Revista Eletrônica de Iniciação Científica em Computação, 2025, 23:1
ISSN: 3085-8461 • doi: 10.5753/reic.2025.6146

Artigo de Pesquisa/Research Paper

Compressão de Matrizes com Suporte a Operações Algébricas Utilizando Pouca Memória

Matrix Compression with Support for Algebraic Operations in Small Space

Filipe C. O. Resende [Universidade Estadual de Campinas | *f266879@dac.unicamp.br*]
Felipe A. Louza [Universidade Federal de Uberlândia | *louza@ufu.br*]

Instituto de Computação, Universidade Estadual de Campinas, Av. Albert Einstein, 1251, Cidade Universitária, Campinas, SP, 13083-889, Brasil.

Resumo. Este trabalho investiga estratégias para reduzir o consumo de memória durante a compressão de matrizes com suporte a operações algébricas diretamente sobre a representação comprimida. Em particular, mostramos como modificar o método mm-RePair [Ferragina *et al*., 2022] realizando a compressão de grandes matrizes de forma particionada, dividindo-as em blocos de linhas consecutivas, que são processados sequencialmente utilizando menos memória. O método proposto mantém o suporte a operações de multiplicação sobre os dados comprimidos e, embora provoque uma pequena piora na taxa de compressão, reduz significativamente o consumo de memória conforme o número de blocos aumenta, possibilitando a sua aplicação em situações em que a matriz é maior do que a memória disponível.

Abstract. This work investigates strategies to reduce memory consumption in matrix compression supporting algebraic operations directly on the compressed representation. In particular, we show how to modify the method mm-RePair [Ferragina *et al*., 2022] by performing the compression of large matrices in a partitioned manner, dividing them into blocks of consecutive rows, which are processed sequentially using less memory. The proposed method preserves support for multiplication operations on the compressed data and, although it causes a slight degradation in compression ratio, significantly reduces memory consumption as the number of blocks increases, enabling its application in scenarios where the matrix is larger than the available memory.

Palavras-chave: Algoritmos de Compressão, Compressão de Matrizes, Compressed Linear Algebra, Multiplicação de Matrizes, Compressão Gramatical

Keywords: Compression Algorithms, Matrix Compression, Compressed Linear Algebra, Matrix Multiplication, Grammar Compression

Recebido/Received: 20 June 2025 • **Aceito/Accepted:** 23 June 2025 • **Publicado/Published:** 09 July 2025

1 Introdução

No contexto atual da área de Inteligência Artificial, é cada vez mais comum lidar com matrizes de grandes dimensões. Vários algoritmos de aprendizado de máquina utilizam operações de multiplicação entre matrizes e vetores para treinar modelos. Essas operações requerem que a matriz inteira seja percorrida, com duas operações de ponto flutuante sendo realizadas por elemento. Nessa situação, é essencial que a matriz caiba na memória RAM disponível para garantir um bom desempenho, o que torna a compressão dessas matrizes uma necessidade importante. Ao comprimir matrizes permitindo a multiplicação por vetores diretamente nos dados comprimidos, sem a necessidade de descompactá-los, podemos não apenas armazenar maiores matrizes na memória RAM, mas também minimizar o número de posições lidas na memória durante a multiplicação, melhorando, assim, o desempenho de espaço e tempo.

Frequentemente, métodos de compressão utilizados em Aprendizado de Máquina são do tipo com perdas. Sistemas como TensorFlow [Abadi *et al*., 2016] e PStore [Bhattacherjee *et al*., 2014], por exemplo, aplicam truncamento de mantissa para melhorar a eficiência em tarefas específicas. Em cenários onde pequenas imprecisões são aceitáveis, como em cálculos iniciais de gradientes, a compressão com perdas também pode ser útil [Elgohary *et al*., 2018]. Já no contexto de operações de álgebra linear, a compressão de matrizes sem perdas é desejável para garantir resultados precisos. Infelizmente, métodos de compressão sem perdas para dados unidimensionais geralmente não conseguem aproveitar as redundâncias das matrizes de forma eficiente. Além disso, esses métodos frequentemente exigem a descompressão total das matrizes para a execução de operações algébricas [Ferragina *et al*., 2022].

Nos trabalhos de [Elgohary *et al*., 2018] e [Elgohary *et al*., 2019], foi introduzida uma nova linha de pesquisa chamada *Compressed Linear Algebra* (CLA). Nesses estudos, são apresentados métodos de compressão sem perdas que, além de oferecerem boas taxas de compressão, também possibilitam a realização eficiente de operações algébricas, como a multiplicação de matrizes por vetores, diretamente na forma comprimida da matriz. Seguindo essa linha, [Ferragina *et al*., 2022] propuseram o método de compressão sem perdas para matrizes mm-RePair, que permite a multiplicação de uma matriz comprimida por um vetor, tanto à direita quanto à esquerda, em tempo e espaço proporcionais ao tamanho da matriz comprimida.

Neste trabalho, investigamos o particionamento de matrizes em blocos com o objetivo de aplicar, de forma sequencial, o método mm-RePair a cada bloco da matriz e, em seguida, utilizar os blocos comprimidos para realizar multiplicações.

Essa abordagem visa reduzir o pico de uso de memória tanto na etapa de compressão quanto na multiplicação. Além de reduzir o pico de uso de memória, essa solução também reduz o tempo de compressão das matrizes, embora de forma menos expressiva do que dos ganhos obtidos com o uso de *multithreading*, apresentados no artigo original [Ferragina *et al*., 2022]. Para avaliar nossa proposta, realizamos experimentos com diferentes matrizes, analisando as taxas de compressão, os tempos de compressão, os tempos de execução da multiplicação e os picos de uso de memória para diferentes números de blocos. Neste trabalho, realizamos multiplicações apenas por vetores à direita das matrizes.

2 Trabalhos relacionados

Dada uma cadeia de caracteres ω de um alfabeto Σ, podemos derivar um conjunto de regras R cujo símbolo incial S gera exclusivamente ω. O princípio central é substituir padrões repetidos na cadeia por regras reutilizáveis. Se a cadeia ω possuir muitas subcadeias repetidas, sua representação na forma de gramática livre de contexto (GLC) pode economizar espaço. Esta forma de compressão é utilizada para comprimir textos e é chamada de compressão por gramáticas [Navarro, 2016].

Por exemplo, a cadeia $\omega = abcabcabcabcabcabcabcabc$ pode ser representada pela gramática $G = (\{S, A, B\}, \{a, b, c\}, R, S)$, em que R é o conjunto de regras:

$$R = \{(S, AA), (A, BBBB), (B, abc)\}$$

O método Re-Pair (*Recursive Pairing*) [Larsson and Moffat, 1999], é um algoritmo de compressão por gramáticas que substitui recursivamente pares de símbolos adjacentes por um único símbolo em um texto, reduzindo, assim, o tamanho do texto original. A cada iteração, o par de símbolos adjacentes com maior número de ocorrências no texto, digamos ab, é substituído por um novo símbolo não-terminal A, e uma nova regra é adicionada à gramática: $A \to ab$. Quando nenhum par aparecer duas ou mais vezes, o algoritmo termina. No fim, tem-se uma sequência C de símbolos, chamada de *parser*, com as substituições apropriadas e um conjunto de regras R que, juntos, podem reconstruir o texto original.

Em [Ferragina *et al*., 2022], é apresentado o método de compressão sem perdas de matrizes mm-RePair, que dá suporte a multiplicação de uma matriz comprimida por um vetor, tanto à direita quanto à esquerda, em tempo e espaço proporcionais ao tamanho da matriz comprimida. Os autores introduzem o formato CSRV (*compressed sparse row/value*), uma adaptação do conhecido formato CSR (*compressed sparse row*) [Saad, 2003].

A representação CSRV de uma matriz M é composta por duas estruturas: um vetor V, que guarda os valores não nulos distintos de M, e uma sequência de símbolos S, que indica as posições da matriz onde os valores de V aparecem. Esta sequência delimita as linhas da matriz por meio do símbolo especial \$. Se, em uma linha, aparecer o símbolo $\langle i, j\rangle$, significa que o valor $V[i]$ aparece na coluna j desta linha. Por exemplo, considere a seguinte matriz de entrada:

$$M = \begin{bmatrix} 5.3 & 8.1 & 6.0 & 2.7 & 6.0 & 5.3 \\ 2.7 & 0 & 8.1 & 0 & 6.0 & 5.3 \\ 2.7 & 0 & 8.1 & 0 & 6.0 & 5.3 \\ 5.3 & 8.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6.0 & 5.3 \\ 5.3 & 8.1 & 6.0 & 2.7 & 0 & 0 \\ 5.3 & 8.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6.0 & 2.7 & 0 & 0 \end{bmatrix}$$

Sua representação no formato CSRV é dada por:

$$V = [5.3, 8.1, 6.0, 2.7]$$
$$\begin{aligned} S = &\langle 0,0\rangle\langle 1,1\rangle\langle 2,2\rangle\langle 3,3\rangle\langle 2,4\rangle\langle 0,5\rangle\$ \\ &\langle 3,0\rangle\langle 1,2\rangle\langle 2,4\rangle\langle 0,5\rangle\$ \\ &\langle 3,0\rangle\langle 1,2\rangle\langle 2,4\rangle\langle 0,5\rangle\$\langle 0,0\rangle\langle 1,1\rangle\$\langle 2,4\rangle\langle 0,5\rangle\$ \\ &\langle 0,0\rangle\langle 1,1\rangle\langle 2,2\rangle\langle 3,3\rangle\$\langle 0,0\rangle\langle 1,1\rangle\$\langle 2,2\rangle\langle 3,3\rangle\$ \end{aligned}$$

Em uma implementação, a sequência S pode ser armazenada como uma sequência de inteiros (por exemplo, inteiros de 32 bits sem sinal). O símbolo \$ pode ser representado pelo inteiro 0 e um par $\langle i, j\rangle$ pode ser codificado como $1 + im + j$, onde m é o número de colunas da matriz. Como $0 \leq j < m$, é possível extrair i e j através da divisão euclidiana de $im + j$ por m: i é o quociente e j, o resto.

O método mm-RePair aplica o compressor Re-Pair diretamente à sequência S da represenção CSRV (S, V) de M. O resultado é uma representação compactada (C, R, V), onde C é a sequência final de símbolos e R é o conjunto de regras que descrevem a expansão dos símbolos não-terminais em C.

Para permitir que as operações de álgebra linear possam ser realizadas sem a necessidade de descompressão da matriz, [Ferragina *et al*., 2022] modificaram o algoritmo Re-Pair para que nenhuma regra de R contenha o símbolo especial \$. Dessa forma, a sequência C resultante assume a forma $C = \omega_1\$\omega_2\$\cdots\omega_n\$$, onde $\omega_1, \omega_2, \cdots, \omega_n \in (\Sigma \cup N)^*$, com Σ representando o conjunto de símbolos terminais da sequência original S (cada par $\langle i, j\rangle$ está sendo tratado como um único símbolo) e N o conjunto de símbolos não-terminais gerados pelo Re-Pair. Por exemplo, considere a sequência S da representação CSRV da matriz M. Após a aplicação do Re-Pair sobre S, obtemos:

$$C = N_4N_2\$N_6\$N_6\$N_1\$N_2\$N_4\$N_1\$N_3\$$$
$$\begin{aligned} R = \{&N_1 \to \langle 0,0\rangle\langle 1,1\rangle, \\ &N_2 \to \langle 2,4\rangle\langle 0,5\rangle, \\ &N_3 \to \langle 2,2\rangle\langle 3,3\rangle, \\ &N_4 \to N_1N_3, \\ &N_5 \to \langle 3,0\rangle\langle 1,2\rangle, \\ &N_6 \to N_5N_2\} \end{aligned}$$

Considerando uma matriz $M \in \mathbb{R}^{n\times m}$ na representação (S, V) e um vetor $x \in \mathbb{R}^m$, temos as seguintes definições [Ferragina *et al*., 2022]:

Definição 2.1. Dado um par $\langle l, j\rangle \in S$,

$$eval_x(\langle l, j\rangle) = V[l] \cdot x[j]$$

Definição 2.2. Dado um símbolo não-terminal $N_i \in C$ cuja a expansão é $\langle l_1, j_1\rangle, \langle l_2, j_2\rangle, \cdots \langle l_k, j_k\rangle$,

$$eval_x(N_i) = \sum_{i=1}^{k} eval_x(\langle l_i, j_i\rangle) = \sum_{i=1}^{k} V[l_i] \cdot x[j_i]$$

É fácil observar que, se a regra de produção associada a N_i é $N_i \rightarrow AB$, com $A, B \in \Sigma \cup N$, então $eval_x(N_i) = eval_x(A) + eval_x(B)$.

Para fins didáticos e simplificação nas explicações subsequentes, vamos assumir que C segue a forma:

$$C = N_{i_1}\$N_{i_2}\$\cdots N_{i_n}\$, \text{onde } N_{i_1}, N_{i_2}, \cdots, N_{i_n} \in N.$$

Embora sempre seja possível converter C para esse formato com a inclusão de regras adicionais ao final da execução do Re-Pair, isso não traz benefícios em termos de eficiência ou compressão. Portanto, essa forma será utilizada apenas para facilitar a compreensão da operação de multiplicação do método.

Note que, na nomenclatura que estamos adotando, N_1 representa o primeiro símbolo não-terminal gerado durante a execução do Re-Pair, enquanto N_{i_1} denota o símbolo não-terminal correspondente à primeira linha da matriz.

Lema 2.1 (Ferragina *et al*. [2022], Lema 3.3)**.** *Considerando uma matriz $M \in \mathbb{R}^{n\times m}$ na representação (C, R, V), com $C = N_{i_1}\$N_{i_2}\$\cdots N_{i_n}\$$ e um vetor $x \in \mathbb{R}^m$, se $y = M \cdot x$, então $y[r] = eval_x(N_{i_r})$, para $r = 1, 2, \cdots, n$.*

Para calcular $y = M \cdot x$ sem descomprimir a matriz, precisamos primeiramente preencher um vetor auxiliar $W[1, q]$, onde q é o número de regras de R. O vetor W deve ser preenchido de forma sequencial, do índice 1 ao índice q, de modo que $W[i] = eval_x(N_i)$. Se $N_i \rightarrow AB$, então $W[i]$ recebe $\text{eval}_x(A) + \text{eval}_x(B)$. Se A ou B for um símbolo terminal $\langle l, j\rangle$, então basta fazer $eval_x(\langle l, j\rangle) = V[l] \cdot x[j]$. Caso um deles seja um símbolo não-terminal N_k, então $eval_x(N_k)$ já estará presente em $W[k]$, pois $k < i$ (uma vez que N_k foi criado antes de N_i). Como cada posição de W é preenchida em $\mathcal{O}(1)$, temos que o preenchimento completo de W ocorre em $\mathcal{O}(q) = \mathcal{O}(|R|)$.

A etapa final consiste em iterar sobre $C = N_{i_1}\$N_{i_2}\$\cdots N_{i_n}\$$ à medida que preenchemos o vetor y. Se $N_{i_r} = N_j$, então $y[r]$ é preenchido com $W[j]$. Esta etapa ocorre em $\mathcal{O}(|C|)$.

Ao todo, a multiplicação da matriz comprimida (C, R, V) pelo vetor x é executada em tempo $\mathcal{O}(|C| + |R|)$ com uso de $O(|R|)$ palavras de memória auxiliar (devido ao vetor W).

3 Redução do uso de memória

Neste trabalho, propomos uma modificação para o método mm-RePair em que a matriz de entrada é particionada em blocos de linhas consecutivas, que são comprimidos e multiplicados de forma sequencial. Nosso principal objetivo com isso é reduzir o pico de memória utilizada.

Nossa implementação funciona da seguinte forma: dividimos a matriz $M_{n\times m}$ em b blocos, cada um com n/b linhas e m colunas. Caso n não seja divisível por b, particionamos a matriz de forma que alguns blocos tenham $\lfloor n/b\rfloor$ linhas e outros $\lceil n/b\rceil$ linhas. Particionamos a matriz antes de convertê-la para o formato CSRV. Com isso, temos b grupos (C_i, R_i, V_i) ao fim da etapa de compressão com o Re-Pair. Observe que, como os valores não nulos distintos de cada bloco podem variar, os vetores V_i resultantes podem ser diferentes entre si. Isso contrasta com a abordagem de [Ferragina *et al*., 2022], na qual temos um único vetor V, que contém todos os valores não nulos distintos da matriz.

Por fim, para a operação de multiplicação, cada submatriz M_i na forma (C_i, R_i, V_i) é multiplicada por um vetor à direita $x \in \mathbb{R}^m$, obtendo uma sequência de subvetores $y_1, y_2, \cdots, y_b$, cada um com número de elementos igual ao número de linhas do respectivo bloco. Estes subvetores, quando considerados em conjunto, formam o vetor $y = A \cdot x$.

Por exemplo, utilizando 3 blocos, com a matriz M apresentada anteriormente, teremos dois blocos com três linhas e um bloco com duas linhas, com suas respectivas representações CSRV.

- $M_1 = \begin{bmatrix} 5.3 & 8.1 & 6.0 & 2.7 & 6.0 & 5.3 \\ 2.7 & 0 & 8.1 & 0 & 6.0 & 5.3 \\ 2.7 & 0 & 8.1 & 0 & 6.0 & 5.3 \end{bmatrix}$
  $V_1 = [5.3, 8.1, 6.0, 2.7]$
  $S_1 = \langle 0,0\rangle\langle 1,1\rangle\langle 2,2\rangle\langle 3,3\rangle\langle 2,4\rangle\langle 0,5\rangle\$$
  $\langle 3,0\rangle\langle 1,2\rangle\langle 2,4\rangle\langle 0,5\rangle\$$
  $\langle 3,0\rangle\langle 1,2\rangle\langle 2,4\rangle\langle 0,5\rangle\$$

- $M_2 = \begin{bmatrix} 5.3 & 8.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6.0 & 5.3 \\ 5.3 & 8.1 & 6.0 & 2.7 & 0 & 0 \end{bmatrix}$
  $V_2 = [5.3, 8.1, 6.0, 2.7]$
  $S_2 = \langle 0,0\rangle\langle 1,1\rangle\$\langle 2,4\rangle\langle 0,5\rangle\$$
  $\langle 0,0\rangle\langle 1,1\rangle\langle 2,2\rangle\langle 3,3\rangle\$$

- $M_3 = \begin{bmatrix} 5.3 & 8.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6.0 & 2.7 & 0 & 0 \end{bmatrix}$
  $V_3 = [5.3, 8.1, 6.0, 2.7]$
  $S_3 = \langle 0,0\rangle\langle 1,1\rangle\$\langle 2,2\rangle\langle 3,3\rangle\$$

Em seguida, aplicamos o Re-Pair a cada sequência S_i:

- $C_1 = \langle 0,0\rangle\langle 1,1\rangle\langle 2,2\rangle\langle 3,3\rangle N_1\$N_3\$N_3\$$
  $R_1 = \{N_1 \rightarrow \langle 2,4\rangle\langle 0,5\rangle,$
  $N_2 \rightarrow \langle 3,0\rangle\langle 1,2\rangle,$
  $N_3 \rightarrow N_2N_1\}$

- $C_2 = N_1\$\langle 2,4\rangle\langle 0,5\rangle\$N_1\langle 2,2\rangle\langle 3,3\rangle\$$
  $R_2 = \{N_1 \rightarrow \langle 0,0\rangle\langle 1,1\rangle\}$

- $C_3 = \langle 0,0\rangle\langle 1,1\rangle\$\langle 2,2\rangle\langle 3,3\rangle\$$
  $R_3 = \varnothing$

Dessa forma, os três blocos (C_1, R_1, V_1),(C_2, R_2, V_2) e (C_3, R_3, V_3) constituem a representação comprimida da matriz M. Os vetores de valores não nulos distintos V_1, V_2 e V_3 são iguais neste caso, mas poderiam ser diferentes se os valores não nulos distintos que aparecem em cada bloco (ou sua ordem de aparição) fossem diferentes.

Do ponto de vista do uso de memória RAM, a compressão em blocos apresenta uma vantagem: enquanto na compressão global a sequência S, o vetor V e o conjunto

Tabela 1. Estrutura das matrizes utilizadas nos experimentos

Dataset	Tamanho (MB)	Linhas	Colunas
Covtype	239,37	581.012	54
Census	1.275,36	2.458.285	68
Optical	432,55	325.834	174
Airline78	3.199,96	14.462.943	29

Tabela 2. Densidade de não nulos nas matrizes

Dataset	% Não Nulos	Não Nulos Distintos
Covtype	22,00%	6.682
Census	43,03%	45
Optical	97,50%	897.176
Airline78	72,66%	7.794

de regras R são armazenados integralmente na memória, na compressão em blocos os dados são processados de forma sequencial. No exemplo, armazenam-se S_1, R_1 e V_1, seguidos por S_2, R_2 e V_2, e assim sucessivamente. Essa abordagem reduz o pico de uso de memória, pois cada S_i é menor que S, e cada R_i e V_i têm tamanhos iguais ou menores que R e V, respectivamente. Em nossa implementação do Re-Pair, a sequência C é gerada diretamente a partir de S, reutilizando o espaço de memória já alocado para S. Por esta razão, C não aumenta o pico de uso de memória.

Além disso, o método Re-Pair executa mais rápido (já que realiza menos substituições na criação de sua gramática) na abordagem em blocos, o que resulta em um tempo de compressão menor quando comparado com a abordagem original do mm-RePair.

Observamos que, em [Ferragina *et al.*, 2022], os autores apresentam uma versão *multithread* do mm-RePair, na qual a matriz de entrada M também é particionada. Primeiro, M é convertida para a representação CSRV (S, V). Depois, S é dividida em b blocos $S_1, S_2, \cdots, S_b$, de modo que cada bloco contenha a mesma quantidade de linhas, exceto, possivelmente, o último bloco S_b. Em seguida, cada bloco S_i é comprimido em paralelo com o Re-Pair, gerando uma sequência de símbolos C_i e um conjunto de regras R_i. Ao final, obtêm-se b grupos (C_i, R_i) e um único vetor V de valores não nulos distintos, compartilhado por todos os blocos. Os autores avaliaram os tempos de execução e os picos de memória nesta abordagem. Apesar de uma redução significativa dos tempos de compressão e multiplicação, houve um aumento do uso de memória.

4 Experimentos

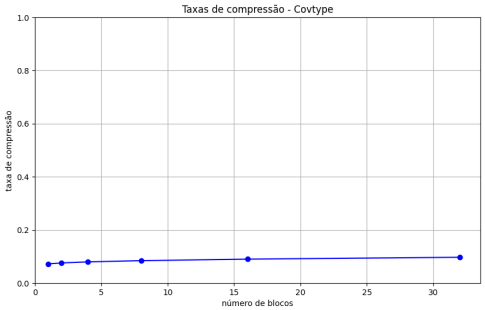
Executamos todos os experimentos em uma máquina equipada com um processador `AMD Ryzen 9 7900` de 12 núcleos, com frequência máxima de 5,48 GHz. A máquina possui 24 processadores (12 núcleos com 2 threads por núcleo) e está configurada com 128 GB de RAM.

Todos os algoritmos foram implementados em `C++` e estão disponíveis para download[1]. Para medir os tempos de execução utilizamos a biblioteca `<time.h>`.

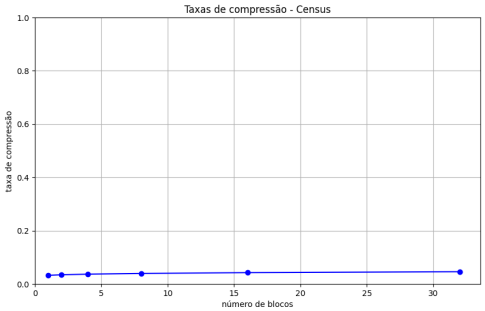
Realizamos experimentos com matrizes disponíveis em `https://kaggle.com/datasets/giovannimanzini/some-machine-learning-matrices`. As Tabelas 1 e 2 fornecem informações detalhadas sobre essas matrizes. Os tamanhos correspondem à representação da matriz em memória utilizando 8 bytes por valor.

Avaliamos a divisão da matriz em diferentes números de blocos (1, 2, 4, 8, 16 e 32). Para cada matriz e número de blocos, realizamos a compressão e executamos a multiplicação da matriz por um vetor à direita. Para cada matriz $M \in \mathbb{R}^{n \times m}$, o vetor $x \in \mathbb{R}^m$ escolhido para a multiplicação foi $x = (1, 1, \cdots, 1)^T$. As taxas de compressão são dadas por $\frac{\text{Tamanho Comprimido}}{\text{Tamanho Original}}$.

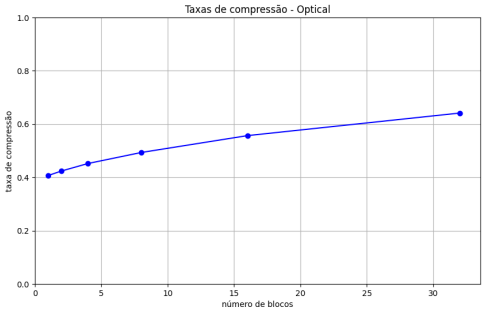
Na Figura 1, podemos observar que a abordagem de



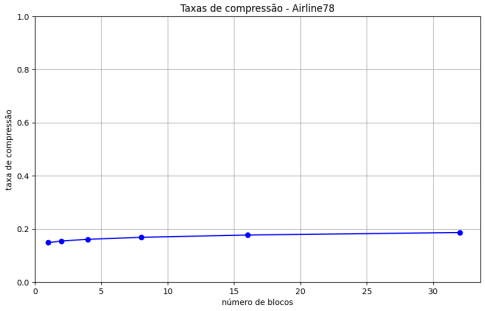
(a) Covtype



(b) Census



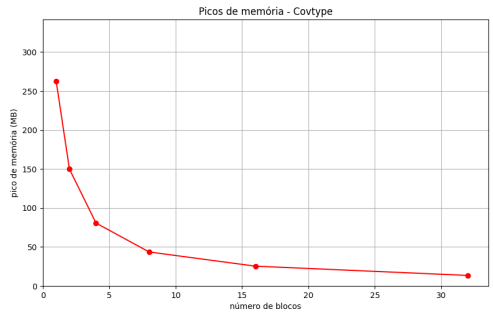
(c) Optical



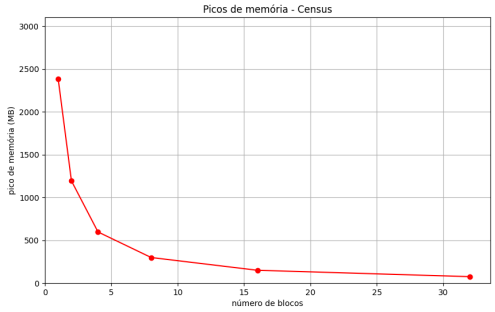
(d) Airline78

Figura 1. Taxa de compressão × Número de blocos

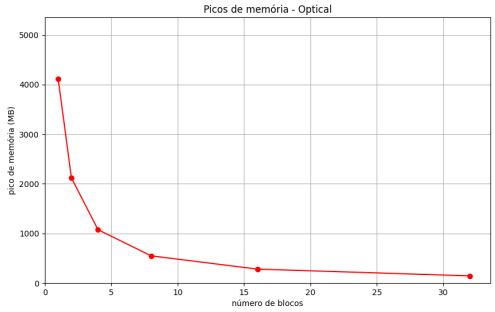
[1] `https://github.com/filipecoresende/compressed-matrix-multiplication`



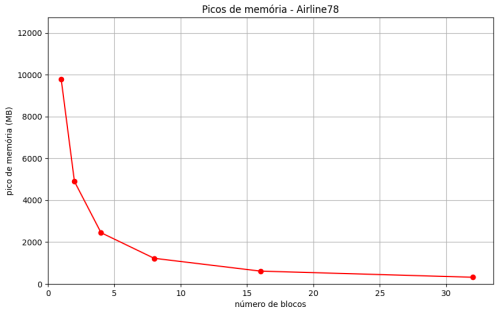
(a) Covtype



(b) Census

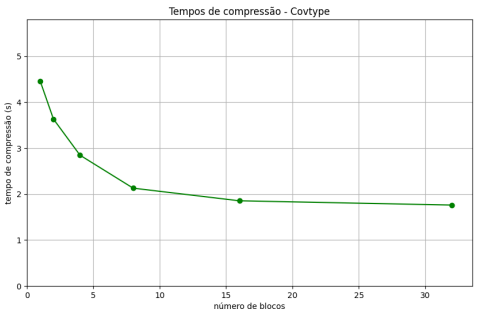


(c) Optical

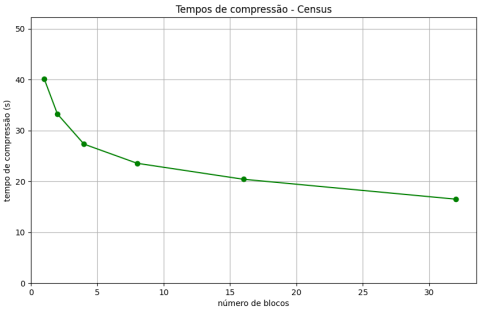


(d) Airline78

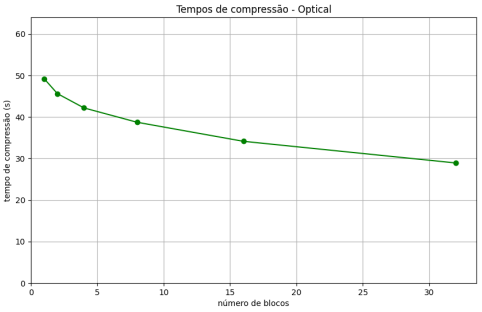
Figura 2. Pico de memória da compressão × Número de blocos



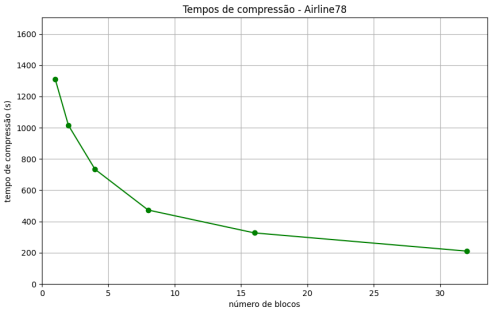
(a) Covtype



(b) Census



(c) Optical



(d) Airline78

Figura 3. Tempo de compressão × Número de blocos

compressão em blocos aumentou em todos os testes o tamanho dos dados comprimidos, isto é, piorou levemente a taxa de compressão em relação à compressão global. Esse resultado era esperado, uma vez que a divisão em blocos faz com que o mm-RePair aproveite menos as redundâncias da matriz de entrada.

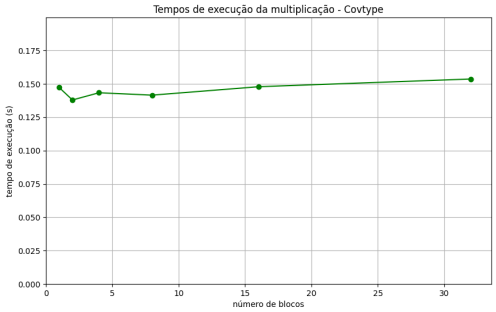
Por outro lado, na Figura 2, podemos ver que a compressão em blocos reduziu significativamente, em todos os testes, o consumo de memória em comparação com a compressão global.

A Tabela 3 apresenta a quantidade de memória economizada na versão com 32 blocos, quando comparada com a versão original. Além disso, podemos ver na Figura 3 que a compressão em blocos também resultou em uma redução no tempo de compressão. Na Tabela 3 temos também os *speedups* obtidos, comparando o tempo de compressão com 1 bloco pelo tempo de compressão com 32 blocos.

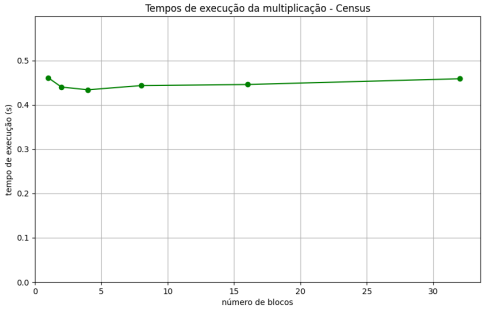
Na Figura 4, podemos ver que o tempo de execução da multiplicação apresentou pouca variação com o aumento do número de blocos. Em contrapartida, observa-se na Figura 5

Tabela 3. Speedup e redução de pico de memória na compressão: comparação entre 1 bloco e 32 blocos

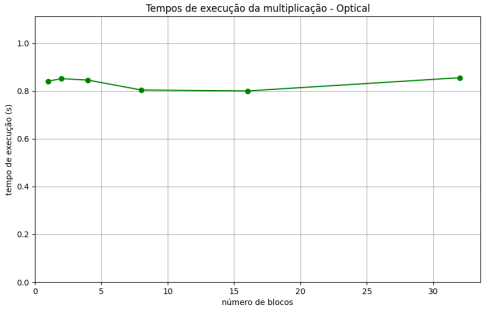
Dataset	Speedup	Redução do pico de memória
Covtype	2,53	94,87%
Census	2,44	96,87%
Optical	1,70	96,49%
Airline78	6,25	96,74%



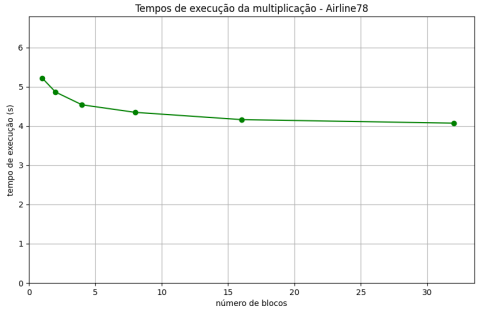
(a) Covtype



(b) Census



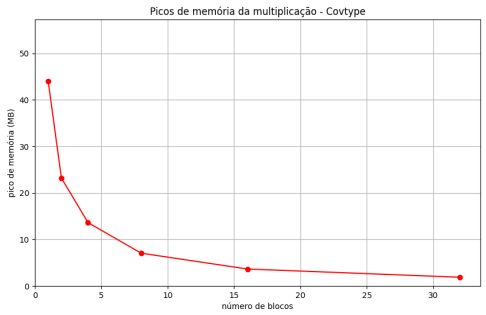
(c) Optical



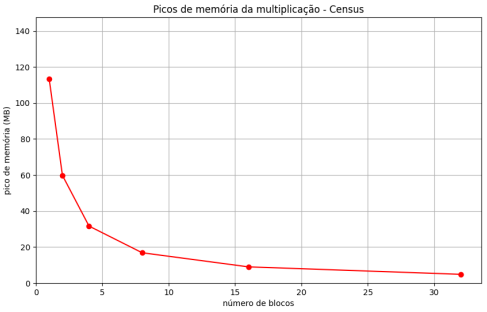
(d) Airline78

Figura 4. Tempo de execução da multiplicação × Número de blocos

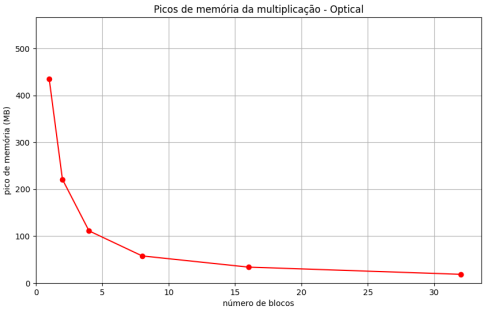
uma redução significativa no pico de uso de memória durante a multiplicação. Portanto, a divisão da matriz em blocos se



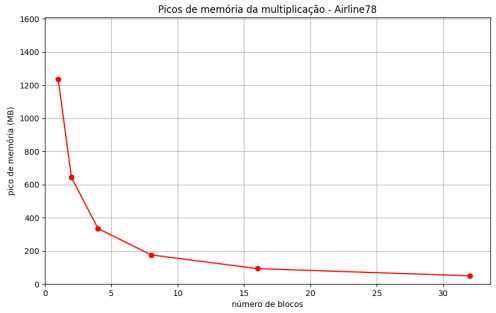
(a) Covtype



(b) Census



(c) Optical



(d) Airline78

Figura 5. Pico de memória da multiplicação × Número de blocos

mostrou vantajosa para a multiplicação de forma geral, pois alterou minimamente o tempo de execução mas reduziu muito o uso de memória RAM.

5 Conclusão

Neste trabalho, mostramos como reduzir a quantidade de memória utilizada pelo método mm-RePair [Ferragina *et al.*, 2022]. A matriz de entrada foi particionada e processada de maneira sequencial. Nos experimentos, avaliamos o uso de memória de acordo com o número de blocos de divisão

da matriz, tanto durante a compressão quanto na multiplicação das matrizes comprimidas com vetores à direita. Os resultados mostram uma redução significativa no consumo de memória. Além disso, observou-se uma redução no tempo da compressão à medida que o número de blocos aumenta. Já na multiplicação, de modo geral, o tempo de execução não variou significativamente com o aumento do número de blocos. O único ponto que ficou relativamente comprometido foi a taxa de compressão, que apresentou uma degradação à medida que o número de blocos aumentava.

Declarações complementares

Financiamento

Os autores agradecem o apoio financeiro das agências de fomento CNPq (311128/2025-4 e 408314/2023-0) e FAPEMIG (APQ-01217-22).

Contribuições dos autores

Louza contribuiu para a concepção e desenvolvimento da metodologia deste estudo. Resende realizou os experimentos e a análise dos dados. Ambos os autores colaboraram na redação e revisão crítica do manuscrito. Ambos os autores leram e aprovaram a versão final do manuscrito.

Conflitos de interesse

Os autores declaram que não têm nenhum conflito de interesses.

Disponibilidade de dados e materiais

Os códigos dos algoritmos implementados estão disponíveis para download em `https://github.com/filipecoresende/compressed-matrix-multiplication`. As matrizes utilizadas nos experimentos estão disponíveis em `https://kaggle.com/datasets/giovannimanzini/some-machine-learning-matrices`.

Referências

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. DOI: 10.48550/arXiv.1603.04467.

Bhattacherjee, S., Deshpande, A., and Sussman, A. (2014). Pstore: an efficient storage framework for managing scientific data. In Jensen, C. S., Lu, H., Pedersen, T. B., Thomsen, C., and Torp, K., editors, *Conference on Scientific and Statistical Database Management, SSDBM '14, Aalborg, Denmark, June 30 - July 02, 2014*, pages 25:1–25:12. ACM. DOI: 10.1145/2618243.2618268.

Elgohary, A., Boehm, M., Haas, P. J., Reiss, F. R., and Reinwald, B. (2018). Compressed linear algebra for large-scale machine learning. *VLDB J.*, 27(5):719–744. DOI: 10.1007/S00778-017-0478-1.

Elgohary, A., Boehm, M., Haas, P. J., Reiss, F. R., and Reinwald, B. (2019). Compressed linear algebra for declarative large-scale machine learning. *Commun. ACM*, 62(5):83–91. DOI: 10.1145/3318221.

Ferragina, P., Manzini, G., Gagie, T., Köppl, D., Navarro, G., Striani, M., and Tosoni, F. (2022). Improving matrix-vector multiplication via lossless grammar-compressed matrices. *Proc. VLDB Endow.*, 15(10):2175–2187. DOI: 10.14778/3547305.3547321.

Larsson, N. J. and Moffat, A. (1999). Offline dictionary-based compression. In *Data Compression Conference, DCC 1999, Snowbird, Utah, USA, March 29-31, 1999*, pages 296–305. IEEE Computer Society. DOI: 10.1109/DCC.1999.755679.

Navarro, G. (2016). *Compact Data Structures - A Practical Approach*. Cambridge University Press.

Saad, Y. (2003). *Iterative methods for sparse linear systems*. SIAM.