arXiv:2302.04515v1 [cs.SC] 9 Feb 2023

使用拟可分矩阵进行精确计算

Clément Pernet Grenoble INP、Univ. Grenoble Alpes CNRS、LJK, UMR 5224，法国格勒诺布尔

Hippolyte Signargout ENS de Lyon、U. Lyon、CNRS、Inria、UCBL、LIP UMR 5668，法国里昂；LJK UMR 5224，法国格勒诺布尔

Gilles Villard CNRS、U. Lyon、Inria、ENS de Lyon、UCBL、LIP UMR 5668，法国里昂

摘要

拟可分矩阵是一类秩结构矩阵，广泛应用于数值线性代数，并且在计算机代数领域日益受到关注，例如应用于多项式矩阵线性化。这类矩阵存在多种表示格式，但人们很少对它们进行比较。

我们展示了如何将最核心的 SSS 和 HSS 格式应用于符号计算，在这种计算中，精确秩取代了基于阈值的数值秩。我们阐明它们之间的联系，并将其与 Bruhat 格式进行比较。为此，我们给出基于快速矩阵乘法的空间和时间成本估算，并比较它们的首项常数。软件实验为这一比较提供了支持。

我们进一步推进了 Bruhat 格式的研究，给出了一个生成算法，该算法遵循 Crout消元方案，并可进一步专门化为快速算法，用于从稀疏矩阵或 Bruhat 表示之和构造该格式。

关键词

准可分矩阵，SSS，HSS，Bruhat 生成器

1 引言

拟可分矩阵常见于数值分析中的各种问题，并且在计算机代数中的重要性日益提升，例如，它们可用于处理多项式矩阵的线性化 [2]。针对这类矩阵及其推广形式的结构化表示已得到广泛研究，但据我们所知，人们尚未对这些表示进行详细比较。本文旨在将数值分析中两种最重要的格式 SSS [9] 和 HSS [5, 16],应用于精确计算，并在理论和实验上将它们与 Bruhat 格式 [21]进行比较。这些格式的存储大小均随维数和结构参数线性增长。我们不研究 Givens权表示 [7]，因为它高度依赖正交变换 $\mathbb{C}$，而在代数框架中转化这类变换更具挑战性。关于拟可分矩阵计算的详尽参考文献，请见 [13, 25, 26]。

定义 1.1。 如果对于所有 $k \in [\![1, n]\!]$，都有 秩$(A_{1..k,k+1..n}) \leq s$ 且 秩$(A_{k+1..n,1..k}) \leq s$，则一个 $n \times n$矩阵 A称为 s-拟可分矩阵。

复杂度界记法。 我们考虑抽象交换域 K 上的矩阵，并统计 K 中的算术运算次数。我们对各种表示形式进行详细比较，尤其旨在确定渐近乘法常数，这是衡量算法实际表现的一项有参考价值的指标。为此，我们将复杂度中的首项作为比较标准：也就是说，存在函数 $\mathrm{T}_{XXX}(n, s)$，使得使用参数 n，s 运行算法 XXX 所需的域运算次数渐近为 $\mathrm{T}_{XXX}(n, s) + o(\mathrm{T}_{XXX}(n, s))$，其中 n和 s趋于渐近范围。我们接下来同样地，我们用记号 $\mathrm{S}_{XXX}(n, s)$表示空间开销界。我们用 ω 表示方阵乘法的一个可行指数，用 C_ω 表示相应的首项常数；也就是说，采用上述记号，$\mathrm{T}_{\mathrm{MM}}(n) = C_\omega n^\omega$，其中 MM 表示运算$C = C+AB$，且 A，B，C 属于 $\mathrm{K}^{n\times n}$。直接推广可得，对于一个$m \times k$ 乘以一个 $k \times n$ 矩阵的乘积，$\mathrm{T}_{\mathrm{MM}}(m, k, n) = C_\omega mnk \min(m, k, n)^{\omega-3}$。

1.1 秩揭示分解

拟可分矩阵的空间高效表示依赖于秩揭示分解：一个秩为 r 的矩阵 $A \in \mathrm{K}^{m\times n}$ 可表示为两个矩阵 $L \in \mathrm{K}^{m\times r} R \in \mathrm{K}^{r\times n}$，使得 $A = LR$。在精确线性代数中，此类分解通常使用高斯消元计算，例如 PLUQ、CUP、PLE、CRE 分解 [8, 15, 24],，我们统称为秩揭示分解（RF）。

上述分解算法的开销估计，要么写作 $O(mnr^{\omega-2})$，要么在一般性假设 $m = n = r$ 和一般秩剖面 [8, 15]下，给出显式的首项常数 $\mathrm{T}_{\mathrm{RF}}(m, n, r) = K_\omega n^\omega$。关于分治高斯消元算法两种主要变体在非一般情形下首项常数的开销分析，请参见 [20]。因此，我们可以假设对于 $\omega \geq 1 + \log_2 3$，有 $\mathrm{T}_{\mathrm{RF}}(m, n, r) = C_{\mathrm{RF}} mnr^{\omega-2}$，其中 C_{RF}为常数；所有实用的矩阵乘法算法都满足这一条件。注意，对于 $\omega = 3$，这两种开销都等于$2mnr$。遗憾的是，待处理块之间不可预测的秩分布会导致某些中间开销被高估，从而无法得到更紧的常数（即在一般情形下内插已知的常数 $K_3 = 2/3$）。本文提出的算法在 ω取较小值时仍然适用，但为清晰起见，我们略去推导其首项常数估计值的复杂过程。

我们的 SSS 和 HSS 算法可以使用任意秩揭示分解。另一方面，Bruhat 格式要求分解还能揭示秩剖面矩阵的额外信息，例如本文使用的 CRE 分解（见 [8]）。

定理 1.2（[8, 17]）。任意秩 r 矩阵 $A \in \mathrm{K}^{m\times n}$ 都有一个 *CRE* 分解 $A = CRE$，其中 $C \in \mathrm{K}^{m\times r}$和 $E \in \mathrm{K}^{r\times n}$ 分别为列阶梯形和行阶梯形，且 $R \in \mathrm{K}^{r\times r}$ 为置换矩阵。

因此，我们给出的 Bruhat 生成器相关成本依赖于因式分解中的常数$C_{\text{秩揭示分解}}$，这些因式分解能够生成 CRE 分解，例如 [20]中的分解。

1.2 贡献

在第 2 节中，我们定义了 SSS、HSS 和 Bruhat 格式。随后，我们将文献中针对 HSS 和 SSS 生成元的算法改编到精确计算的背景下。我们给出了 HSS 生成算法的一种全新迭代版本，并改进了 SSS 乘积算法的计算成本。对于 SSS，我们重点介绍一些基础构件，其他

表1：运算和存储成本摘要

	ω			$\omega = 3$		
	SSS	HSS	Bruhat	SSS	HSS	Bruhat
存储	$7ns$	$18ns$	$4ns$	$7ns$	$18ns$	$4ns$
从稠密矩阵生成	$2C_{\text{秩揭示分解}}n^2s^{\omega-2}$	$2^\omega C_{\text{秩揭示分解}}n^2s^{\omega-2}$	$C_{\text{秩揭示分解}}n^2s^{\omega-2}$	$4n^2s$	$16n^2s$	$2n^2s$
× 稠密块向量$(n \times v)$	$7C_\omega nsv^{\omega-2}$	$18C_\omega nsv^{\omega-2}$	$8C_\omega nsv^{\omega-2}$	$14nsv$	$36nsv$	$16nsv$
加法	$(10+2^\omega)C_\omega ns^{\omega-1}$		$\left(\frac{9\cdot 2^{\omega-2}-8}{2^{\omega-2}-1}C_\omega + 2C_{\text{秩揭示分解}}\right)ns^{\omega-1}\log n/s$	$36ns^2$		$24ns^2$ 对数 n/s
乘积	$(31+2^\omega)C_\omega ns^{\omega-1}$			$78ns^2$		

操作可以基于这些构件构建。这为改编求逆和方程组求解的快速算法 [3, 4, 10]以及合并、拆分和模型降阶等格式建模操作打开了大门[4]。在第3.3节中，我们给出一种通用的 Bruhat 生成算法，并以此推导出从稀疏矩阵以及 Bruhat 形式矩阵之和生成矩阵的新型快速算法。

表1展示了论文中介绍的三种格式下，对一个 $n \times n$ s-准可分矩阵执行不同操作的最佳成本估算。Bruhat 格式实现了最佳且最优的存储大小，同时其生成器计算算法也最快。然而，这并未体现在以下操作的成本中：使用 SSS 生成元将准可分矩阵作用于稠密矩阵的成本最低，而以 Bruhat 形式给出的 $n \times n$ 矩阵的加法和乘积，其复杂度关于 n呈超线性。我们在命题2.5中注意到，HSS 的成本是 SSS 的两倍，在我们的语境中并无优势。因此，我们只比较到生成器计算为止。表1中仍给出了准可分 × 稠密乘积的成本，该成本与生成器大小 [16]成正比。最后，我们通过实验补充分析：尽管 SSS 的渐近成本估算略逊一筹，但在第3.5节所述的构造以及第4.3节中与稠密块向量的乘积运算中，实际表现优于 Bruhat。

2 格式的表示

2.1 SSS 生成器

SSS 生成器于 [9], 中引入，随后在 [10] 和 [4] 中被独立改进，采用了本文将介绍的块版本。特别是，所需空间从 $O(ns^2)$ 减少到 $O(ns)$。

s-准可分矩阵按由 $s \times s$ 个块组成的网格进行切分。对角线上的块、上方的块和下方的块分别处理。在对角线的一侧，每个块由一个乘积定义，该乘积取决于其所在的行（乘积中最左侧的块）、列（乘积中最右侧的块）以及它与对角线的距离（乘积中的块数）。

定义 *2.1.* 设 $A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,N} \\ \vdots & & \vdots \\ A_{N,1} & \cdots & A_{N,N} \end{bmatrix} \in K^{n\times n}$，其 $t\times t$ 块为 $A_{i,j}$，对于 $i,j < N$ 和 $N = \lceil n/t \rceil$. 若 A 由阶数为 t 的顺序半可分格式给出，则称其为 *t(t-SSS)* 格式，该格式由矩阵 $(P_i, V_i)_{i\in[\![2,N]\!]}$, $(Q_i, U_i)_{i\in[\![1,N-1]\!]}$, $(R_i, W_i)_{i\in[\![2,N-1]\!]}$, $(D_i)_{i\in[\![1,N]\!]}$ 给出，且满足：

$$A_{i,j} = \begin{cases} P_i R_{i-1} \ldots R_{j+1} Q_j & \text{if } i > j \\ D_i & \text{if } i = j \\ U_i W_{i+1} \ldots W_{j-1} V_j & \text{otherwise} \end{cases} \tag{1}$$

命题 2.2. 任意 $n\times n$ s-准可分矩阵都有一个 s-SSS 表示。它使用 $\text{SSS}(n, s) = 7ns$ 个域元素。

证明。由命题 3.1 直接得出。 □

2.2 HSS 生成元

HSS格式最早由 [6],提出，不过这一思想源自 [12] 的均匀 H矩阵 ，而 [14], 的H^2矩阵 则对其进行了更详细的阐述，其算法依赖于 [23]。H^2 格式与HSS略有不同，更多详情见 [13]。

该格式与SSS（见命题2.4）相近，因为块的定义方式类似。不过，切分网格是递归构造的，且块乘积的定义取决于递归树中所遵循的路径。此外，对角线两侧是联合处理的，因此该格式的紧凑性较低；正如后文将要说明的，这使得HSS的效率更低。

这种结构较为复杂，文献中的记法也不尽相同。我们作出如下选择：避免采用源自快速多极子方法 [6] 的递归树定义，因此只考虑递归深度固定的块划分。作出这一选择是为了专注于线性代数和拟可分矩阵，不需要任何先验知识（例如秩的位置）。出于同样的原因，我们专注于均匀划分。大多数关于HSS的文献采用非均匀网格，以适应拟可分秩结构内部具有结构的矩阵 [6]。尽管这种做法更具一般性，但会带来不必要的混淆。

我们使用一种与 [27]转移矩阵类似的记号。

定义 2.3. 令 $A \in \mathrm{K}^{n\times n}$ 以及统一的分块划分

$$A = \begin{bmatrix} A_{k;1,1} & \cdots & A_{k;1,2^k} \\ \vdots & & \vdots \\ A_{k;2^k,1} & \cdots & A_{k;2^k,2^k} \end{bmatrix}. \tag{2}$$

A 若A 由 $t\times t$矩阵$(U_{K;i}, V_{K;i}, D_i)_{i\in[\![1,N]\!]}$, $(R_{k;i}, W_{k;i})_{k\in[\![2,K]\!],i\in[\![1,2^k]\!]}$和$(B_{k;i})_{k\in[\![1,K]\!]}$ 给出，且满足 $N = \lceil n/t \rceil$和 $K \geq \log N$，使得对于$i\in[\![1, 2^k]\!] \in [\![1, N]\!]$, $A_{K;i,i} = D_i$，如果我们从$K-1$到1对 k递归定义 $i \in [\![1, 2^k]\!]$, $U_{k;i} = \begin{bmatrix} U_{k+1;2i-1}R_{k+1;1} & U_{k+1;2i}R_{k+1;2} \end{bmatrix}$和$V_{k;i} = \begin{bmatrix} W_{k+1;2i-1}V_{k+1;2i-1} & W_{k+1;2i}V_{k+1;2i} \end{bmatrix}$，那么称其为阶数为 t 的层次半可分格式（*t-HSS*）

$$\begin{aligned} A_{k;2i-1,2i} &= U_{k;2i-1}B_{k;2i-1}V_{k;2i} \\ A_{k;2i,2i-1} &= U_{k;2i}B_{k;2i}V_{k;2i-1} \end{aligned} \tag{3}$$

HSS 生成元可以看作是一种递归 SSS 生成元，但有两点不同：一是使用 B 矩阵，二是变换矩阵的分布方式。命题2.4清楚地说明了这种相似性。

命题2.4。 令 $U_{K;i}, V_{K;i}, D_i, R_{k;i}, W_{k;i}, B_{k;i}$ 为适当的 $k \le K$，$i \le 2^k$ 的 t-HSS 生成元。令 $I, J \in [\![1, 2^K]\!]$ 以及 k 为最高递归层级，使得 $A_{K;I,J}$ 未包含在对角块中。对于 $i_1 = \lfloor I/2^{K-k-1} \rfloor$，$i_0 = \lfloor I/2^{K-k} \rfloor$ 和 $j_1 = \lfloor J/2^{K-k-1} \rfloor$，我们有

$$A_{K;I,J} = U_{K;I} R_{K;I} \dots R_{k+1;i_1} B_{k;i_0} W_{k+1;j_1} \dots W_{K;J} V_{K;J}. \tag{4}$$

证明。对等式 (3) 使用归纳法。 □

命题2.5。 任意 $n \times n$ s-准可分矩阵都有一个 $2s$-HSS 表示。这是最优块参数，该表示使用 $\text{SHSS}(n, s) = 18ns$ 个域元素。

证明。由命题 3.2 可得。为证明最优性，设 A 以 s-准可分的 t-HSS 形式给出。我们使用命题 2.4：

$$\begin{bmatrix} A_{K;3\dots4,1\dots2} & A_{K;3\dots4,5\dots6} \end{bmatrix} = \begin{bmatrix} U_{K;3} R_{K;3} \\ U_{K;4} R_{K;4} \end{bmatrix} H \tag{5}$$

其中 $H \in K^{t \times 4t}$。A 的准可分性将式 (5) 左侧部分的秩限定为 $2s$，而右侧部分的秩则限定为 t。当第一个界达到紧致时，我们得到 $t \ge 2s$。 □

2.3 Bruhat 生成器

Bruhat 生成器最早在 [19, 21] 中定义。与 SSS 和 HSS 不同，它不使用预定义网格，而是依赖于准可分矩阵的下三角部分和上三角部分的秩剖面矩阵 [8] 中所包含的秩剖面信息。

回顾 [21]，如果矩阵是 t-重叠的，则其任意包含 $t+1$ 个非零列（相应地，行）的子集，至少有一列（相应地，一行）的首个非零元素位于另一列（相应地，另一行）的末个非零元素之后（相应地，之前）。我们将 J_n 称为维数为 n 的反单位矩阵，并定义 左 算子：$\mathrm{K}^{n \times n} \to \mathrm{K}^{n \times n}$，使得 $\triangledown$

$$\triangledown(A)_{i,j} = \begin{cases} A_{i,j} & \text{if } i + j \le n \\ 0 & \text{otherwise} \end{cases}. \tag{6}$$

定义 *2.6.* 若一个 $n \times n$ 矩阵 A 由一个对角矩阵 $D \in K^{n \times n}$ 和 6 个矩阵 $C_{(L)}, R_{(L)}, E_{(L)}, C_{(U)}, R_{(U)}, E_{(U)}$ 给出，则称其为 *t-Bruhat* 格式，其中 $C_{(L)} \in K^{n \times u}$ 和 $C_{(U)} \in K^{n \times v}$ 为列阶梯形且 t-重叠，$E_{(L)} \in K^{u \times n}$ 和 $E_{(U)} \in K^{v \times n}$ 为列阶梯形且 t-重叠，并且 $R_{(L)} \in K^{u \times u}$、$R_{(U)} \in K^{v \times v}$ 为置换矩阵，且满足

$$A = D + \mathrm{J}_n \triangledown \left(C^{(L)} R^{(L)} E^{(L)} \right) + \triangledown \left(C^{(U)} R^{(U)} E^{(U)} \right) \mathrm{J}_n$$

命题 2.7. 任何 $n \times n$ s-准可分矩阵都有一个 s- Bruhat 表示。它使用 $\text{SBruhat}(n, s) = 4ns$ 个域元素，这是最优的。

证明。根据 [21, 定理 20]。由于表示所有秩为 s 的三角矩阵需要 $2ns$ 个系数，因此 $4ns$ 是最优的。 □

3 生成元的构造

3.1 从稠密矩阵构造 SSS 生成元

我们回顾 DenseToSSS 算法中如何从一个稠密的 s-准可分矩阵 $A \in \mathrm{K}^{n \times n}$ 构造 SSS 生成器。该方法改编自 [4, §6.1] 和 [10, 算法 6.5]，其中以 SVD 为基础的数值秩揭示分解被秩揭示分解所替代。

块 D_i 直接从稠密矩阵的第 3 行中提取。随后分别对每个块三角部分进行压缩。每一步都会消去一个由 A 的块行和上一步的余项组成的块。最终得到生成器的三个块，以及一个将在后续步骤中消去的余项。

Algorithm 3.1 DenseToSSS

Input: A an $n \times n$ s-quasi-separable matrix with $s \le t$
Output: $P_i, Q_i, R_i, U_i, V_i, W_i, D_i$ for appropriate $i \in [\![1, N]\!]$ a t-SSS representation of A

1: $A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,N} \\ \vdots & & \vdots \\ A_{N,1} & \cdots & A_{N,N} \end{bmatrix}, H = \begin{bmatrix} H_{0,1} & \cdots & H_{0,N} \\ \vdots & & \vdots \\ H_{N,1} & \cdots & H_{N,N} \end{bmatrix} \leftarrow 0$
2: **for** $k = 1 \dots N-1$ **do**
3: $\quad D_k \leftarrow A_{k,k}$
4: $\quad \left(\begin{bmatrix} W_k \\ U_k \end{bmatrix}, \begin{bmatrix} V_{k+1} & H_{k,k+2\dots N} \end{bmatrix} \right) \leftarrow \mathsf{RF}\left(\begin{bmatrix} H_{k-1,k+1\dots N} \\ A_{k,k+1\dots N} \end{bmatrix} \right)$
5: $\quad \left(\begin{bmatrix} Q_{k+1} \\ H_{k+2\dots N,k} \end{bmatrix}, \begin{bmatrix} R_k & P_k \end{bmatrix} \right) \leftarrow \mathsf{RF}\left(\begin{bmatrix} H_{k+1\dots N,k-1} & A_{k+1\dots N,k} \end{bmatrix} \right)$
6: $D_N = A_{N,N}$

命题 3.1。 *DenseToSSS* 算法通过 t-SSS 生成器，为一个 n $2C\ n^2 s^{\omega-2}$ -准可分矩阵 $(s \le t)$ 计算出 $\mathsf{TDenseToSSS}(\cdot, t)$ 秩揭示分解域运算。

证明。对于 $k \in [\![1, N-1]\!]$，第 4 行和第 5 行输出的维度是足够的，因为分解的输入是同一对角线侧面上一个块与另一个 A 的块的秩揭示因子的串接，因此其秩至多为 s。

令 $i, j \in [\![1, N]\!]$。若 $i = j$，则针对 $k = i$ 的第 3 行给出 $D_i = A_{i,i}$。若 $i < j$，第 4 行给出

$$W_{j-1} V_j = H_{j-2,j} \tag{7}$$
$$W_k H_{k,j} = H_{k-1,j} \qquad (k \in [\![1, j-2]\!]) \tag{8}$$
$$U_i H_{i,j} = A_{i,j} \qquad (i < N) \tag{9}$$
$$U_i V_{i+1} = A_{i,i+1} \tag{10}$$

将其合并得到 $U_i W_{i+1} \dots W_{j-1} V_j = A_{i,j}$。同理，若 $i > j$，则 $P_i R_{i-1} \dots R_{j+1} Q_j = A_{i,j}$。
The cost is $\sum_{k=1}^{N-1} 2\mathsf{T}_{\mathsf{RF}}(t(N-k), 2t, s) = 2C_{\mathsf{RF}} n^2 s^{\omega-2}$. □

3.2 从稠密矩阵构造 HSS 生成元

文献 [6] 给出了第一个适用于一般准可分矩阵的构造算法。我们在算法 DenseToHSS 中介绍了文献 [27] 中更快、更简单算法的迭代版本。

循环的每一步 k 都会对从上一步继承的矩阵逐块行、逐块列进行处理，每次将块行和块列两两分解。在每一步

中，每个块因此会被分解两次，生成转移矩阵R和W；余项则要么传递到下一步，要么最终存储为一个 B矩阵。

算法 3.2 DenseToHSS

输入：A一个 $n \times n$ 阶数为 s 的准可分矩阵
输出：$U_{K;\,i}$, $V_{K;\,i}$, D_i, $R_{k;\,i}$, $W_{k;\,i}$, $B_{k;\,i}$适当的 $k \le K$, $i \le 2^k$ 一个 t-HSS 表示，其中 A具有 $t \ge 2s$

1: $H \leftarrow A$ ▷ 使用式 (2) 的分块划分，其中 $k = K$
2: **对** $i = 1 \ldots 2^K$ **执行**
3: $D_i \leftarrow A_{K;i,\,i}$
4: **对** $k = K \ldots 1$ **执行**
5: **对** $i = 1 \ldots 2^k$ **执行** ▷ 所有操作都在此循环中
▷ $R_{K+1;2i}$行（分别为列）维度为 0
6: $\left(\begin{bmatrix} R_{k+1;\,2i-1} \\ R_{k+1;\,2i} \end{bmatrix}, \begin{bmatrix} H''_{k;i,\,1\ldots i-1} & H'_{k;i,\,i+1\ldots 2^k} \end{bmatrix}\right) \leftarrow$ 秩揭示分解 $\left(\begin{bmatrix} H'_{k;1\ldots i-1\,i,\,:,i} \\ H_{k;i+1\ldots 2^k,\,:,i} \end{bmatrix}\right)$
7: $\left(\begin{bmatrix} H''_{k;1\ldots i-1\,i,\,:,i} \\ H'_{k;i+1\ldots 2^k,\,:,i} \end{bmatrix}, \begin{bmatrix} W_{k+1;2i-1} & W_{k+1;2i} \end{bmatrix}\right) \leftarrow$ 秩揭示分解
8: **对于** $i = 1 \ldots 2^{k-1}$ **执行** ▷ 从这里开始仅更名
9: $B_{k;\,2i-1} \leftarrow H''_{k;\,2i-1,\,2i}$
10: $B_{k;\,2i} \leftarrow H''_{k;\,2i,\,2i-1}$
11: **对于** $j = 1 \ldots 2^{k-1}$, $j \ne i$ **则**
12: $H_{k-1;i,j} \leftarrow \begin{bmatrix} H''_{k;2i-1,\,2j-1} & H''_{k;2i-1,\,2j} \\ H''_{k;2i,\,2j-1} & H''_{k;2i,\,2j} \end{bmatrix}$
13: $H_{k-1;j,i} \leftarrow \begin{bmatrix} H''_{k;2j-1,\,2i-1} & H''_{k;2j-1,2i} \\ H''_{k;2j,\,2i-1} & H''_{k;2j,\,2i} \end{bmatrix}$
14: **对于** $i = 1 \ldots 2^K$，**执行**
15: $U_{K;\,i} \leftarrow R_{K+1;\,2i-1}$
16: $V_{K;i} \leftarrow W_{K+1;2i-1}$

命题 3.2. *DenseToHSS* 算法为 s-准可分矩阵计算一个 t-HSS 生成器，只需 $2s \le t$ $C_{\text{秩揭示分解}} n^2 t^{\omega-2}$ 次域运算。对于 $t = 2s$，其计算量为 $\mathsf{T}_{\text{DenseToHSS}}(n, s) = 2^\omega C_{\text{秩揭示分解}} n^2 s^{\omega-2}$。

证明。令 $k \in [\![1, K]\!]$, $i \ne j \in [\![1, 2^k]\!]$。第 6 行和第 7 行中输出的维数是充分的，因为每次进行因式分解的矩阵都是两个秩至多为 s 的块的串接，因此秩至多为 $2s \le t$。若 $|i - j| = 1$，这些指令给出

$$H_{k;i,j} = \begin{bmatrix} R_{k+1;2i-1} \\ R_{k+1;2i} \end{bmatrix} B_{k;i} \begin{bmatrix} W_{k+1;2j-1} & W_{k+1;2j} \end{bmatrix}. \quad (11)$$

否则，

$$H_{k;i,j} = \begin{bmatrix} R_{k+1;2i-1} \\ R_{k+1;2i} \end{bmatrix} H''_{k;i,j} \begin{bmatrix} W_{k+1;2j-1} & W_{k+1;2j} \end{bmatrix}. \quad (12)$$

现在令 $I, J \in [\![1, N]\!]$。若 $I = J$，第 3 行给出 $D_I = A_{K;I,J}$。否则，令 k 为最高递归层级，使得 $A_{K;I,J}$ 未包含在对角块中。根据第 1 行，$A_{K;\,I,J} = H_{K;I,J}$。公式 (12) 可使用 $K - k$ 次，再结合第 12 行得到

$$A_{K;I,J} = R_{K+1;2I-1} \ldots R_{k+2;i_2} H''_{k+1;i_1,j_1} W_{k+2;j_2} \ldots W_{K+1;2J-1} \quad (13)$$

其中，$i_2 = \lfloor I/2^{K-k-2} \rfloor$，$i_1 = \lfloor I/2^{K-k-1} \rfloor$，$j_1 = \lfloor J/2^{K-k-1} \rfloor$ 以及 $j_2 = \lfloor J/2^{K-k-2} \rfloor$。$R_{K+1;\,2I-1}$, $W_{K+1;\,2J-1}$ 以及 $H''_{k+1;\,i_1,\,j_1}$可以代入公式 (13)，利用第 15 行和第 16 行以及公式 (11)（根据 k 的定义，我们有 $|i_1 - j_1| = 1$），从而得到公式 (4)；至此，正确性证明完成。

第 6 行在 $k < K$和 i上对由 $2t \times (i-1)t$块 $H'_{k;i,1\ldots i-1}$以及 $2t \times 2t(2^k - i)$ 块 $H_{k;i,i+1\ldots 2^k}$构成的输入进行秩揭示分解，运算成本为 $\mathsf{TRF}(t(2^{k+1} - i),\ 2t,\ t)$。第 7 行的成本相同。因此，总成本为 $\sum_{k=1}^{\log n} \sum_{i=1}^{2^k} 4C\ \ 2^{k+1} t^\omega \le 4C\ \ n^2 t^{\omega-2} \le 2^\omega C\ \ n^2 s^{\omega-2}$. □

由于对角线两侧的各个块均由相同的矩阵定义，DenseToHSS 算法和任何 HSS 构造算法都会对这些块进行秩揭示分解；对于 s-拟可分矩阵，其秩上界为 $2s$，而 DenseToSSS 算法中的秩上界为 s。因此，s-拟可分矩阵的最优 HSS 块大小为 $2s$，这使得 HSS 在存储和运算成本方面效率较低。

由于成本较高，且 HSS 与 SSS 有相同的缺点，即需要固定切片网格和预先计算好的准可分性阶，因此我们不再详细介绍 HSS 的其他算法。关于数值环境下的相关信息，我们主要参阅 [16, 22]。请注意，还存在更快的构造算法：[18] 中的概率型算法，以及 [6]中带有输入约束的算法。

3.3 从稠密矩阵构造 Bruhat 生成器

从稠密矩阵构造 Bruhat 生成器可通过 [21, Alg. 12] 实现：对输入矩阵的上三角部分和下三角部分分别运行一次，并直接从稠密矩阵中提取对角矩阵 D。

我们在算法 LBruhatGen 中给出了 [21, Alg. 12], 的更新版本，其中将舒尔补计算延迟到需要时才进行。当输入不是以稠密矩阵形式给出时，这种做法可以加快计算速度；该算法还将用于计算 Bruhat 形式下两个矩阵之和，以及从稀疏矩阵构造生成元。

LBruhatGen 算法可以采用任意输入格式，前提是我们能够计算输入矩阵的任意子矩阵 B

(1) CRE(B, G, H) 对 $B - GH$T进行 CRE 分解；
(2) 对于 R 这个索引集合，$B_{\mathcal{R}*}$以及$B_{*,\,\mathcal{R}}$表示 B中索引属于 R 的行和列。

我们使用 TRSM 表示三角矩阵求解：TRSM(L, A) 对 L三角矩阵输出 $L^{-1}A$。

命题 3.3。可以在Bruhat 生成器中，从一个 $n \times n$稠密 s准可分矩阵计算得到 s-Bruhat 生成器，计算形式为$\mathsf{T}_{\text{DenseToB}}(n, s) = C_{\text{秩揭示分解}} n^2 s^{\omega-2}$。

证明。LBruhatGen 算法改编自 [21, 算法 12]；因此，我们引用 [21, 定理 24] 的证明来说明其正确性。除执行顺序不同外，当输入为稠密矩阵时，这两个算法中的操作完全相同，因此计算成本也相同。从稠密矩阵计算 Bruhat 生成器需要两次应用 LBruhatGen 算法。其成本满足：

$$\mathsf{T}_{\mathsf{LBG}}(n, s) \le C_{\mathsf{RF}}/4 n^2 s^{\omega-2} + 2\mathsf{T}_{\mathsf{LBG}}(n/2, s) \le C_{\mathsf{RF}}/2 n^2 s^{\omega-2}. \quad \square$$

3.4 从稀疏矩阵生成 Bruhat 生成元

在应用中，矩阵通常以稀疏结构呈现。为了检测和/或利用其准可分结构，

Algorithm 3.3 LBruhatGen

Input: $A \in \mathrm{K}^{m\times m}$ left triangular s_A-quasiseparable
Input: $G, H \in \mathrm{K}^{m\times t}$ ▷ $t = 0$ *on the first call*
Output: C, R, E a left-Bruhat generator for $A - \triangledown\left(GH^{\mathsf{T}}\right)$

1: Split $A = \begin{bmatrix} A^{(11)} & A^{(12)} \\ A^{(21)} & \end{bmatrix}, G = \begin{bmatrix} G^{(1)} \\ G^{(2)} \end{bmatrix}, H = \begin{bmatrix} H^{(1)} \\ H^{(2)} \end{bmatrix}$ where $A^{(11)} \in \mathrm{K}^{\frac{m}{2}\times\frac{m}{2}}$
2: $C_0, R_0, E_0 \leftarrow \mathsf{CRE}(A^{(11)}, G^{(1)}, H^{(1)})$
3: $\mathcal{R} \leftarrow \mathsf{RRP}(C_0); \mathcal{C} \leftarrow \mathsf{CRP}(E_0); r_0 \leftarrow \#\mathcal{R}$
4: $\begin{bmatrix} U & V \end{bmatrix} \leftarrow E_0 Q_{\mathcal{C}}$ where $U \in \mathrm{K}^{r_0\times r_0}$ is upper triangular.
5: $\begin{bmatrix} L \\ M \end{bmatrix} \leftarrow P_{\mathcal{R}} C_0$ where $L \in \mathrm{K}^{r_0\times r_0}$ is lower triangular.
6: $X \leftarrow A^{(12)}_{\mathcal{R},*} - G^{(1)}_{\mathcal{R},*} H^{(2)\mathsf{T}}$
7: $B^{(12)} \leftarrow \triangledown(\mathsf{TRSM}(L, X))_{\mathcal{R},*}$ ▷ $A^{(12)}_{\mathcal{R},*} = \triangledown\left(LB^{(12)} + G^{(1)}_{\mathcal{R},*} H^{(2)\mathsf{T}}\right)_{\mathcal{R},*}$
8: $Y \leftarrow A^{(21)}_{*,\mathcal{C}} - G^{(2)} H^{(1)\mathsf{T}}_{\mathcal{C},*}$
9: $B^{(21)\mathsf{T}} \leftarrow \triangledown\left(\mathsf{TRSM}(U^{\mathsf{T}}, Y^{\mathsf{T}})\right)_{*,\mathcal{C}}$
▷ $A^{(21)}_{*,\mathcal{C}} = \triangledown\left(B^{(21)}U + G^{(2)} H^{(1)\mathsf{T}}_{*,\mathcal{C}}\right)_{*,\mathcal{C}}$
10: $C_1, R_1, E_1 \leftarrow$ $\mathsf{LBruhatGen}\left(A^{(21)}_{*,\overline{\mathcal{C}}} \mathrm{I}_{\overline{\mathcal{C}},*}, \begin{bmatrix} G^{(2)} & B^{(21)} \end{bmatrix}, \mathrm{I}_{*,\overline{\mathcal{C}}} \begin{bmatrix} H^{(1)}_{\overline{\mathcal{C}},*} & V^{\mathsf{T}} \end{bmatrix}\right)$
11: $C_2, R_2, E_2 \leftarrow$ $\mathsf{LBruhatGen}\left(\mathrm{I}_{*,\overline{\mathcal{R}}} A^{(12)}_{\overline{\mathcal{R}},*}, \mathrm{I}_{*,\overline{\mathcal{R}}} \begin{bmatrix} G^{(1)}_{\overline{\mathcal{R}},*} & M \end{bmatrix}, \begin{bmatrix} H^{(2)} & B^{(12)\mathsf{T}} \end{bmatrix}\right)$
12: $P_{01} \leftarrow$ the permutation which sorts the rows of E_0 and E_1 by increasing column of pivot
13: $P_{02} \leftarrow$ the permutation which sorts the columns of C_0 and C_2 by increasing row of pivot
14: $C \leftarrow \begin{bmatrix} C_0 & C_2 \\ B^{(21)} R_0^{\mathsf{T}} & C_1 \end{bmatrix} \begin{bmatrix} P_{02} & \\ & I \end{bmatrix}$
15: $R \leftarrow \begin{bmatrix} P_{02}^{\mathsf{T}} & \\ & I \end{bmatrix} \begin{bmatrix} R_0 & \\ & R_2 \\ R_1 & \end{bmatrix} \begin{bmatrix} P_{01}^{\mathsf{T}} & \\ & I \end{bmatrix}$
16: $E \leftarrow \begin{bmatrix} P_{01} & \\ & I \end{bmatrix} \begin{bmatrix} E_0 R_0^{\mathsf{T}} B^{(12)} \\ E_1 \\ & E_2 \end{bmatrix}$
17: **return** C, R, E

在构造准可分生成元时，充分利用稀疏性至关重要。

对于 Bruhat 生成器的构造，可以将通用算法 LBruhatGen 算法应用于稀疏矩阵，前提是对以下两项操作进行专门处理：

(1) 将 $\le s$ 行或列的子集提取为稠密格式，这对稀疏矩阵而言很直接；(2) 计算 CRE分解，该操作在 SparseCRE 算法中经过专门处理，而该算法又使用稀疏秩剖面算法

引理 3.4. *SparseRankProfiles* 算法的正确概率至少为 $1 - 2r\ |S|$ $n\ r\ =\ 2\ C_\omega\ +/$，运行时间为 $C\quad nr^{\omega-1} + 2r|A| \qquad r = t + s$ $\mathsf{T}_{\mathsf{SparseRP}}(,)$(秩揭示分解)，其中 。

证明。在第 3 行和第 4 行应用托普利兹预条件器的代价为 $\frac{nt}{r}\tilde{O}(r)$，这被 $nr^{\omega-1}$ 主导。□

算法 3.4稀疏CRE

输入： $A \in \mathrm{K}^{m\times m}$ 秩为 $\le s$ 的稀疏矩阵
输入： $G, H \in \mathrm{K}^{m\times t}$
输出： C， R， E使得 $A = CRE + GH\mathsf{T}$

1: $\mathrm{R}, \mathrm{C} \leftarrow$ 稀疏秩剖面(A, G, H)
2: $P = \begin{bmatrix} \mathrm{IR}_{*} & \\ & \mathrm{IR} \\ , & \end{bmatrix}; Q = \begin{bmatrix} \mathrm{I}_{*,} & \\ & \mathrm{C} \end{bmatrix}$
▷与 $\bar{A}^{(11)} \in \mathrm{K}^{|\mathcal{R}|\times|\mathcal{R}|}$ 写入 $P\left(A - GH^{\mathsf{T}}\right)Q = \begin{bmatrix} \bar{A}^{(11)} & \bar{A}^{(12)} \\ \bar{A}^{(21)} & \bar{A}^{(22)} \end{bmatrix} - \begin{bmatrix} \bar{G}^{(1)} \\ \bar{G}^{(2)} \end{bmatrix} \begin{bmatrix} \bar{H}^{(1)(2)} \\ \bar{H} \end{bmatrix}^{\mathsf{T}}$
3: $M(11) \leftarrow \bar{A}^{(11)} - \bar{G}^{(1)}(\bar{H}(1))\mathrm{T}$
4: $M(12) \leftarrow \bar{A}^{(12)} - \bar{G}^{(1)}(\bar{H}(2))^{\mathsf{T}}$
5: $M(21) \leftarrow \bar{A}^{(21)} - \bar{G}^{(2)}\ (\bar{H}(1))^{\mathsf{T}}$
6: $(L,\ R,\ U) \leftarrow$稠密CRE分解$\left(M\ (11)\right)$
7: $C \leftarrow \mathrm{TRSM}(L, \bar{M}(12))$ ▷ $C = L^{-1}(\bar{A}^{(12)} - G(1)H(2))$
8: $D \leftarrow \mathrm{TRSM}(\bar{A}(21)U\mathrm{T})$ ▷ $D = (\bar{A}^{(21)} - G(2)H(1))U^{-1}$
9: $E \leftarrow \begin{bmatrix} U & R\mathrm{T}C \end{bmatrix} Q^{\mathrm{T}}$
10: $C \leftarrow P\mathrm{T} \begin{bmatrix} L^{\mathrm{T}} \\ DR \end{bmatrix}$
11: **返回** (C, R, E)

算法 3.5稀疏秩剖面

输入： $A \in \mathrm{K}^{n\times n}$，一个秩为 $\le s$ 的稀疏矩阵。输入： G, $H \in \mathrm{K}^{n\times t}$，稠密矩阵。输出：$\mathrm{R}_A, \mathrm{C}_A$， $A - GH\mathrm{T}$ 的行秩剖面和列秩剖面。1： $T(1) \leftarrow$，从 $S \subseteq \mathrm{K}$ 中取一个均匀随机的 $n \times (s + t)$ 托普利兹矩阵。2： $T(2) \leftarrow$，从 $S \subseteq \mathrm{K}$ 中取一个均匀随机的 $(s + t) \times n$ 托普利兹矩阵。3： $K \leftarrow HTT(1)$。4： $L \leftarrow T(2)G$。5： $P \leftarrow AT(1) - GK$。6： $Q \leftarrow T(2)A - LH\mathrm{T}$。7：返回 RowRankProfile($P$), ColRankProfile($Q$)

命题 3.5. *SparseCRE* 算法以至少 $1-2r/|S|$ 的概率计算 $A-GH\mathrm{T}$ 的 *CRE*分解，在 $\mathsf{T}_{\mathsf{SparseCRE}}(n, r) = \left(\frac{2^\omega-3}{2^{\omega-2}-1}C_\omega + 2C_{\text{秩揭示分解}}\right) nr^{\omega-1} + 2r|A|$ 域运算内完成，适用于 $s + t \le r$。

证明。设 ρ 为 $A - GH\mathrm{T}$ 的秩。

$$\begin{aligned}\mathrm{TSparseCRE}(n, r) &= 2\mathrm{TMM}(n, r, t) + \mathrm{TCRE分解}(\rho, \rho, \rho) + 2\mathrm{TTRSM}(n - \rho, \rho) \\ &\quad + \mathsf{T}_{\mathsf{SparseRP}}(n, r) \\ &\le nr^{\omega-1}\left(4C_\omega + \frac{2C_\omega}{2^{\omega-1}-2} + 2C_{\mathsf{RF}}\right) + 2r|A|.\end{aligned}$$

□

命题 3.6. *LBruhatGen* 算法从一个稀疏 s-准可分矩阵 $A \in \mathrm{K}^{n\times n}$ 中计算出一个左 *Bruhat* 生成元

$$\mathsf{T}_{\mathsf{SpGenB}}(n, s, |A|) = \left(\frac{2^{\omega+1}-9}{2^{\omega-1}-2}C_\omega + C_{\mathsf{RF}}\right) ns^{\omega-1}\log n/s + 2s|A|$$

域运算的成功概率至少为 $1 - 2n/|S|$。

证明。首先注意，G和 H矩阵对应于此前调用中已处理主元的延迟舒尔补更新。因此，在每次调用 LBruhatGen 算法时，这些主元都位于工作矩阵的左侧、顶部或左上角。拟可分条件要求它们的数量为 $t \le 2s$。此外，在调用 SparseCRE 算法时，各阶满足 $r_A + r_B + t \le s$。因此，我们可以对 t 进行界定，并将 LBruhatGen 算法的代价仅用以下因素表示：n 矩阵的维数、s 初始准可分性阶，以及 $|\cdot|$ 所考虑子矩阵中非零系数的数量。

$$\begin{aligned} T(n,s,|A|) \le\ & T(n/2,s,|A_2|) + T(n/2,s,|A_3|) \\ & + \mathsf{T}_{\mathsf{SparseCRE}}(n/2,s,|A_1|) \\ & + 2\mathsf{T}_{\mathsf{MM}}(s,2s,n/2) + 2\mathsf{T}_{\mathsf{TRSM}}(s,n/2) \\ \le\ & T(n/2,s,|A_2|) + T(n/2,s,|A_3|) + 2s|A_1| \\ & + \left(\frac{2^{\omega+1}-9}{2^{\omega-1}-2}C_\omega + C_{\mathsf{RF}}\right) n s^{\omega-1}. \end{aligned}$$

失败概率由对 n/s 次 SparseCRE 算法调用中各次失败概率应用并集界得到。□

据我们所知，目前还没有利用输入矩阵的稀疏性来计算 SSS 或 HSS 生成元的类似算法，因此只能将我们的结果与从稠密矩阵进行二次生成的方法进行比较。

3.5 实验比较

为补充渐近计算成本分析，我们在图 1 中给出了构造 SSS 和 Bruhat 生成元的计算时间对比实验。Bruhat 的计时结果关于 s呈次线性增长，这与命题 3.3 的预期相符，但也会受到 r的轻微影响，这源于被忽略的成本，例如大量置换所产生的成本。对于我们测试的数值，SSS 的成本保持不变，原因目前尚无法解释。SSS 的成本几乎总是低于 Bruhat 计算成本。不过请记住，DenseToSSS 算法需要将拟可分离阶作为输入，因此必须先计算该阶（例如使用 LBruhatGen 算法）。

4 应用于块向量

这里研究由其生成元（SSS 或 Bruhat）给出的 s-准可分矩阵 $A \in \mathrm{K}^{n\times n}$ 对一组 v 向量 $B \in \mathrm{K}^{n\times v}$的应用。我们给出 $v \le s$ 的计算开销（也可以通过将 B 切分为每块包含 s 列来推导）。

4.1 SSS ×稠密

这里我们回顾 [4, §2] 中的算法，该算法用于计算 SSS 矩阵与稠密矩阵的乘积（独立发表于 [10, 算法 7.1]）。为简洁起见，LowSSSxDense 算法仅详述与严格下块三角 SSS 矩阵相关的计算，即其 SSS 表示除 P_i, Q_i和 R_i之外均为零的矩阵。由此推广到与任意 SSS 矩阵的乘积，可通过对上块三角部分使用转置算法，并加上与由 D_i构成的块对角矩阵的乘积来实现。

算法 4.1低SSS乘稠密

输入：P_i，Q_i，R_i对于 $i \in [\![1,\ N]\!]$一个严格的 s-SSS 生成元
下块三角矩阵 A；B和 C稠密 $n \times v$矩阵
输出：$C += AB$

1：拆分 $B = \begin{bmatrix} B_1 \\ \vdots \\ B_N \end{bmatrix}$, $C = \begin{bmatrix} C_1 \\ \vdots \\ C_N \end{bmatrix}$ 在 $s \times s$ 块中
2：$H_1 \leftarrow Q_1 B_1$
3：**对于** $i = 2 \ldots N$**执行**
4：$H_i \leftarrow Q_i B_i + R_i H_{i-1}$
5：$C_i \leftarrow C_i + P_i H_{i-1}$

命题 4.1。 </style}由其 -SSS</style}生成元给出的矩阵，与具有的稠密矩阵的乘积，可在 $\mathsf{T}_{\mathsf{SxDense}}(n,s,v) = 7C_\omega n s v^{\omega-2}$内计算。

证明。在 LowSSSxDense 算法中，通过归纳法可得

$$\text{for } i \in [\![1,N]\!], H_i = \sum_{j=1}^{i} R_i \ldots R_{j+1} Q_j B_j. \tag{14}$$

由于乘积的各块满足

$$C_i = P_i \sum_{j=1}^{i-1} R_{i-1} \ldots R_{j+1} Q_j B_j, \tag{15}$$

H_{i-1}可与 P_i 相乘一次以计算 C_i，并与 R_i 相乘一次以计算以下各块。对角块的计算成本为 $N \times C_\omega s^2 v^{\omega-2}$，此外还需两次调用 LowSSSxDense 算法，每一步的成本为 $3C_\omega s^2 v^{\omega-2}$。□

4.2 Bruhat × 稠密

命题 4.2。 由其 s-Bruhat 生成元给出的 $n \times n$ 矩阵，与一个满足 $v \le s$ 的稠密 $n\times v$矩阵的乘积，可在 $\mathsf{T}_{\mathsf{BxDense}}(n,s,v) = 8C_\omega n s v^{\omega-2}$ 内计算得到。

证明。这可通过将 [21, Alg. 14] 分别应用于准可分矩阵的下三角部分和上三角部分两次来实现。□

请注意，为了从快速矩阵乘法中获益，需要将 Bruhat 生成器（使用 $4ns$ 空间）转换为紧致Bruhat形式：将每个阶梯形存储到两个分块对角矩阵中，所用域元素数量增加一倍（新增元素为零）。这种压缩可以在线完成，因此空间存储仍为 $4ns$，但与稠密矩阵相乘的代价变为 $8C_\omega n s t^{\omega-2}$，因而失去了相较于 SSS 格式的优势（后者执行相同操作的代价为 $7C_\omega n s v^{\omega-2}$）。

4.3 实验比较

实验结果如图4（）。正如命题4.1和4.2所预期的那样，我们得到的成本关于s呈线性；还可以观察到，Bruhat计算成本对 r的轻微依赖性与第3.5节中的情况相同。在我们选择的参数下，SSS的速度约为Bruhat的四倍。这可以用乘积所需的Bruhat生成器的紧致化来解释。该操作不涉及算术运算，因此不会

出现在命题 4.2的成本中，但在实践中，数据传输的开销不可忽略。

5 拟可分矩阵之和 ES

两个阶数分别为 s_B和 s_C的拟可分矩阵，其和与乘积都是阶数至多为 $s_B + s_C$的拟可分矩阵。本节将介绍如何计算两个拟可分矩阵之和的 SSS 生成元和 Bruhat 生成元。

命题 5.1 给出的关于 SSS 形式矩阵求和的结果，只能用于定义在同一网格上的两个生成元。这是 SSS 中大多数运算的一个缺点，而 Bruhat 格式则避免了这一问题。因此，在一系列运算中，SSS 网格大小需要根据所有中间结果中的最大拟可分离阶数来选择，而 Bruhat 格式始终与当前的拟可分离阶相匹配。这会影响总成本。 [9] 的原始 SSS 格式速度较慢，但可避免此问题，代价是空间和时间成本都要乘以拟可分性阶，如 $[1, 2]$所示。

5.1 SSS 求和

考虑两个阶数相同的矩阵 B和 C，其阶数为 s。首先注意到，将两个输入生成元的块串接后得到的矩阵满足式 (1)，其中 $A = C + B$ [4, §10.2]。

令 $P_i \quad V_i \quad Q_i \quad U_i \quad R_i \quad W_i \quad D_i \qquad i \in (K),\ (K),$ $[\![1\ N]\!] \quad s \qquad\qquad K \quad K \in \ B\ C$ $(K),\ (K),\ (K),\ (K),\ (K)$对于适当的，构成 $\{,\ \}$的 -SSS 表示。以下矩阵在适当的 $i \in [\![1,\ N]\!]$下满足式 (1)，其中 $A = B+C$。

$$P_i = \begin{bmatrix} P_i^{(B)} & P_i^{(C)} \end{bmatrix}, Q_i = \begin{bmatrix} Q_i^{(B)} \\ Q_i^{(C)} \end{bmatrix}, R_i = \begin{bmatrix} R_i^{(B)} & \\ & R_i^{(C)} \end{bmatrix} \tag{16}$$

$$U_i = \begin{bmatrix} U_i^{(B)} & U_i^{(C)} \end{bmatrix}, V_i = \begin{bmatrix} V_i^{(B)} \\ V_i^{(C)} \end{bmatrix}, W_i = \begin{bmatrix} W_i^{(B)} & \\ & W_i^{(C)} \end{bmatrix} \tag{17}$$

$$D_i = D_i^{(B)} + D_i^{(C)} \tag{18}$$

将满足式 (1) 且维数如上的这组矩阵称为 A 的 $(s,\ 2s)$-SSS 生成元。其描述粒度仍为 $s \times s$块，但表示中矩阵的维数加倍，导致存储大小不理想。因此，第二步是使用算法 SssCompression 得到该和的 $2s$-SSS 生成元，并将存储大小减少 $4s(n - 2s)$。

命题 5.1。 $2s$-SSS 表示可由 s-SSS 表示在时间内计算得到，其中 $B + C \in \mathrm{K}^{n\times n}$，且输入为 B和 C

$$\mathsf{T}_{\mathsf{S+S}}(n, s) \leftarrow \left(10 + 2^{\omega}\right) C_{\omega} n s^{\omega-1}. \tag{19}$$

证明。对于 $A = B + C$ 的任意 $s \times s$块 $A_{i,j}$，可以验证，对第 5.1 节的生成元调用 SssCompression 算法后，其输出中的表示与之匹配。式 (18) 中的加法，其计算量主要由调用 SssCompression 算法决定，该算法的代价为 M步，其中包括四次 $2s \times 2s$乘以 $2s \times s$的乘积、两次 $2s \times 2s$方阵乘积，以及两次 $s \times 2s$乘以 $2s \times s$的乘积。□

注意，$(s, 2s)$-SSS 生成元介于 SSS 形式与 [9], 中给出的拟可分矩阵原始定义之间；在该定义中，生成元是 $s \times s$ 矩阵，但描述的粒度为 1 维。

算法5.1 SssCompression

输入： $P_i, Q_i, R_i, U_i, V_i, W_i, D_i$，对于适当的 $i \in [\![1, N]\!]$，一个 $(s, 2s)$-用于 $A \in \mathrm{K}^{n\times n}$的 SSS 生成元

输出： $P_{\mathit{ni}},\ Q_{\mathit{ni}},\ R_{\mathit{ni}},\ U_{\mathit{ni}},\ V_{\mathit{ni}},\ W_{\mathit{ni}},\ D_{\mathit{ni}}$对于适当的 $i \in [\![1,\ M]\!]$，一个 $2s$-A 的 SSS 表示，其中包含 $M = \lceil N/2 \rceil$

1: **for** $i \leftarrow 1 \ldots M$ **do**
2: $\quad P_i' \leftarrow \begin{bmatrix} P_{2i-1} \\ P_{2i}R_{2i-1} \end{bmatrix}$
3: $\quad Q_i' \leftarrow \begin{bmatrix} R_{2i}Q_{2i-1} & Q_{2i} \end{bmatrix}$
4: $\quad R_i' \leftarrow R_{2i}R_{2i-1}$
5: $\quad U_i' \leftarrow \begin{bmatrix} U_{2i-1}W_{2i} \\ U_{2i} \end{bmatrix}$
6: $\quad V_i' \leftarrow \begin{bmatrix} V_{2i-1} & W_{2i-1}V_{2i} \end{bmatrix}$
7: $\quad W_i' \leftarrow W_{2i-1}W_{2i}$
8: $\quad D_i' \leftarrow \begin{bmatrix} D_{2i-1} & U_{2i-1}V_{2i} \\ P_{2i}Q_{2i-1} & D_{2i} \end{bmatrix}$

5.2 布鲁哈特求和

与 SSS 类似，Bruhat 形式的两个矩阵之和可以通过先串接两个生成元，再在第二步提取 Bruhat 格式来计算。

给定两个左三角矩阵 A和 B，它们由 Bruhat 生成器 $C^{(A)}R^{(A)}E^{(A)}C^{(B)}R^{(B)}E^{(B)}$给出，它们的和确实可以写为

$$A + B = \mathbin{\triangledown}\left(\begin{bmatrix} C^{(A)} & C^{(B)} \end{bmatrix} \begin{bmatrix} R^{(A)} & \\ & R^{(B)} \end{bmatrix} \begin{bmatrix} E^{(A)} \\ E^{(B)} \end{bmatrix}\right). \tag{20}$$

公式（20）右侧的 Bruhat 生成器可以通过调用 LBruhatGen 算法获得，此处将其视为一种压缩算法。这依赖于一种特定的 CRE分解（BruhatSumCRE 算法），并且要求 $D\mathcal{R}*$是公式（20）中给出的和的一个子矩阵，且 R 是一组行索引（命题 5.3）。

算法5.2 BruhatSumCRE

Input: $A, B \in \mathrm{K}^{n\times n}$ of rank $\le r_A$ and $\le r_B$ given by generators $C^{(A)}, R^{(A)}, E^{(A)}, C^{(B)}, R^{(B)}, E^{(B)}$ s.t. $A = C^{(A)}R^{(A)}E^{(A)}$ and $B = C^{(B)}R^{(B)}E^{(B)}$ which are submatrices of Bruhat generators of matrices comprising A and B

Input: $G, H \in \mathrm{K}^{n\times t}$

Output: C, R, E such that $A + B = CRE + GH^{\mathsf{T}}$

1: $C^{(R)}, R^{(R)}, E^{(R)} \leftarrow \mathsf{DenseCRE}\left(\begin{bmatrix} R^{(A)}E^{(A)} \\ R^{(B)}E^{(B)} \\ -H^{\mathsf{T}} \end{bmatrix}\right)$
2: $C^{(L)}, R^{(L)}, E^{(L)} \leftarrow \mathsf{DenseCRE}\left(\begin{bmatrix} C^{(A)} & C^{(B)} & G \end{bmatrix}\right)$
3: $X \leftarrow R^{(L)}E^{(L)}C^{(R)}R^{(R)}$
4: $C^{(X)}, R^{(X)}, E^{(X)} \leftarrow \mathsf{DenseCRE}(X)$
5: $C \leftarrow C^{(L)}C^{(X)}$
6: $R \leftarrow R^{(X)}$
7: $E \leftarrow E^{(X)}E^{(R)}$

命题5.2. 算法*BruhatSumCRE*计算 $A + B - GH^{\mathsf{T}}$在$\mathsf{T}_{\mathsf{BSumCRE}}(n, r)$ $= (3C_{\omega} + 2C$ 秩揭示分解 $)\ nr^{\omega-1}$ 的 *CRE* 分解，其中 $r_A + r_B + t \le r$。

证明。矩阵 C和 E 分别为列阶梯形和行阶梯形，因为它们是两个阶梯形矩阵的乘积。其

代价为两次大小为 $n\times$（r_A+r_B+t）的稠密 CRE 分解，以及一个 $n\times$（r_A+r_B+t）矩阵与两个（r_A+r_B+t）$\times$（r_A+r_B+t）矩阵和一个（r_A+r_B+t）$\times n$矩阵的乘积。□

命题 5.3。对于 $D\in\mathrm{K}^{n\times n}$，它是式 (20) 中所示和的左三角部分的一个子矩阵；令 $\mathcal{R}$为一组 s行索引，则可在$\mathsf{T}_{\mathsf{SumExp}}(n,s)$ $=C_\omega ns^{\omega-1}$内计算 $D\mathcal{R}*$。

证明。影响 D的 A（分别为 B）主元至多有 s_A（分别为 s_B）个。因此，我们可以将 $D=CRE$ 写成 C，其中由 $\left[C_{(A)}C_{(B)}\right]$的 n 行和 s_A+s_B 列构成；R 是一个置换，而 E 由 $E_{(A)}$ 的 n 列和 s_A 行，以及 $E_{(B)}$ 的 s_B 行构成。□

命题 5.4。两个拟可分离阶为 s_A 和 s_B 的 $n\times n$ 矩阵之和的 Bruhat 形式，若两矩阵均为 Bruhat 形式，则可在 $\mathsf{T}_{\mathsf{B+B}}(n,s)=\left(9\cdot\frac{2^{\omega-2}-8}{2^{\omega-2}-1}C_\omega+2C^{\text{秩揭示分解}}\right)ns^{\omega-1}$对数 n/s 内计算，其中 $s=s_A+s_B$。

证明。每个下三角部分和上三角部分都转换为一个左三角实例，并分别计算。随后两次调用 LBruhatGen 算法，输入矩阵采用式 (20) 中的因子分解形式，并带有 $t=0$。

证明与命题 3.5 相同，唯一的区别是，代价中的 T稀疏 CRE 项替换为 $\mathsf{T}_{\mathsf{BruhatSumCRE}}$ 项，而子矩阵的行和列按 $\mathsf{T}_{\mathsf{SumExp}}$ 给出的代价计算。于是有

$$\begin{aligned}T(n,s)\le\;& 2T(n/2,s)+\mathsf{T}_{\mathsf{BSumCRE}}(n/2,s)+2\mathsf{T}_{\mathsf{SumExp}}(n/2,s,s)\\&+2\mathsf{T}_{\mathsf{MM}}(s,2s,n/2)+2\mathsf{T}_{\mathsf{TRSM}}(s,n/2)\\\le\;& 2T(n/2,s)+\left(\frac{9\cdot2^{\omega-3}-4}{2^{\omega-2}-1}C_\omega+C_{\mathsf{RF}}\right)ns^{\omega-1}\end{aligned}$$

调用一次 LBruhatGen 算法的代价。□

6 矩阵乘积 INSSS

两个以 SSS 形式给出的矩阵相乘时，会用到此前介绍过的两个技巧。第一个技巧是在压缩之前，先计算 $(s,2s)$-SSS 表示，方法与求和时相同。与求和不同的是，除了串接之外，还需要进行计算才能得到这一表示。第二个技巧是采用类似 LBruhatGen 算法中的类霍纳累加来加速这些计算。计算所有必要乘积时，将在两侧进行这种累加 $A_{i,k}B_{k,j}$ 其中 $A_{i,k}$ 位于对角线下方（或上方），而$B_{k,j}$位于对角线上方（或下方）。

SSSxSSS 算法详细说明了这些计算，使用 G_i和 H_i作为累加器。它改进了 [4, §3] 和 [10, 算法 7.2]中的算法：通过将每一步中的 4 个乘积保存在内存的 T_i和 S_i中，避免了这些乘积的计算。在数值环境中也可以避免这些计算。

定理 6.1。*SSSxSSS* 算法根据 s-SSS 形式给出的两个 $n\times n$ 矩阵，计算其乘积的 a 2s-SSS 生成元，其中

$$\mathsf{T}_{\mathsf{SSSxSSS}}(n,s)=\left(31+2^\omega\right)C_\omega ns^{\omega-1}. \tag{21}$$

证明。使用第 2 行和第 4 行处理 G_i，使用第 10 行和第 14 行处理 H_i，对 i进行归纳可得

$$G_i=\sum_{k=1}^{i-1}R_{i-1}^{(A)}\cdots R_{k+1}^{(A)}Q_k^{(A)}U_k^{(B)}W_{k+1}^{(B)}\cdots W_{i-1}^{(B)} \tag{22}$$

Algorithm 6.1 SSSxSSS

Input: For both $M\in\{A,B\}$, $P_i^{(M)},Q_i^{(M)},R_i^{(M)},U_i^{(M)},V_i^{(M)},W_i^{(M)}$, $D_i^{(M)}$ for appropriate $i\in[\![1,N]\!]$ an s-SSS generator for M
Output: A 2s-SSS generator for $C=AB$
▷ *All values not given as input are initialised to 0*
1: **for** $i\leftarrow1\ldots N$ **do**
2: $G_i\leftarrow Q_{i-1}^{(A)}U_{i-1}^{(B)}+T_{i-1}W_{i-1}^{(B)}$
3: $T_i\leftarrow R_i^{(A)}G_i$
4: $S_i\leftarrow P_i^{(A)}G_i$
5: $Q_i\leftarrow\begin{bmatrix}Q_i^{(B)}\\Q_i^{(A)}D_i^{(B)}+T_iV_i^{(B)}\end{bmatrix}$
6: $R_i\leftarrow\begin{bmatrix}R_i^{(B)}&0\\Q_i^{(A)}P_i^{(B)}&R_i^{(A)}\end{bmatrix}$
7: $U_i\leftarrow\begin{bmatrix}U_i^{(A)}&D_i^{(A)}U_i^{(B)}+S_iW_i^{(B)}\end{bmatrix}$
8: $W_i\leftarrow\begin{bmatrix}W_i^{(A)}&V_i^{(A)}U_i^{(B)}\\0&W_i^{(B)}\end{bmatrix}$
9: **for** $i\leftarrow N\ldots1$ **do**
10: $H_i\leftarrow V_{i+1}^{(A)}P_{i+1}^{(B)}+T_{i+1}R_{i+1}^{(B)}$
11: $T_i\leftarrow U_i^{(A)}H_i$
12: $D_i\leftarrow D_i^{(A)}D_i^{(B)}+S_iV_i^{(B)}+T_iQ_i^{(B)}$
13: $P_i^C\leftarrow\begin{bmatrix}D_i^{(A)}P_i^{(B)}+T_iR_i^{(B)}&P_i^{(A)}\end{bmatrix}$
14: $T_i\leftarrow W_i^{(A)}H_i$
15: $V_i\leftarrow\begin{bmatrix}V_i^{(A)}D_i^{(B)}+T_iQ_i^{(B)}\\V_i^{(B)}\end{bmatrix}$
16: **重新返回**SSS压缩$_i$, $_i$, $(_iP\ Q_i, R\ _iU\ W_i,\ \ _iD\in[\![1,\ N]\!]\ )$

$$H_i=\sum_{k=i+1}^{N}W_{i+1}^{(A)}\cdots W_{k-1}^{(A)}V_k^{(A)}P_k^{(B)}R_{k-1}^{(B)}\cdots R_{i+1}^{(B)} \tag{23}$$

将这些结果与 S_i的第 3 行、T_i的第 11 行以及最后的第 12 行结合起来，可得 $D_{(C)i}=\sum_{k=1}^{N}A_{i,\ k}B_{k,\ i}=C_{i,\ i}$。

当 $i<j$时，乘积 $A_{i,\ k}B_{k,\ j}$具有五种形状：$A\times$的下三角块、B的上三角块、$\times$的对角块、$\times$的上三角块、上三角的$\times$、以及上三角的$\times$。等式

$$U_i^{(C)}W_{i+1}^{(C)}\cdots W_{j-1}^{(C)}V_j^{(C)}=\sum_{k=1}^{N}A_{i,k}B_{k,j} \tag{24}$$

而当 $i>j$时，其对应情形可通过繁琐但直接的计算来验证。

其代价是在每个 N步骤中以及每次调用 SssCompression 算法时，对 $s\times s$矩阵进行 21 次乘法和 8 次加法。□

同样，定理 6.1 的结果仅适用于定义在同一网格上的矩阵，而且无论其拟可分离阶数如何，结果的存储大小始终相同。数值分析中的 HSS 生成元乘积 [22] 也有这一特性。Bruhat 格式可以避免这些问题，但据我们所知，目前还不存在用于计算两个 Bruhat 生成器乘积的次二次算法。第 5.2 节中用于求和的方法，为使用 LBruhatGen 算法实现线性或拟线性乘积算法开辟了道路。

A 实验

这里我们报告了在有限域上处理 SSS 和 Bruhat 生成器的算法的实现效果；该实现采用 fflas-ffpack 库 [11]，对应提交版本 33474b31aa。该库提供高效的稠密基础线性代数例程，例如矩阵乘法、TRSM 以及可揭示秩剖面矩阵的高斯消元。该库使用 GNU C++ 编译器 g++ 版本 9.3.0 编译，并链接至版本为 0.3.81 的 OpenBLAS 库。基准测试在单核上运行，硬件为英特尔 i5-i7300U@2.6GHz，操作系统为 Linux Mint-20。

在所有实验中，矩阵的维数固定为 $n = 3000$，定义在有限域 $\mathbb{Z}/131071\mathbb{Z}$ 上。我们绘制计算时间随拟可分阶数变化的曲线，测试三类实例：其上三角部分和下三角部分的秩分别为 1000、1500 和 1750。

每个点对应于 50 个具有相同参数的随机实例的运行时间均值。图 1 比较了从稠密矩阵生成矩阵的运行时间。图 2 使用相同的生成元，比较了与随机稠密$n \times 500$ 块向量相乘的运行时间。

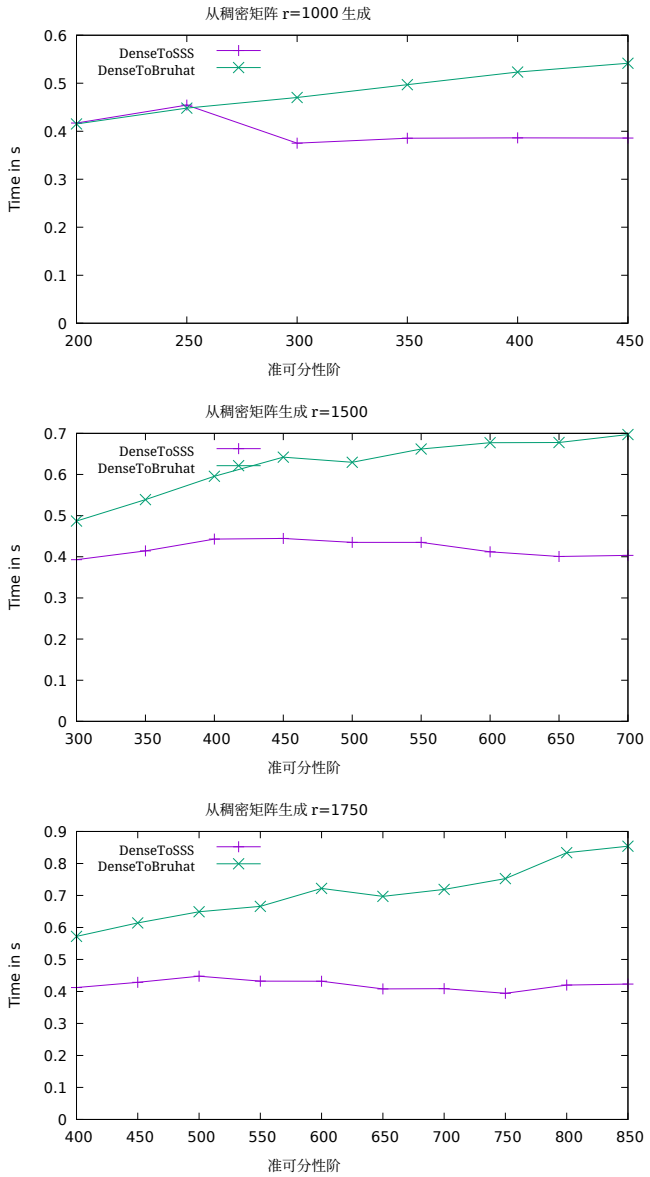

图 1：计算 SSS 和布鲁哈特生成元的实验计时，使用 $n = 3000$ 在 $\mathbb{Z}/131071\mathbb{Z}$

参考文献

[1] P. Boito、Y. Eidelman 和 L. Gemignani。2014。秩结构矩阵束的隐式 QR。*BIT Numerical Mathematics* 54，1（2014年3月），85–111。https: //doi.org/ 10 1007 10543 014 0478 0 。/s - - - P. Boito、Y. Eidelman 和 L. Gemignani。2017。[2] 结构化伴随矩阵束的实 QZ 算法。*Calcolo* 54，4（2017年12月），1305–1338。https://doi.org/10.1007/ s10092-017-0231-6[3] S. Chandrasekaran、P. Dewilde、M. Gu、T. Pals、X. Sun、A. J. van der Veen 和 D. White。2005年。序贯半可分表示的一些快速算法。*SIAM J. Matrix Anal. Appl.* 27，2（2005年），341–364。https://doi.org/10.1137/ S 0895479802405884 10 1137 0895479802405884 arXiv:https://doi.org/ 。/S [4] S. Chandrasekaran、P. Dewilde、M. Gu、T. Pals 和 A. J. van der Veen。2002年。序贯半可分线性方程组的快速稳定求解器。载于高性能计算 — *HiPC 2002*。Springer Berlin Heidelberg，545–554。https://doi.org/ 10 1007 3 540 36265 7_51 。/ - - - S. Chandrasekaran 和 M. Gu。[5] 。带状加半可分线性方程组的快速稳定算法。*SIAM J. Matrix AnalysisApplications* 25（2003年），373–384。https://doi.org/10.1137/S0895479899353373 2003

[6] S. Chandrasekaran、M. Gu 和 T. Pals。2006年。层次半可分表示的快速 ULV 分解求解器。*SIAM* 矩阵分析与应用期刊 28, 3 2006 603 622 10 1137 ( ), – . https://doi.org/ . /S 0895479803436652 [7] 2008 Steven Delvaux 和 Marc Van Barel。. 秩结构矩阵的 Givens-权表示。*SIAM* 矩阵分析与应用期刊 29, 4 (2008), 1147–1170。https://doi.org/10.1137/060654967 _电子版：https://doi.org/10.1137/060654967。[8] J.-G. Dumas、C. Pernet 和 Z. Sultan。2017年。秩剖面矩阵和广义布鲁哈分解的快速计算。符号计算期刊 83 2017 187 210 10 1016 2016 11 011 ( ), – . https://doi.org/ . /j.jsc. . . Y. Eidelman 和 I. Gohberg。[9] . 关于一类新的结构化矩阵。积分方程与算子理论 34 (1999), 293–324。https://doi.org/10.1007/ BF01300581[10] Y. Eidelman 和 I. Gohberg。2005年。拟可分离有限块矩阵的生成元。*Calcolo* 42 (12 2005), 187–214。https://doi.org/10.1007/s10092-005- 0102-4[11] FFLAS-FFPACK 团队。2021年。*FFLAS-FFPACK*：有限域线性代数子程序／软件包 (v2.5.0 版)。http://github.com/linbox-team/fflas-ffpack。[12] W. Hackbusch。1999年。一种基于 H-矩阵的稀疏矩阵算术。第一部分：H-矩阵简介。*Computing* 62 (1999), 89–108。https://doi.org/10. 1007/s006070050015[13] W. Hackbusch。2015年。层次矩阵：算法与分析。第49卷。施普林格。https://doi.org/ 10 1007 978 3 662 47324 5 . / - - - - W. Hackbusch、B. Khoromskij 和 S. A. Sauter。[14] . 关于 H^2-矩阵。载于应用数学讲义。Springer Berlin Heidelberg，9–29。https://doi.org/10. 1007/978-3-642-59709-1 2 2000 [15] C.-P. Jeannerod、C. Pernet 和 A. Storjohann。2013年。秩轮廓揭示高斯消元与 CUP 矩阵分解。符号计算期刊 56 (2013), 46–68。https://doi.org/10.1016/j.jsc.2013.04.004

[16] W. Lyons。2005年。应用于偏微分方程的快速算法。博士学位论文。美国加利福尼亚大学圣巴巴拉分校。http://scg.ece.ucsb.edu/publications/ theses/Lyons _2005_Thesis.pdf [17] W. Manthey 和 U. Helmke。2007年。线性系统的布鲁哈特标准形。《线性代数及其应用》425, 2–3 (2007), 261–282。https://doi.org/10.1016/j.laa.2007. 01.022[18] P.G. Martinsson。2011。用于计算矩阵层次半可分表示的快速随机化算法。《*SIAM* 矩阵分析与应用杂志》 32 10 2011 1251 1274 10 1137 100786617 ( ), – 。[19] 2016 https://doi.org/ 。 / Clément Pernet。 。拟可分矩阵的计算。载于符号与代数计算国际研讨会论文集（加拿大安大略省滑铁卢）。ACM 出版社，389–396。https://doi.org/10.1145/2930889. 2930915[20] C. Pernet、H. Signargout 和 G. Villard。2023年。秩亏高斯消元的首项常数。技术报告。hal:03976168。

[1] https://www.openblas.net

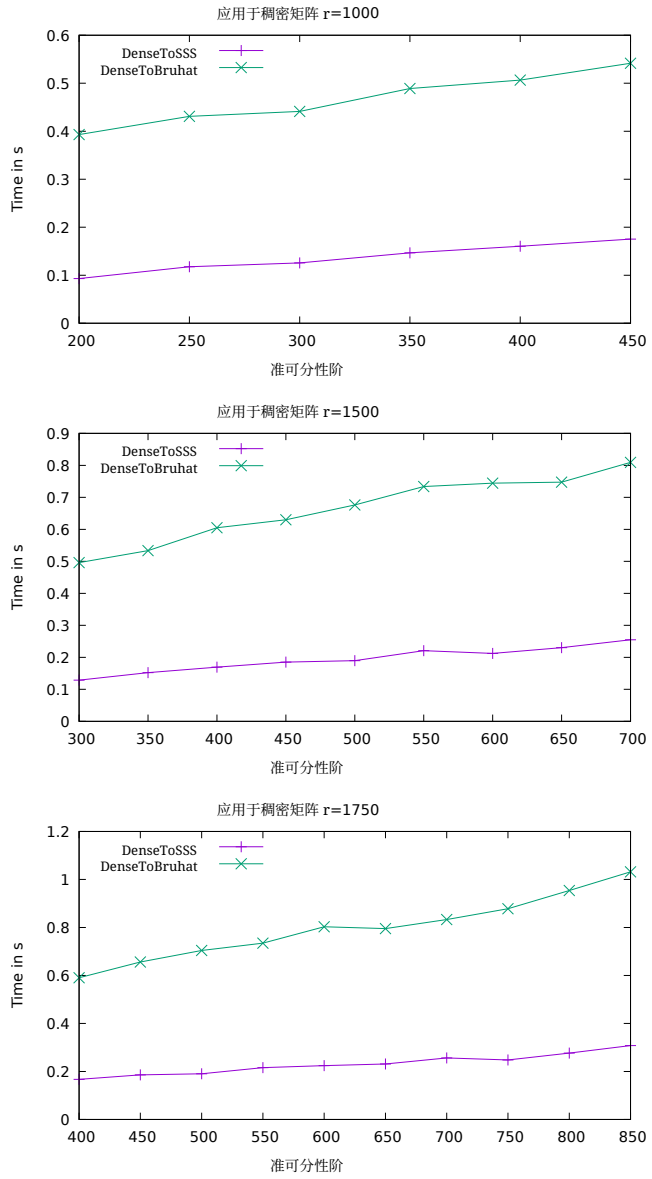

图 2：使用 $n = 3000$ 和 $v = 500$计算 SSS 与 Bruhat 的耗时，对象为稠密矩阵，范围为 $\mathbb{Z}/131071\mathbb{Z}$

[21] C. Pernet 和 A. Storjohann。2018年。拟可分矩阵的高效时间与空间生成元。《符号计算杂志》85（2018年），224 – 246。https://doi.org/10 1016 2017 07 010 。/j.jsc. . . Z. Sheng、P. Dewilde 和 S. Chandrasekaran。2007年。求解层次半可分系统的算法。第 176 卷。255–294。https://doi.org/10.1007/978- 3-7643-8137-0 5[23] H.P. Starr。1992年。一维积分与微分方程的数值求解。博士论文。耶鲁大学，美国。https://cpsc.yale.edu/sites/default/files/files/tr888.pdf UMI 订购编号 GAX92-35558。[24] A. Storjohann。2000年。矩阵标准形算法。博士论文。科学计算研究所，苏黎世联邦理工学院中心校区，苏黎世，瑞士。https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/145127/1/eth-24018-01.pdf

[25] R. Vandebril、M. Van Barel、G.H. Golub 和 N. Mastronardi。2005年。半可分矩阵参考文献目录*。*CALCOLO* 42 (2005), 249–270。https://doi.org/10.1007/s10092-005-0107-z[26] R. Vandebril、M. Van Barel、Gene H. Golub 和 N. Mastronardi。2008年。矩阵计算与半可分矩阵：线性系统。约翰·霍普金斯大学出版社。https://doi.org/10 1353 16537 . /book。J. Xia、S. Chandrasekaran、M. Gu 和 X.S. Li。2010年。层次半可分矩阵的快速算法。《数值线性代数及其应用》17, 6 (2010), 953–976。https://doi.org/10.1002/nla.691[27]