3 + a_4) \cdot (b_1) \\
m_3 &= (a_1) \cdot (b_2 - b_4) \\
m_4 &= (a_4) \cdot (b_3 - b_1) \\
m_5 &= (a_1 + a_2) \cdot (b_4) \\
m_6 &= (a_3 - a_1) \cdot (b_1 + b_2) \\
m_7 &= (a_2 - a_4) \cdot (b_3 + b_4) \\
c_1 &= m_1 + m_4 - m_5 + m_7 \\
c_2 &= m_3 + m_5 \\
c_3 &= m_2 + m_4 \\
c_4 &= \underbrace{m_1 - m_2 + m_3 + m_6}
\end{aligned}$$

No c term uses more than 4 out of the 7 m terms

Solving larger instances: a sparsity-based decomposition

$$
\begin{array}{cc}
 & R \\
U = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} & NM \\
V = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix} & MP \\
W = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} &
\end{array}
$$

Let $K_1 \leq NM + MP$ be an upper bound on the sum of the number of nonzeros in each column of **U, V.**

$$
\begin{aligned}
m_1 &= \overbrace{(a_1 + a_4)} \cdot \overbrace{(b_1 + b_4)} \\
m_2 &= (a_3 + a_4) \cdot (b_1) \\
m_3 &= (a_1) \cdot (b_2 - b_4) \\
m_4 &= (a_4) \cdot (b_3 - b_1) \\
m_5 &= (a_1 + a_2) \cdot (b_4) \\
m_6 &= (a_3 - a_1) \cdot (b_1 + b_2) \\
m_7 &= (a_2 - a_4) \cdot (b_3 + b_4) \\
c_1 &= m_1 + m_4 - m_5 + m_7 \\
c_2 &= m_3 + m_5 \\
c_3 &= m_2 + m_4 \\
c_4 &= \underbrace{m_1 - m_2 + m_3 + m_6}
\end{aligned}
$$

Let $K_2 \leq R$ be an upper bound on the number of m terms used in each c term.

Solving larger instances: a sparsity-based decomposition

$$
\begin{array}{cc}
 & R \\
U = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} & NM \\
V = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix} & MP \\
W = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} &
\end{array}
$$

Let $K_1 \leq NM + MP$ be an upper bound on the sum of the number of nonzeros in each column of **U, V.**

$$
\begin{aligned}
m_1 &= \overbrace{(a_1 + a_4)} \cdot \overbrace{(b_1 + b_4)} \\
m_2 &= (a_3 + a_4) \cdot (b_1) \\
m_3 &= (a_1) \cdot (b_2 - b_4) \\
m_4 &= (a_4) \cdot (b_3 - b_1) \\
m_5 &= (a_1 + a_2) \cdot (b_4) \\
m_6 &= (a_3 - a_1) \cdot (b_1 + b_2) \\
m_7 &= (a_2 - a_4) \cdot (b_3 + b_4) \\
c_1 &= m_1 + m_4 - m_5 + m_7 \\
c_2 &= m_3 + m_5 \\
c_3 &= m_2 + m_4 \\
c_4 &= \underbrace{m_1 - m_2 + m_3 + m_6}
\end{aligned}
$$

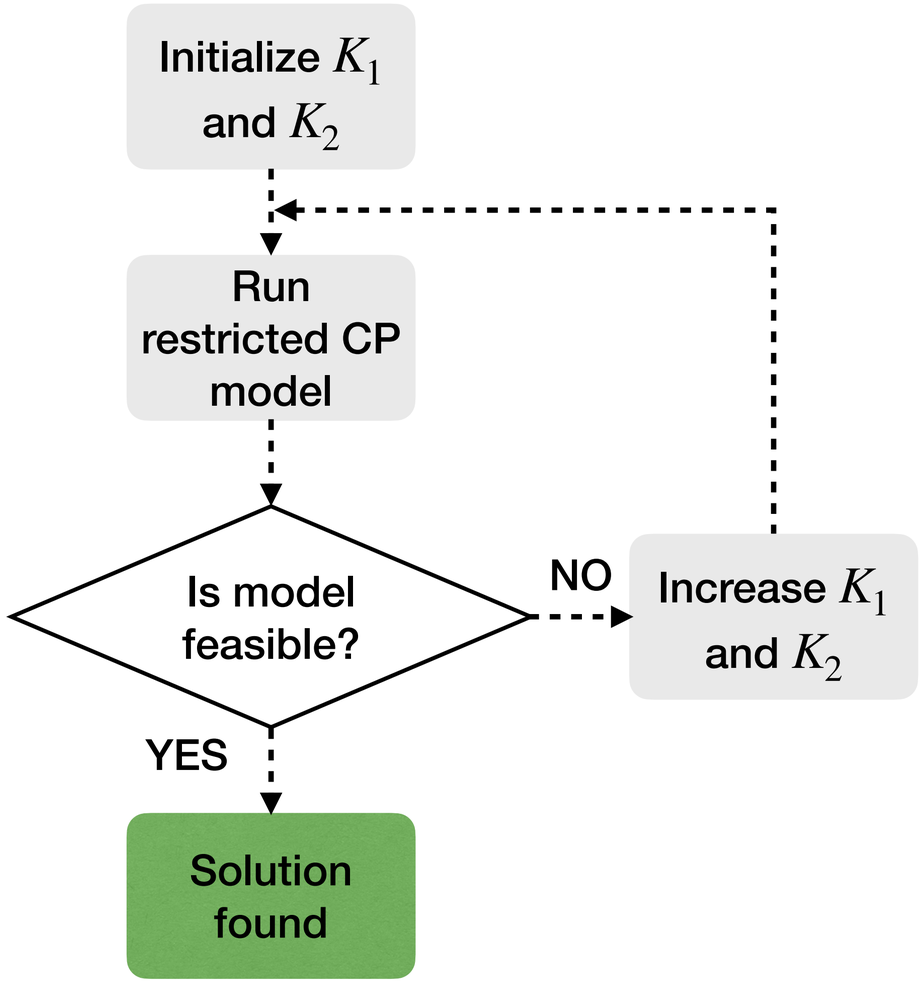
Let $K_2 \leq R$ be an upper bound on the number of m terms used in each c term.

For Strassen's 2x2 algorithm, the following are valid:

$$K_1 = 4 \leq 8 = 2 \cdot 2 + 2 \cdot 2$$

$$K_2 = 4 \leq 7$$

Sparsity-based decomposition: implementation



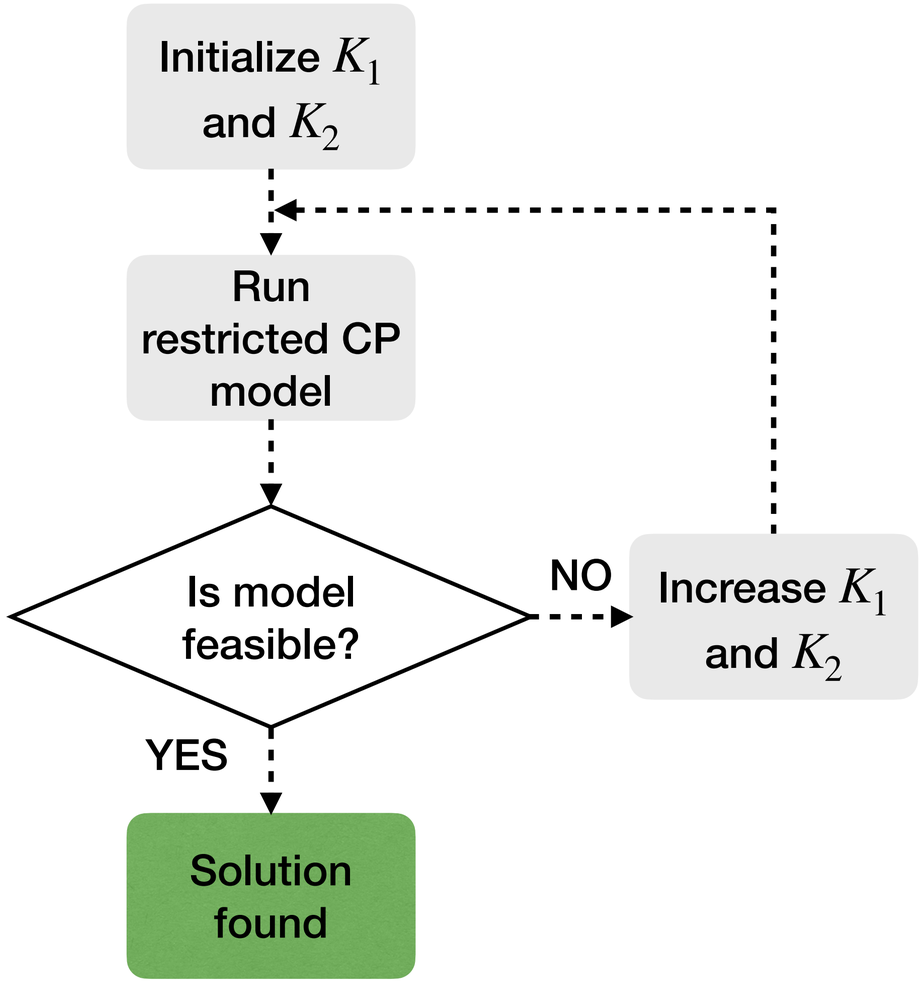
Limit the number of active (i.e., nonzero) terms in each column r (i.e., multiplication term) of **U** and **V**.

$$\sum_{i \in \mathcal{U}} |u_{i,r}| + \sum_{j \in \mathcal{V}} |v_{j,r}| \leq K_1, \qquad \forall r \in \mathcal{R}.$$

Limit the number of multiplication terms used in each output (or number of nonzeros in each row of **W** to at most K_2.

$$\sum_{r \in \mathcal{R}} |w_{k,r}| \leq K_2, \qquad \forall k \in \mathcal{W}.$$

Sparsity-based decomposition: implementation



Solved (2, 2, 4) with $R = 14$ in 19.6 hours; $K_1 = 11$, $K_2 = 7$.
Solved (3, 3, 3) with $R = 23$ in ~10 hours; $K_1 = 9$, $K_2 = 10$.

Limit the number of active (i.e., nonzero) terms in each column r (i.e., multiplication term) of **U** and **V**.

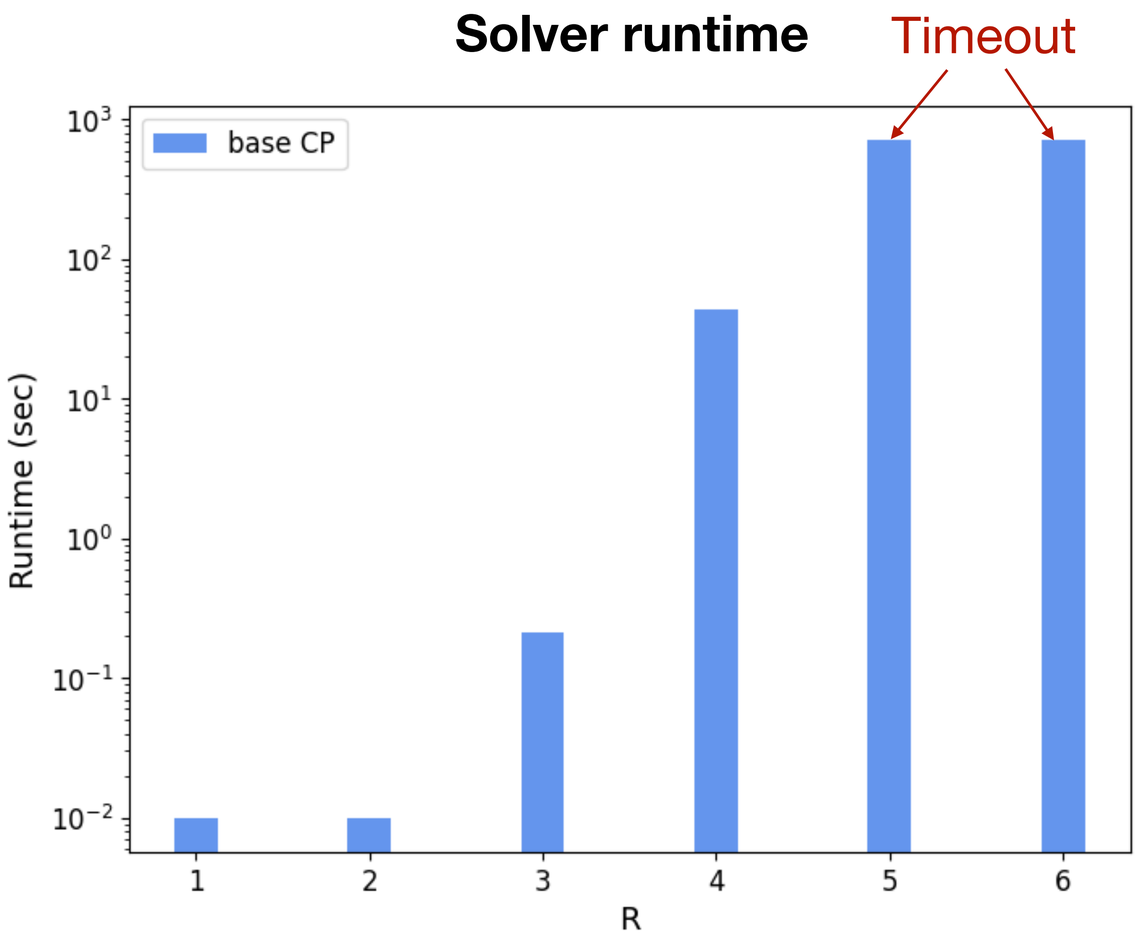
$$\sum_{i \in \mathcal{U}} |u_{i,r}| + \sum_{j \in \mathcal{V}} |v_{j,r}| \leq K_1, \qquad \forall r \in \mathcal{R}.$$

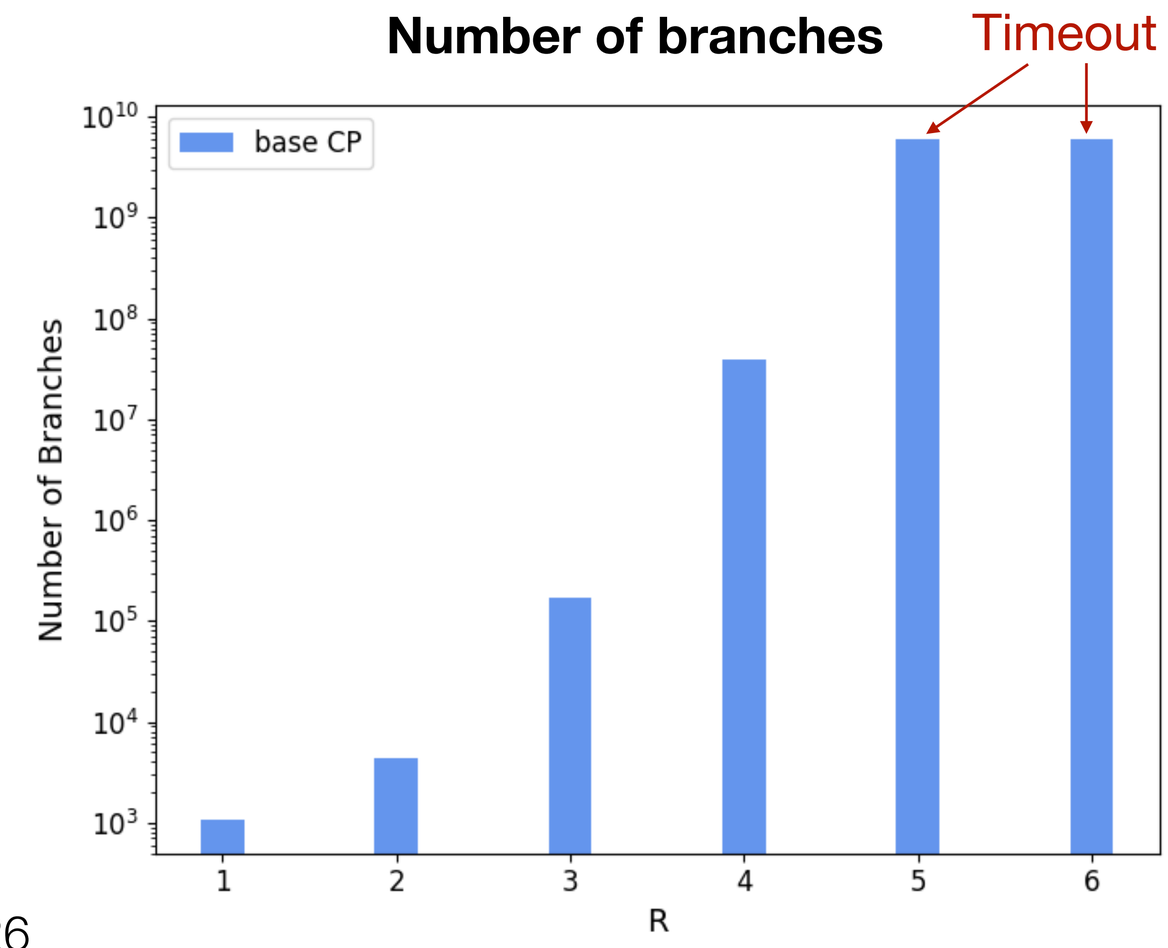
Limit the number of multiplication terms used in each output (or number of nonzeros in each row of **W** to at most K_2.

$$\sum_{r \in \mathcal{R}} |w_{k,r}| \leq K_2, \qquad \forall k \in \mathcal{W}.$$

Proving infeasibility for (2,2,2) with increasing R

- Since CP performs an exhaustive search, it can provide a proof of infeasibility if a given rank R is not achievable for certain matrix dimensions.





Symmetry-breaking constraints

Permutation Symmetry

$m_1 = (a_1 + a_4) \cdot (b_1 + b_4)$
$m_2 = (a_3 + a_4) \cdot (b_1)$
$m_3 = (a_1) \cdot (b_2 - b_4)$
$m_4 = (a_4) \cdot (b_3 - b_1)$
$m_5 = (a_1 + a_2) \cdot (b_4)$
$m_6 = (a_3 - a_1) \cdot (b_1 + b_2)$
$m_7 = (a_2 - a_4) \cdot (b_3 + b_4)$

$c_1 = m_1 + m_4 - m_5 + m_7$
$c_2 = m_3 + m_5$
$c_3 = m_2 + m_4$
$c_4 = m_1 - m_2 + m_3 + m_6$

$m_1 = (a_2 - a_4) \cdot (b_3 + b_4)$
$m_2 = (a_3 + a_4) \cdot (b_1)$
$m_3 = (a_1) \cdot (b_2 - b_4)$
$m_4 = (a_4) \cdot (b_3 - b_1)$
$m_5 = (a_1 + a_2) \cdot (b_4)$
$m_6 = (a_3 - a_1) \cdot (b_1 + b_2)$
$m_7 = (a_1 + a_4) \cdot (b_1 + b_4)$

$c_1 = m_7 + m_4 - m_5 + m_1$
$c_2 = m_3 + m_5$
$c_3 = m_2 + m_4$
$c_4 = m_7 - m_2 + m_3 + m_6$

$$\texttt{lexicographic-strict}([u_{:,r}; v_{:,r}], [u_{:,r+1}; v_{:,r+1}]), \quad \forall r \in \mathcal{R}$$

Sign Symmetry

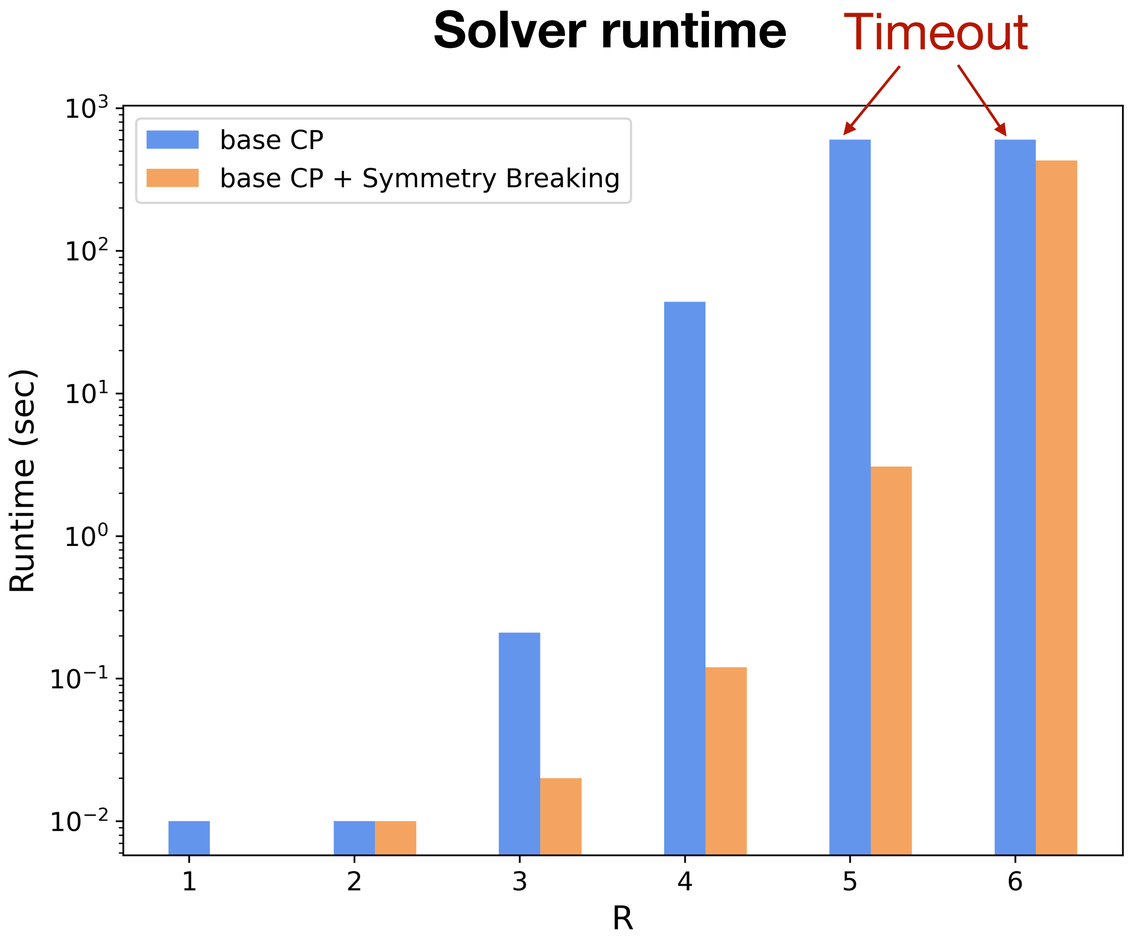
Multiplying two sets of terms by -1 results in same solution.

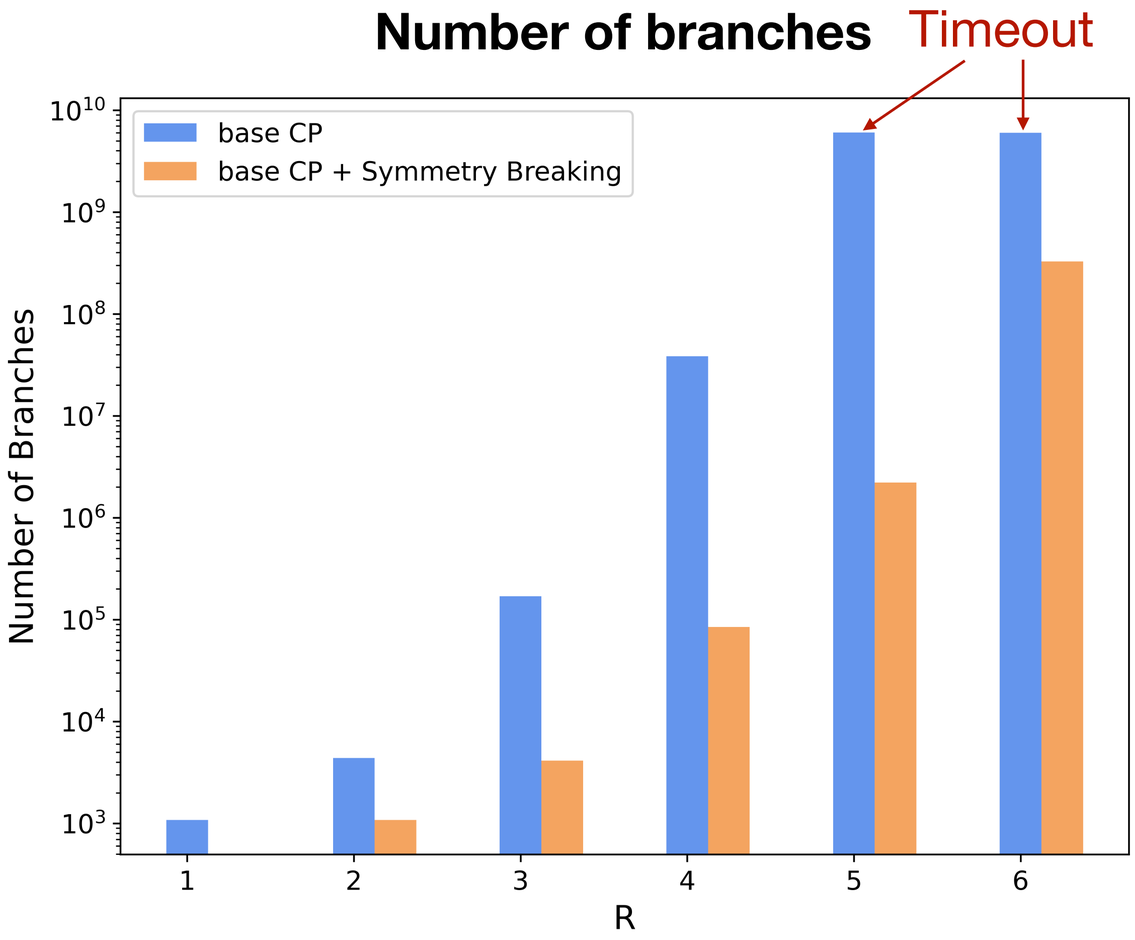
$$m_3 = (a_1) \cdot (b_2 - b_4) \quad \equiv \quad m_3 = (-a_1) \cdot (-b_2 + b_4)$$

$$u_{1,r} \leq 0$$
$$u_{i,r} \leq \sum_{i'=1}^{i-1} |u_{i',r}| \qquad \forall r \in \mathcal{R}, i > 1, i \in \mathcal{U}$$

$$w_{1,r} \leq 0$$
$$w_{k,r} \leq \sum_{k'=1}^{k-1} |w_{k',r}| \qquad \forall r \in \mathcal{R}, k > 1, k \in \mathcal{W}.$$

Proving infeasibility for (2,2,2) with increasing R





Note: we have not been able to tackle the (3,3,3) case yet with $19 \leq R \leq 22$

Conclusions & future work

Novel CP approach to solve the Fast Matrix Multiplication problem with a set of symmetry breaking constraints and a decomposition approach with inexact inequalities.

Feasible Cases

Results:

- CP finds solutions quickly for small instances
- Struggles to scale for larger matrix dimensions or ranks; decomposition helps a lot

Future work:

- Explore branching strategies and other important search-related decisions
- Further investigate on product constraint formulation in CPO

Infeasible Cases

Results:

- Can prove infeasibility for 2×2 matrices for $R \leq 6$
- Have not tackled 3×3 matrices yet

Future work:

- Analyze the highly structured nature of this problem to develop more valid inequalities that can further reduce the search space of our CP model

Thank you! Questions?

CP Optimizer code + minizinc model/data

https://github.com/khalil-research/Matrix-Mult-CP

There remains open problems in this area, such as closing the gap for (3,3,3).

We believe CP might be well positioned to close this.