Exact computations with quasiseparable matrices

Clément Pernet
Grenoble INP, Univ. Grenoble Alpes
CNRS, LJK, UMR 5224
Grenoble, France

Hippolyte Signargout
ENS de Lyon, U. Lyon, CNRS, Inria,
UCBL, LIP UMR 5668 Lyon and LJK
UMR 5224 Grenoble, France

Gilles Villard
CNRS, U. Lyon, Inria, ENS de Lyon,
UCBL, LIP UMR 5668
Lyon, France

ABSTRACT

Quasi-separable matrices are a class of rank-structured matrices widely used in numerical linear algebra and of growing interest in computer algebra, with applications in e.g. the linearization of polynomial matrices. Various representation formats exist for these matrices that have rarely been compared.

We show how the most central formats SSS and HSS can be adapted to symbolic computation, where the exact rank replaces threshold based numerical ranks. We clarify their links and compare them with the Bruhat format. To this end, we state their space and time cost estimates based on fast matrix multiplication, and compare them, with their leading constants. The comparison is supported by software experiments.

We make further progresses for the Bruhat format, for which we give a generation algorithm, following a Crout elimination scheme, which specializes into fast algorithms for the construction from a sparse matrix or from the sum of Bruhat representations.

KEYWORDS

Quasiseparable matrix, SSS, HSS, Bruhat generator

1 INTRODUCTION

Quasiseparable matrices arise frequently in various problems of numerical analysis and are becoming increasingly important in computer algebra, e.g. by their application to handle linearizations of polynomial matrices [2]. Structured representations for these matrices and their generalisations have been widely studied but to our knowledge they have not been compared in detail with each other. In this paper we aim to adapt SSS [9] and HSS [5, 16], two of the most prominent formats of numerical analysis to exact computations and compare them theoretically and experimentally to the Bruhat format [21].These formats all have linear storage size in both the dimension and the structure parameter. We do not investigate the Givens weight representation [7] as it strongly relies on orthogonal transformations in $\mathbb{C}$, which transcription in the algebraic setting is more challenging. See [13, 25, 26] for an extensive bibliography on computing with quasiseparable matrices.

Definition 1.1. *An $n \times n$ matrix A is s-quasiseparable if for all $k \in [\![1, n]\!]$, $\operatorname{rank}(A_{1..k,k+1..n}) \leq s$ and $\operatorname{rank}(A_{k+1..n,1..k}) \leq s$.*

Complexity bound notation. We consider matrices over an abstract commutative field K, and count arithmetic operations in K. Our detailed comparison of formats aims in particular to determine the asymptotic multiplicative constants, an insightful measure on the algorithm's behaviour in pratice. In this regard, we will use the leading term in the complexities as the measure for our comparison: namely a function $\mathsf{T}_{\mathsf{XXX}}(n, s)$ such that the number of field operations for running Algorithm XXX with parameters n, s is $\mathsf{T}_{\mathsf{XXX}}(n, s) + o(\mathsf{T}_{\mathsf{XXX}}(n, s))$ asymptotically in n and s. We proceed similarly for the space cost bounds with the notation $\mathsf{S}_{\mathsf{XXX}}(n, s)$. We denote by ω a feasible exponent for square matrix multiplication, and C_ω the corresponding leading constant; namely, using above notation, $\mathsf{T}_{\mathsf{MM}}(n) = C_\omega n^\omega$, where MM corresponds to the operation $C = C + AB$ with $A, B, C \in \mathrm{K}^{n\times n}$. The straightforward generalization gives $\mathsf{T}_{\mathsf{MM}}(m, k, n) = C_\omega mnk \min(m, k, n)^{\omega-3}$ for the product of an $m \times k$ by a $k \times n$ matrix.

1.1 Rank revealing factorizations

Space efficient representations for quasiseparable matrices rely on rank revealing factorizations: a rank r matrix $A \in \mathrm{K}^{m\times n}$ is represented by two matrices $L \in \mathrm{K}^{m\times r} R \in \mathrm{K}^{r\times n}$ such that $A = LR$. In exact linear algebra, such factorizations are usually computed using Gaussian elimination, such as PLUQ, CUP, PLE, CRE decompositions [8, 15, 24], which we will generically denote by RF.

Cost estimates of the above factorization algorithms are either given as $O(mnr^{\omega-2})$ or with explicit leading constants $\mathsf{T}_{\mathsf{RF}}(m, n, r) = K_\omega n^\omega$ under genericity assumptions: $m = n = r$ and generic rank profile [8, 15]. We refer to [20] for an analysis in the non-generic case of the leading constants in the cost of the two main variants of divide and conquer Gaussian elimination algorithms. We may therefore assume that $\mathsf{T}_{\mathsf{RF}}(m, n, r) = C_{\mathsf{RF}} mnr^{\omega-2}$ for a constant C_{RF}, for $\omega \geq 1 + \log_2 3$, which is the case for all pratical matrix multiplication algorithm. Note that for $\omega = 3$, these costs are both equal to $2mnr$. Unfortunately, the non-predictable rank distribution among the blocks being processed leads to an over-estimation of some intermediate costs which forbids tighter constants (i.e. interpolating the known one $K_3 = 2/3$ in the generic case). The algorithms presented here still carry on for smaller values of ω, but we chose to skip the more complex derivation of estimates on their leading constants for the sake of clarity.

Our algorithms for SSS and HSS can use any rank revealing factorization. On the other hand, the Bruhat format requires one revealing the additional information of the rank profile matrix, e.g. the CRE decompositions used here (See [8]).

Theorem 1.2 ([8, 17]). *Any rank r matrix $A \in \mathrm{K}^{m\times n}$ has a CRE decomposition $A = CRE$ where $C \in \mathrm{K}^{m\times r}$ and $E \in \mathrm{K}^{r\times n}$ are in column and row echelon form, and $R \in \mathrm{K}^{r\times r}$ is a permutation matrix.*

The costs we give in relation to Bruhat generator therefore rely on constants C_{RF} from factorizations allowing to produce a CRE decomposition, like the ones in [20].

1.2 Contributions

In Section 2 we define the SSS, HSS and Bruhat formats. We then adapt algorithms operating with HSS and SSS generators from the literature to the exact context. The HSS generation algorithm is given in a new iterative version and the SSS product algorithm has an improved cost. We focus for SSS on basic bricks on which other

Table 1: Summary of operation and storage costs

	ω			$\omega = 3$		
	SSS	HSS	Bruhat	SSS	HSS	Bruhat
Storage	$7ns$	$18ns$	$4ns$	$7ns$	$18ns$	$4ns$
Gen. from Dense	$2C_{\mathsf{RF}}n^2s^{\omega-2}$	$2^{\omega}C_{\mathsf{RF}}n^2s^{\omega-2}$	$C_{\mathsf{RF}}n^2s^{\omega-2}$	$4n^2s$	$16n^2s$	$2n^2s$
$\times$ Dense block vector$(n \times v)$	$7C_{\omega}nsv^{\omega-2}$	$18C_{\omega}nsv^{\omega-2}$	$8C_{\omega}nsv^{\omega-2}$	$14nsv$	$36nsv$	$16nsv$
Addition	$(10+2^{\omega})C_{\omega}ns^{\omega-1}$		$\left(\frac{9\cdot 2^{\omega-2}-8}{2^{\omega-2}-1}C_{\omega}+2C_{\mathsf{RF}}\right)ns^{\omega-1}\log n/s$	$36ns^2$		$24ns^2\log n/s$
Product	$(31+2^{\omega})C_{\omega}ns^{\omega-1}$			$78ns^2$		

operations can be built. This opens the door to adaptation of fast algorithms for inversion and system solving [3, 4, 10] and format modeling operations such as merging, splitting and model reduction [4]. In Section 3.3 we give a generic Bruhat generation algorithm from which we derive new fast algorithms for the generation from a sparse matrix and from a sum of matrices in Bruhat form.

Table 1 displays the best cost estimates for differents operations on an $n \times n$ s-quasi-separable matrix in the three formats presented in the paper. The best and optimal storage size is reached by the Bruhat format which also has the fastest generator computation algorithm. However, this is not reflected in the following operation costs as applying a quasiseparable matrix to a dense matrix is least expensive with an SSS generator and addition and product of $n \times n$ matrices given in Bruhat form is super-linear in n. We notice in Proposition 2.5 that HSS is twice as expensive as SSS and gives no advantage in our context. We thus stop the comparison at the generator computation. We still give in Table 1 the cost of quasiseparable $\times$ dense product which is proportional to the generator size [16]. We complete this analysis with experiments showing that despite slightly worse asymptotic cost estimates, SSS performs better than Bruhat in practice for the construction in Section 3.5 and the product by a dense block vector in Section 4.3.

2 PRESENTATION OF THE FORMATS

2.1 SSS generators

Introduced in [9], SSS generators were later improved independently in [10] and [4] using block-versions, which we present here. In particular, the space was improved from $O(ns^2)$ to $O(ns)$.

An s-quasiseparable matrix is sliced following a grid of $s \times s$ blocks. Blocks on, over and under the diagonal are treated separately. On one side of the diagonal, each block is defined by a product depending on its row (left-most block of the product), its column (right-most block), and its distance to the diagonal (number of blocks in the product).

Definition 2.1. *Let* $A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,N} \\ \vdots & & \vdots \\ A_{N,1} & \cdots & A_{N,N} \end{bmatrix} \in \mathrm{K}^{n\times n}$ *with* $t\times t$ *blocks* $A_{i,j}$ *for* $i, j < N$ *and* $N = \lceil n/t \rceil$. *A is given in sequentially semi-separable format of order t (t-SSS) if it is given by the* $t \times t$ *matrices* $(P_i, V_i)_{i\in[\![2,N]\!]}$, $(Q_i, U_i)_{i\in[\![1,N-1]\!]}$, $(R_i, W_i)_{i\in[\![2,N-1]\!]}$, $(D_i)_{i\in[\![1,N]\!]}$ *s.t.*

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

Proposition 2.2. *Any* $n\times n$ *s-quasiseparable matrix has an s-SSS representation. It uses* $\mathrm{S}_{\mathsf{SSS}}(n,s) = 7ns$ *field elements.*

Proof. Direct consequence of Proposition 3.1. ☐

2.2 HSS generators

The HSS format was first introduced in [6], although the idea originated with the *uniform* $\mathcal{H}$*-matrices* of [12] and in more details with the $\mathcal{H}^2$*-matrices* of [14], with algorithms relying on [23]. The $\mathcal{H}^2$ format is slightly different from HSS, more details in [13].

The format is close to SSS (see Proposition 2.4) as the way of defining blocks is similar. Yet, the slicing grid is built recursively and the definition of blocks product depends on the path to follow in the recursion tree. Also, both sides of the diagonal are treated jointly and the format is therefore less compact, which as will be shown makes HSS less efficient.

The structure is complex and notations differ in the literature. We made the following choices: we avoid the recursive tree definition inherited from the Fast Multipole Method [6] and thus only consider constant-depth recursive block divisions. We made this choice to focus on linear algebra and quasiseparable matrices with no prerequisites (no notion of where the rank is). For the same reason we focus on uniform subdivisions. Most literature on HSS uses non-uniform grids in order to adapt to matrices with a structure within the quasiseparable rank structure [6]. Despite being more general, this adds confusion which is not needed in our case.

We use a notation similar to [27] with transition matrices.

Definition 2.3. *Let* $A \in \mathrm{K}^{n\times n}$ *and the uniform block divisions*

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

A is given in hierarchically semi-separable format of order t (t-HSS) if it is given by the $t\times t$ *matrices* $(U_{K;i}, V_{K;i}, D_i)_{i\in[\![1,N]\!]}$, $(R_{k;i}, W_{k;i})_{k\in[\![2,K]\!],\, i\in[\![1,2^k]\!]}$ *and* $(B_{k;i})_{k\in[\![1,K]\!],\, i\in[\![1,2^k]\!]}$ *with* $N = \lceil n/t \rceil$ *and* $K \geq \log N$ *such that for* $i \in [\![1,N]\!]$, $A_{K;i,i} = D_i$ *and if we define recursively for k from* $K-1$ *to* 1 *and* $i \in [\![1,2^k]\!]$, $U_{k;i} = \begin{bmatrix} U_{k+1;2i-1}R_{k+1;2i-1} \\ U_{k+1;2i}R_{k+1;2i} \end{bmatrix}$ *and* $V_{k;i} = \begin{bmatrix} W_{k+1;2i-1}V_{k+1;2i-1} & W_{k+1;2i}V_{k+1;2i} \end{bmatrix}$ *then*

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

The HSS generator can be seen as a recursive SSS generator with two differences : the use of the B matrices, and the distribution of the translation matrices. The similarity is made clear in Proposition 2.4.

Proposition 2.4. *Let $U_{K;i}, V_{K;i}, D_i, R_{k;i}, W_{k;i}, B_{k;i}$ for appropriate $k \le K, i \le 2^k$ a t-HSS generator for A. Let $I, J \in [\![1, 2^K]\!]$ and k the highest level of recursion for which $A_{K;I,J}$ is not included in a diagonal block. For $i_1 = \lfloor I/2^{K-k-1} \rfloor$, $i_0 = \lfloor I/2^{K-k} \rfloor$ and $j_1 = \lfloor J/2^{K-k-1} \rfloor$ we have*

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

Proof. By induction on Equation (3). ◻

Proposition 2.5. *Any $n \times n$ s-quasiseparable matrix has a $2s$-HSS representation. This is the optimal block parameter and the representation uses $\mathsf{S}_{\mathsf{HSS}}(n, s) = 18ns$ field elements.*

Proof. Consequence of Proposition 3.2. For optimality let A be s-quasiseparable given in t-HSS form. We use Proposition 2.4:

$$\begin{bmatrix} A_{K;3\ldots4,1\ldots2} & A_{K;3\ldots4,5\ldots6} \end{bmatrix} = \begin{bmatrix} U_{K;3} R_{K;3} \\ U_{K;4} R_{K;4} \end{bmatrix} H \tag{5}$$

where $H \in \mathrm{K}^{t \times 4t}$. The quasi-separability of A bounds the rank of the left part of Eq. (5) by $2s$ while the one of the right side is bounded by t. When the first bound is tight we get $t \ge 2s$. ◻

2.3 Bruhat generators

The Bruhat generator was first defined in [19, 21]. Contrarily to SSS and HSS, it does not use on a pre-defined grid but relies on the rank profile information contained in the rank profile matrix [8] of the lower and upper triangular parts of the quasiseparable matrix.

Recall from [21] that a matrix is t-overlapping if any subset of $t+1$ of its non-zero columns (resp. rows) contains at least one whose leading non-zero element is below (resp. before) the trailing non-zero element of another. We call J_n the anti-identity matrix of dimension n and define the *Left* operator $\diagup\!\!\!\!\sqsubset : \mathrm{K}^{n\times n} \to \mathrm{K}^{n \times n}$ s.t.

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

Definition 2.6. *An $n \times n$ matrix A is represented in t-Bruhat format if it is given by a diagonal matrix $D \in \mathrm{K}^{n\times n}$ and 6 matrices $C^{(L)}, R^{(L)}, E^{(L)}, C^{(U)}, R^{(U)}, E^{(U)}$ where $C^{(L)} \in \mathrm{K}^{n\times u}$ and $C^{(U)} \in \mathrm{K}^{n\times v}$ are in column echelon form and t-overlapping, $E^{(L)} \in \mathrm{K}^{u\times n}$ and $E^{(U)} \in \mathrm{K}^{v \times n}$ are in column echelon form and t-overlapping and $R^{(L)} \in \mathrm{K}^{u\times u}, R^{(U)} \in \mathrm{K}^{v\times v}$ are permutation matrices and satisfy*

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

Proposition 2.7. *Any $n \times n$ s-quasiseparable matrix has an s-Bruhat representation. It uses $\mathsf{S}_{\mathsf{Bruhat}}(n, s) = 4ns$ field elements which is optimal.*

Proof. By [21, Theorem 20]. As $2ns$ coefficients are necessary to represent all rank s triangular matrices, $4ns$ is optimal. ◻

3 CONSTRUCTION OF THE GENERATORS

3.1 SSS generator from a dense matrix

We recall in Algorithm DenseToSSS the construction of an SSS generetor from a dense s-quasiseparable matrix $A \in \mathrm{K}^{n\times n}$. It is adapted from [4, §6.1] and [10, Alg. 6.5] where the SVD based numerical rank revealing factorizations are replaced by RF.

The blocks D_i are directly extracted from the dense matrix in Line 3. Each block-triangular part is then compressed independently. Each step eliminates a chunk made of a block-row of A and a remainder from the previous step. The result is three blocks of the generator and a remainder to be eliminated at the subsequent step.

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

Proposition 3.1. *Algorithm DenseToSSS computes a t-SSS generator for an s-quasiseparable matrix ($s \le t$) in $\mathsf{T}_{\mathsf{DenseToSSS}}(n, t) = 2C_{\mathsf{RF}} n^2 s^{\omega-2}$ field operations.*

Proof. For $k \in [\![1, N-1]\!]$, the dimensions of the output of Lines 4 and 5 are sufficient since the input of the factorisation is a concatenation of a block of A with a rank-revealing factor of another block of A on the same side of the diagonal, and is hence of rank at most s.

Let $i, j \in [\![1, N]\!]$. If $i = j$ Line 3 for $k = i$ gives $D_i = A_{i,i}$. If $i < j$, Line 4 gives

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

which combines to $U_i W_{i+1} \ldots W_{j-1} V_j = A_{i,j}$. The same way, if $i > j$ then $P_i R_{i-1} \ldots R_{j+1} Q_j = A_{i,j}$.

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 HSS generator from a dense matrix

The first construction algorithm for a general quasiseparable matrix is presented in [6]. We present in Algorithm DenseToHSS an iterative version of the faster and simpler algorithm of [27].

Each step of the loop on k passes block-row-wise and block-column-wise on the matrix inherited from the previous step, factorising block rows and block columns two by two. At each step each block is hence factorised twice, producing transition matrices R and W, the remainder being either passed to the following step or finally stored as a B matrix.

Algorithm 3.2 DenseToHSS

Input: A an $n \times n$ quasiseparable matrix of order s

Output: $U_{K;i}, V_{K;i}, D_i, R_{k;i}, W_{k;i}, B_{k;i}$ for appropriate $k \leq K, i \leq 2^k$ a t-HSS representation of A with $t \geq 2s$

1: $H \leftarrow A$ ▷ *Use the block division of Eq.* (2) *with* $k = K$
2: **for** $i = 1 \dots 2^K$ **do**
3: $\quad D_i \leftarrow A_{K;i,i}$
4: **for** $k = K \dots 1$ **do**
5: $\quad$ **for** $i = 1 \dots 2^k$ **do** ▷ *All operations are in this loop*
▷ *$R_{K+1;2i}$ (resp. $W_{K+1;2i}$) has row (resp. column) dimension 0*
6: $\quad\quad \left(\begin{bmatrix} R_{k+1;2i-1} \\ R_{k+1;2i} \end{bmatrix}, \begin{bmatrix} H''_{k;i,1\dots i-1} & H'_{k;i,i+1\dots 2^k} \end{bmatrix}\right) \leftarrow \mathsf{RF}\left(\begin{bmatrix} H'_{k;i,1\dots i-1} & H_{k;i,i+1\dots 2^k} \end{bmatrix}\right)$
7: $\quad\quad \left(\begin{bmatrix} H''_{k;1\dots i-1,i} \\ H'_{k;i+1\dots 2^k,i} \end{bmatrix}, \begin{bmatrix} W_{k+1;2i-1} & W_{k+1;2i} \end{bmatrix}\right) \leftarrow \mathsf{RF}\left(\begin{bmatrix} H'_{k;1\dots i-1,i} \\ H_{k;i+1\dots 2^k,i} \end{bmatrix}\right)$
8: $\quad$ **for** $i = 1 \dots 2^{k-1}$ **do** ▷ *Only renaming from here*
9: $\quad\quad B_{k;2i-1} \leftarrow H''_{k;2i-1,2i}$
10: $\quad\quad B_{k;2i} \leftarrow H''_{k;2i,2i-1}$
11: $\quad\quad$ **for** $j = 1 \dots 2^{k-1}, j \neq i$ **do**
12: $\quad\quad\quad H_{k-1;i,j} \leftarrow \begin{bmatrix} H''_{k;2i-1,2j-1} & H''_{k;2i-1,2j} \\ H''_{k;2i,2j-1} & H''_{k;2i,2j} \end{bmatrix}$
13: $\quad\quad\quad H_{k-1:j,i} \leftarrow \begin{bmatrix} H''_{k:2j-1,2i-1} & H''_{k:2j-1,2i} \\ H''_{k:2j,2i-1} & H''_{k:2j,2i} \end{bmatrix}$
14: **for** $i = 1 \dots 2^K$ **do**
15: $\quad U_{K;i} \leftarrow R_{K+1;2i-1}$
16: $\quad V_{K;i} \leftarrow W_{K+1;2i-1}$

Proposition 3.2. *Algorithm DenseToHSS computes a t-HSS generator for an s-quasiseparable matrix if $2s \leq t$ in $C_{\mathsf{RF}} n^2 t^{\omega-2}$ field operations. For $t = 2s$, this is $\mathsf{T}_{\mathsf{DenseToHSS}}(n, s) = 2^\omega C_{\mathsf{RF}} n^2 s^{\omega-2}$.*

Proof. Let $k \in [\![1, K]\!]$, $i \neq j \in [\![1, 2^k]\!]$. The dimensions of the output in Lines 6 and 7 is sufficient since the matrices being factorized are each time a concatenation of two blocks of rank at most s and are hence of rank at most $2s \leq t$. If $|i - j| = 1$, the instructions give

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

Otherwise,

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

Let now $I, J \in [\![1, N]\!]$. If $I = J$, Line 3 gives $D_I = A_{K;I,J}$. Otherwise, let k be the highest level of recursion for which $A_{K;I,J}$ is not included in a diagonal block. From Line 1, $A_{K;I,J} = H_{K;I,J}$. Equation (12) can be used $K - k$ times, together with Line 12 to get

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

where $i_2 = \lfloor I/2^{K-k-2} \rfloor$, $i_1 = \lfloor I/2^{K-k-1} \rfloor$, $j_1 = \lfloor J/2^{K-k-1} \rfloor$ and $j_2 = \lfloor J/2^{K-k-2} \rfloor$. $R_{K+1;2I-1}$, $W_{K+1;2J-1}$ and $H''_{k+1;i_1,j_1}$ can be replaced in Eq. (13) using Lines 15 and 16 and Eq. (11) (from the definition of k we have $|i_1 - j_1| = 1$) in order to get Eq. (4); this concludes the proof of correctness.

Line 6 at $k < K$ and i peforms a rank revealing decompositions on an input formed by the $2t \times (i-1)t$ block $H'_{k;i,1\dots i-1}$ and the $2t \times 2t(2^k - i)$ block $H_{k;i,i+1\dots 2^k}$ at cost $\mathsf{T}_{\mathsf{RF}}(t(2^{k+1} - i), 2t, t)$. The cost is equal for Line 7. The overall cost is then $\sum_{k=1}^{\log_2 \frac{n}{t}} \sum_{i=1}^{2^k} 4C_{\mathsf{RF}}(2^{k+1} - i)t^\omega \leq 4C_{\mathsf{RF}} n^2 t^{\omega-2} \leq 2^\omega C_{\mathsf{RF}} n^2 s^{\omega-2}$. ◻

Because the blocks of each side of the diagonal are defined by the same matrices, Algorithm DenseToHSS and any HSS construction algorithm applies rank revealing factorisations on blocks with rank bounded by $2s$ for s-quasiseparable matrices instead of s in Algorithm DenseToSSS. The optimal HSS block size of s-quasiseparable matrices is thus $2s$, which makes HSS less efficient in terms of storage and operation cost.

As the costs are higher and HSS has the same drawbacks as SSS, namely needing a fixed slicing grid and a previously computed quasiseparability order, we do not detail more algorithms for HSS. For information in the numerical context we mainly refer to [16, 22]. Note that faster construction algorithms exist, probabilistic in [18] and with constraints on the input in [6].

3.3 Bruhat generator from a dense matrix

The construction of a `Bruhat` generator from a dense matrix is achieved by [21, Alg. 12] run twice, once for each of the upper and lower triangular parts of the input matrix, and the diagonal matrix D is directly extracted from the dense matrix.

We give in Algorithm LBruhatGen an updated version of [21, Alg. 12], where Schur complement computations are delayed until they are needed. This allows for faster computations when the input is not given as a dense matrix and will be used for computing the sum of two matrices in `Bruhat` form and generators from a sparse matrix.

Algorithm LBruhatGen can be given any input format, provided we have a way to compute for any submatrix B of the input matrix

(1) $\mathsf{CRE}(B, G, H)$ a CRE decomposition of $B - GH^\mathsf{T}$;
(2) for $\mathcal{R}$ a set of indices, $B_{\mathcal{R},*}$ and $B_{*,\mathcal{R}}$ the rows and columns of B with indices in $\mathcal{R}$.

We use the notation TRSM for *TRiangular Solve Matrix*: $\mathsf{TRSM}(L, A)$ outputs $L^{-1}A$ for L triangular.

Proposition 3.3. *An s-`Bruhat` generator can be computed from an $n \times n$ dense s-quasiseparable matrix in $\mathsf{T}_{\mathsf{DenseToB}}(n, s) = C_{\mathsf{RF}} n^2 s^{\omega-2}$.*

Proof. Algorithm LBruhatGen is adapted from [21, Alg. 12]; we therefore refer to the proof of [21, Theorem 24] for its correctness. Apart from the order in which they are made, the operations are the same in both algorithms when the input is dense and the cost is hence the same. Computing a `Bruhat` generator from a dense matrix is two applications of Algorithm LBruhatGen. The cost satisfies:

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

3.4 Bruhat generator from a sparse matrix

In applications, matrices are often presented in a sparse structure. In order to detect and/or harness their quasiseparable structure, it

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 $A - \text{◺}\left(GH^\mathsf{T}\right)$

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 $A^{(11)} \in \mathrm{K}^{\frac{m}{2}\times\frac{m}{2}}$

$C_0, R_0, E_0 \leftarrow \mathtt{CRE}(A^{(11)}, G^{(1)}, H^{(1)})$

$\mathcal{R} \leftarrow \mathtt{RRP}(C_0); \mathcal{C} \leftarrow \mathtt{CRP}(E_0); r_0 \leftarrow \#\mathcal{R}$

$\begin{bmatrix} U & V \end{bmatrix} \leftarrow E_0 Q_\mathcal{C}$ where $U \in \mathrm{K}^{r_0\times r_0}$ is upper triangular.

$\begin{bmatrix} L \\ M \end{bmatrix} \leftarrow P_\mathcal{R} C_0$ where $L \in \mathrm{K}^{r_0\times r_0}$ is lower triangular.

$X \leftarrow A^{(12)}_{\mathcal{R},*} - G^{(1)}_{\mathcal{R},*} {H^{(2)}}^\mathsf{T}$

$B^{(12)} \leftarrow \text{◺}(\mathtt{TRSM}\,(L, X))_{\mathcal{R},*}$ ▷ $A^{(12)}_{\mathcal{R},*} = \text{◺}\left(LB^{(12)} + G^{(1)}_{\mathcal{R},*}{H^{(2)}}^\mathsf{T}\right)_{\mathcal{R},*}$

$Y \leftarrow A^{(21)}_{*,\mathcal{C}} - G^{(2)} {H^{(1)}_{\mathcal{C},*}}^\mathsf{T}$

${B^{(21)}}^\mathsf{T} \leftarrow \text{◺}\left(\mathtt{TRSM}(U^\mathsf{T}, Y^\mathsf{T})\right)_{*,\mathcal{C}}$

▷ $A^{(21)}_{*,\mathcal{C}} = \text{◺}\left(B^{(21)}U + G^{(2)}{H^{(1)}}^\mathsf{T}_{*,\mathcal{C}}\right)_{*,\mathcal{C}}$

$C_1, R_1, E_1 \leftarrow$ $\mathtt{LBruhatGen}\left(A^{(21)}_{*,\overline{\mathcal{C}}}\mathrm{I}_{\overline{\mathcal{C}},*}, \left[\, G^{(2)} \;\; B^{(21)} \,\right], \mathrm{I}_{*,\overline{\mathcal{C}}}\left[\, H^{(1)}_{\overline{\mathcal{C}},*} \;\; V^\mathsf{T} \,\right]\right)$

$C_2, R_2, E_2 \leftarrow$ $\mathtt{LBruhatGen}\left(\mathrm{I}_{*,\overline{\mathcal{R}}}A^{(12)}_{\overline{\mathcal{R}},*}, \mathrm{I}_{*,\overline{\mathcal{R}}}\left[\, G^{(1)}_{\overline{\mathcal{R}},*} \;\; M \,\right], \left[\, H^{(2)} \;\; {B^{(12)}}^\mathsf{T} \,\right]\right)$

$P_{01} \leftarrow$ the permutation which sorts the rows of E_0 and E_1 by increasing column of pivot

$P_{02} \leftarrow$ the permutation which sorts the columns of C_0 and C_2 by increasing row of pivot

$C \leftarrow \begin{bmatrix} C_0 & C_2 & \\ B^{(21)}{R_0}^\mathsf{T} & & C_1 \end{bmatrix} \begin{bmatrix} P_{02} & \\ & I \end{bmatrix}$

$R \leftarrow \begin{bmatrix} {P_{02}}^\mathsf{T} & \\ & I \end{bmatrix} \begin{bmatrix} R_0 & & \\ & & R_2 \\ & R_1 & \end{bmatrix} \begin{bmatrix} {P_{01}}^\mathsf{T} & \\ & I \end{bmatrix}$

$E \leftarrow \begin{bmatrix} P_{01} & \\ & I \end{bmatrix} \begin{bmatrix} E_0 & {R_0}^\mathsf{T}B^{(12)} \\ E_1 & \\ & E_2 \end{bmatrix}$

return C, R, E

is crucial to exploit the sparsity in the construction of the quasiseparable generators.

For the construction of a Bruhat generator, the generic algorithm Algorithm LBruhatGen can be applied on a sparse matrix, provided two operations are specialized:

(1) the extraction of a subset of $\leq s$ rows or columns into a dense format, which is straightforward for a sparse matrix;
(2) the computation of a CRE decompoistion, which is specialized in Algorithm SparseCRE which in turn uses Algorithm SparseRankProfiles

Lemma 3.4. *Algorithm SparseRankProfiles is correct with probablity at least* $1 - 2r/|S|$ *and runs in* $\mathsf{T}_{\mathsf{SparseRP}}(n, r) = 2(C_\omega + C_{\mathsf{RF}})nr^{\omega-1} + 2r|A|$ *with* $r = t + s$.

Proof. Applying the Toeplitz pre-conditionners in Lines 3 and 4 costs $\frac{nt}{r}\tilde{O}(r)$ which is domintated by $nr^{\omega-1}$. □

Algorithm 3.4 SparseCRE

Input: $A \in \mathrm{K}^{m\times m}$ a rank $\leq s$ sparse matrix

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

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

$\mathcal{R}, \mathcal{C} \leftarrow$ SparseRankProfiles(A, G, H)

$P = \begin{bmatrix} \mathrm{I}_{\mathcal{R},*} \\ \mathrm{I}_{\overline{\mathcal{R}},*} \end{bmatrix}; Q = \begin{bmatrix} \mathrm{I}_{*,\mathcal{C}} & \mathrm{I}_{*,\overline{\mathcal{C}}} \end{bmatrix}$

▷ *With* $\bar{A}^{(11)} \in \mathrm{K}^{|\mathcal{R}|\times|\mathcal{R}|}$ *write* $P\left(A - GH^\mathsf{T}\right)Q = \begin{bmatrix} \bar{A}^{(11)} & \bar{A}^{(12)} \\ \bar{A}^{(21)} & \bar{A}^{(22)} \end{bmatrix} - \begin{bmatrix} \bar{G}^{(1)} \\ \bar{G}^{(2)} \end{bmatrix} \begin{bmatrix} \bar{H}^{(1)} \\ \bar{H}^{(2)} \end{bmatrix}^\mathsf{T}$

$M^{(11)} \leftarrow \bar{A}^{(11)} - \bar{G}^{(1)}(\bar{H}^{(1)})^\mathsf{T}$

$M^{(12)} \leftarrow \bar{A}^{(12)} - \bar{G}^{(1)}(\bar{H}^{(2)})^\mathsf{T}$

$M^{(21)} \leftarrow \bar{A}^{(21)} - \bar{G}^{(2)}(\bar{H}^{(1)})^\mathsf{T}$

$(L, R, U) \leftarrow \mathtt{DenseCRE}\left(M^{(11)}\right)$

$C \leftarrow \mathtt{TRSM}(L, \bar{M}^{(12)})$ ▷ $C = L^{-1}(\bar{A}^{(12)} - G^{(1)}H^{(2)})$

$D \leftarrow \mathtt{TRSM}(\bar{A}^{(21)}, U^\mathsf{T})$ ▷ $D = (\bar{A}^{(21)} - G^{(2)}H^{(1)})U^{-1}$

$E \leftarrow \begin{bmatrix} U & R^\mathsf{T}C \end{bmatrix} Q^\mathsf{T}$

$C \leftarrow P^\mathsf{T} \begin{bmatrix} L \\ DR^\mathsf{T} \end{bmatrix}$

return (C, R, E)

Algorithm 3.5 SparseRankProfiles

Input: $A \in \mathrm{K}^{n\times n}$ a sparse matrix of rank $\leq s$.

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

Output: $\mathcal{R}_A, \mathcal{C}_A$ the row and column rank profiles of $A - GH^\mathsf{T}$

1: $T^{(1)} \leftarrow$ a unif. random $n \times (s + t)$ Toeplitz matrix from $S \subseteq \mathrm{K}$
2: $T^{(2)} \leftarrow$ a unif. random $(s + t) \times n$ Toeplitz matrix from $S \subseteq \mathrm{K}$
3: $K \leftarrow H^\mathsf{T}T^{(1)}$
4: $L \leftarrow T^{(2)}G$
5: $P \leftarrow AT^{(1)} - GK$
6: $Q \leftarrow T^{(2)}A - LH^\mathsf{T}$
7: **return** $\mathtt{RowRankProfile}(P), \mathtt{ColRankProfile}(Q)$

Proposition 3.5. *Algorithm SparseCRE computes a CRE decomposition of* $A - GH^\mathsf{T}$ *with probablity at least* $1 - 2r/|S|$ *in* $\mathsf{T}_{\mathsf{SparseCRE}}(n, r) = \left(\frac{2^\omega - 3}{2^{\omega-2} - 1}C_\omega + 2C_{\mathsf{RF}}\right)nr^{\omega-1} + 2r|A|$ *field operations for* $s + t \leq r$.

Proof. Let ρ be the rank of $A - GH^\mathsf{T}$.

$$\begin{aligned} \mathsf{T}_{\mathsf{SparseCRE}}(n, r) &= 2\mathsf{T}_{\mathsf{MM}}(n, r, t) + \mathsf{T}_{\mathsf{CRE}}(\rho, \rho, \rho) + 2\mathsf{T}_{\mathsf{TRSM}}(n - \rho, \rho) \\ &\quad + \mathsf{T}_{\mathsf{SparseRP}}(n, r) \\ &\leq nr^{\omega-1}\left(4C_\omega + \frac{2C_\omega}{2^{\omega-1} - 2} + 2C_{\mathsf{RF}}\right) + 2r|A|. \end{aligned}$$

□

Proposition 3.6. *Algorithm LBruhatGen computes a Left-*Bruhat *generator from an sparse s-quasiseparable matrix* $A \in \mathrm{K}^{n\times n}$ *in*

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

field operations with probability at least $1 - 2n/|S|$.

PROOF. First, remark that the G and H matrices correspond to delayed Schur complement updates for pivots processed in the previous calls. Hence, in every call to Algorithm LBruhatGen, these pivots are located to the left, to the top or in the left-top corner of the work matrix. The quasiseparable condition imposes that there are $t \leq 2s$ of them. Moreover, in the call to Algorithm SparseCRE, the ranks verify $r_A + r_B + t \leq s$. Hence we can bound t and write the cost of Algorithm LBruhatGen only in terms of n the dimension of the matrix, s the initial quasiseparability order, and $|\cdot|$ the amount of non-zero coefficients of the submatrices we consider.

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

The failure probability is obtained by a union bound on the failure probability of each of the n/s calls to Algorithm SparseCRE. □

We are not aware of any similar algorithm for computing an SSS or HSS generator using the sparsity of the input matrix and can hence only compare our result to the quadratic generation from a dense matrix.

3.5 Experimental comparison

To complement the asymptotic cost analysis, we present in Fig. 1 experiments comparing the computation time for the construction of SSS and Bruhat generators. The timings for Bruhat are sublinear in s, as could be expected from Proposition 3.3 but also slightly depends on r which comes from neglected costs arising e.g. from the numerous permutations. The SSS cost is constant on our values for reasons we are unable to explain yet. It is almost always lower than the Bruhat cost. Yet remember that Algorithm DenseToSSS takes the quasiseparable order as input, so it has to be computed first (for example with Algorithm LBruhatGen).

4 APPLICATION TO A BLOCK VECTOR

We study here the application of an s-quasi-separable matrix $A \in \mathrm{K}^{n\times n}$ given by its generators (SSS or Bruhat) to a block of v vectors $B \in \mathrm{K}^{n\times v}$. We give the costs for $v \leq s$ (they can be otherwise deduced by slicing B in blocks of s columns).

4.1 SSS × dense

We here recall the algorithm of [4, §2] for computing the product of an SSS matrix with a dense matrix (independently published in [10, Alg. 7.1]). For simplicity, Algorithm LowSSSxDense only details the computations with a strictly lower-block-triangular SSS matrix, that is a matrix whose SSS representation is zero except for the P_i, Q_i and R_i. Extrapolating from there to the product with any SSS matrix can be done by transposing the algorithm for the upper-block-triangular part, and adding the product with the block-diagonal matrix made of the D_i.

Algorithm 4.1 LowSSSxDense

Input: P_i, Q_i, R_i for $i \in [\![1,N]\!]$ an s-SSS generator for a strictly lower-block-triangular matrix A; B and C dense $n \times v$ matrices
Output: $C{+} = AB$

1: Split $B = \begin{bmatrix} B_1 \\ \vdots \\ B_N \end{bmatrix}, C = \begin{bmatrix} C_1 \\ \vdots \\ C_N \end{bmatrix}$ in $s \times s$ blocks
2: $H_1 \longleftarrow Q_1 B_1$
3: **for** $i = 2 \dots N$ **do**
4: $\quad H_i \longleftarrow Q_i B_i + R_i H_{i-1}$
5: $\quad C_i \longleftarrow C_i + P_i H_{i-1}$

PROPOSITION 4.1. *The product of an $n\times n$ matrix given by its s-SSS generator with an $n \times v$ dense matrix with $v \leq s$ can be computed in* $\mathsf{T}_{\mathsf{SxDense}}(n,s,v) = 7C_\omega nsv^{\omega-2}$.

PROOF. In Algorithm LowSSSxDense we have by induction that

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

As the blocks of the product follow

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

H_{i-1} can be multiplied once by P_i to compute C_i and once by R_i to compute the following blocks. The cost is $N \times C_\omega s^2 v^{\omega-2}$ for the diagonal blocks and two applications of Algorithm LowSSSxDense in which each step costs $3C_\omega s^2 v^{\omega-2}$. □

4.2 Bruhat × dense

PROPOSITION 4.2. *The product of an $n \times n$ matrix given by its s-Bruhat generator by a dense $n\times v$ matrix with $v \leq s$ can be computed in* $\mathsf{T}_{\mathsf{BxDense}}(n,s,v) = 8C_\omega nsv^{\omega-2}$.

PROOF. This is given by [21, Alg. 14] called twice on the lower and upper triangular part of the quasiseparable matrix. □

Note that in order to benefit from fast matrix multiplication, the Bruhat generator (using $4ns$ space) needs to be transfered into a Compact-Bruhat form, by storing each echelon from into two block diagonal matrices using twice as many field elements (additonal ones being zeros). This compression can be done online, hence the space storage remains $4ns$, but the cost of the product by a dense matrix becomes $8C_\omega nst^{\omega-2}$ hence losing the advantage over the SSS format (with cost $7C_\omega nsv^{\omega-2}$ for the same operation).

4.3 Experimental comparison

Experimental results are given in Fig. 2 (Appendix A). As expected from Propositions 4.1 and 4.2 we obtain costs that are linear in s; we can also observe the same slight dependance in r of the Bruhat cost as in Section 3.5. On the parameters we chose, SSS is about four times faster than Bruhat. This can be explained by the compactification of the Bruhat generator needed for the product. This operation is free of arithmetic operations and hence does not appear in the cost of Proposition 4.2 but the data tranfers are non-negligible in practice.

5 SUM OF QUASISEPARABLE MATRICES

The sum and product of two quasiseparable matrices of order s_B and s_C are quasiseparable matrices of order at most $s_B + s_C$. In this section we show how to compute SSS and Bruhat generators for the sum of two quasiseparable matrices.

The result we give in Proposition 5.1 for the sum of matrices given in SSS form can only be used on two generators defined on the same grid. This is a drawback of most operations in SSS which is avoided with the Bruhat format. As a consequence, in a large sequence of operations, the SSS grid size needs to be chosen according to the maximal quasi-separability order among all intermediate results, while the Bruhat always fits to the current quasiseparable order. This can impact the overall cost. The slower original SSS format of [9] avoids this issue, at the expense of multiplying space and time costs by the quasiseparability order, as in [1, 2].

5.1 SSS sum

Consider two matrices B and C with the same order s. We first note that the concatenation of the blocks of both input generators leads to matrices which satisfy Eq. (1) for $A = C + B$ [4, §10.2].

Let $P_i^{(K)}, V_i^{(K)}, Q_i^{(K)}, U_i^{(K)}, R_i^{(K)}, W_i^{(K)}, D_i^{(K)}$ for appropriate $i \in [\![1, N]\!]$ be an s-SSS representation of K for $K \in \{B, C\}$. The following matrices satisfy Eq. (1) with $A = B+C$, for appropriate $i \in [\![1, N]\!]$.

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

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

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

Such sets of matrices with these dimensions satisfying Eq. (1) will be called an $(s, 2s)$-SSS generator for A. The granularity of their description remains that of $s \times s$ blocks, but the dimension of the matrices in the representation is doubled and leads to a suboptimal storage size. A second step is therefore to use Algorithm SssCompression to obtain a $2s$-SSS generator for the sum and reduce the storage size by $4s(n - 2s)$.

Proposition 5.1. *A $2s$-SSS representation of $B + C \in \mathbb{K}^{n\times n}$ can be computed from s-SSS representations of B and C in time*

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

Proof. For any $s \times s$ block $A_{i,j}$ of $A = B + C$, it can be checked that the representation in the output of Algorithm SssCompression called on the generator of Section 5.1 matches. The additions of Eq. (18) are dominated by the call to Algorithm SssCompression whose cost is of M steps with four $2s \times 2s$ by $2s \times s$ products, two $2s \times 2s$ square products, and two $s \times 2s$ by $2s \times s$ products. □

Note that the $(s, 2s)$-SSS generator is intermediate between the SSS form and the original definition of quasiseparable matrices given in [9], where the generators are $s \times s$ matrices but the granularity of the description is of dimension 1.

Algorithm 5.1 SssCompression

Input: $P_i, Q_i, R_i, U_i, V_i, W_i, D_i$ for appropriate $i \in [\![1, N]\!]$, an $(s, 2s)$-SSS generator for $A \in \mathbb{K}^{n\times n}$

Output: $P_i', Q_i', R_i', U_i', V_i', W_i', D_i'$ for appropriate $i \in [\![1, M]\!]$, a $2s$-SSS representation of A with $M = \lceil N/2 \rceil$

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

5.2 Bruhat sum

As with SSS, the sum of two matrices in Bruhat form can be computed by first concatenation of both generators, then by retrieving the Bruhat format in a second step.

Given two left triangular matrices A and B given by Bruhat generators $C^{(A)}, R^{(A)}, E^{(A)}, C^{(B)}, R^{(B)}, E^{(B)}$, their sum indeed writes

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

A Bruhat generator for the right side in Eq. (20) can be obtained from a call to Algorithm LBruhatGen, viewed here as a compression algorithm. This relies on a specific CRE decomposition (Algorithm BruhatSumCRE), and on having $D_{\mathcal{R},*}$ for D a submatrix of a sum given as in Eq. (20) and $\mathcal{R}$ a set of row indices (Proposition 5.3).

Algorithm 5.2 BruhatSumCRE

Input: $A, B \in \mathbb{K}^{n\times n}$ 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: $G, H \in \mathbb{K}^{n\times t}$

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

- $C^{(R)}, R^{(R)}, E^{(R)} \leftarrow \mathtt{DenseCRE}\left(\begin{bmatrix} R^{(A)} E^{(A)} \\ R^{(B)} E^{(B)} \\ -H^{\mathsf{T}} \end{bmatrix} \right)$
- $C^{(L)}, R^{(L)}, E^{(L)} \leftarrow \mathtt{DenseCRE}\left(\begin{bmatrix} C^{(A)} & C^{(B)} & G \end{bmatrix} \right)$
- $X \leftarrow R^{(L)} E^{(L)} C^{(R)} R^{(R)}$
- $C^{(X)}, R^{(X)}, E^{(X)} \leftarrow \mathtt{DenseCRE}(X)$
- $C \leftarrow C^{(L)} C^{(X)}$
- $R \leftarrow R^{(X)}$
- $E \leftarrow E^{(X)} E^{(R)}$

Proposition 5.2. *Algorithm BruhatSumCRE computes a CRE decomposition of $A + B - GH^{\mathsf{T}}$ in $\mathsf{T}_{\mathsf{BSumCRE}}(n, r) = (3C_{\omega} + 2C_{\mathsf{RF}})\, n r^{\omega-1}$ for $r_A + r_B + t \le r$.*

Proof. The matrices C and E are in column and row echelon form respectively as they are products of two echelon forms. The cost is that of two dense CRE decompositions of size $n\times(r_A+r_B+t)$ and products of an $n\times(r_A+r_B+t)$ matrix by two $(r_A+r_B+t)\times(r_A+r_B+t)$ and one $(r_A+r_B+t)\times n$ matrices. ◻

Proposition 5.3. *For $D\in\mathrm{K}^{n\times n}$ a submatrix of a the left-triangular part of a sum as in Eq.* (20) *and $\mathcal{R}$ a set of s row indices, $D_{\mathcal{R},*}$ can be computed in* $\mathsf{T}_{\mathsf{SumExp}}(n,s)=C_\omega ns^{\omega-1}$.

Proof. There are at most s_A (resp. s_B) pivots of A (resp. B) impacting D. We can thus write $D=CRE$ with C made of n rows and s_A+s_B columns of $\begin{bmatrix}C^{(A)} & C^{(B)}\end{bmatrix}$, R a permutation and E made of n columns and s_A rows of $E^{(A)}$ and s_B rows of $E^{(B)}$. ◻

Proposition 5.4. *The* `Bruhat` *form of the sum of two $n\times n$ matrices of quasiseparable order s_A and s_B in* `Bruhat` *form can be computed in* $\mathsf{T}_{\mathsf{B+B}}(n,s)=\left(\frac{9\cdot2^{\omega-2}-8}{2^{\omega-2}-1}C_\omega+2C_{\mathsf{RF}}\right)ns^{\omega-1}\log n/s$ *for* $s=s_A+s_B$.

Proof. Each lower and upper triangular part is converted to a left triangular instance and computed independently. Algorithm LBruhatGen is then called twice with $t=0$ on an input matrix in factorized form as in (20).

The proof is the same as for Proposition 3.5 except that in the cost, the $\mathsf{T}_{\mathsf{SparseCRE}}$ terms are replaced by $\mathsf{T}_{\mathsf{BruhatSumCRE}}$ terms and the rows and columns of the submatrices are computed at a cost given by $\mathsf{T}_{\mathsf{SumExp}}$. Then we have

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

for one call to Algorithm LBruhatGen. ◻

6 PRODUCT IN SSS

The product of two matrices given in SSS form uses two tricks we have seen previously. The first one is to start by computing an $(s,2s)$-SSS representation before compression, as in the sum. Unlike the sum, computations are needed in addition to concatenation to get this representation. The second trick is to speed up these computations by using a Horner-like accumulation as in Algorithm LowSSSxDense. This accumulation will be done on both sides for the computation of all necessary products $A_{i,k}B_{k,j}$ where $A_{i,k}$ is under (resp. over) the diagonal and $B_{k,j}$ is over (resp. under) it.

Algorithm SSSxSSS details these computations, using the G_i and H_i as accumulators. It presents an improvement over the algorithm of [4, §3] and [10, Alg. 7.2]: 4 products have been avoided at each step by keeping them in memory in the T_i and S_i. They can also be avoided in the numerical context.

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** $i\leftarrow 1\dots N$ **do**

2: $\quad G_i\leftarrow Q_{i-1}^{(A)}U_{i-1}^{(B)}+T_{i-1}W_{i-1}^{(B)}$

3: $\quad T_i\leftarrow R_i^{(A)}G_i$

4: $\quad S_i\leftarrow P_i^{(A)}G_i$

5: $\quad Q_i\leftarrow\begin{bmatrix}Q_i^{(B)}\\ Q_i^{(A)}D_i^{(B)}+T_iV_i^{(B)}\end{bmatrix}$

6: $\quad R_i\leftarrow\begin{bmatrix}R_i^{(B)} & 0\\ Q_i^{(A)}P_i^{(B)} & R_i^{(A)}\end{bmatrix}$

7: $\quad U_i\leftarrow\begin{bmatrix}U_i^{(A)} & D_i^{(A)}U_i^{(B)}+S_iW_i^{(B)}\end{bmatrix}$

8: $\quad W_i\leftarrow\begin{bmatrix}W_i^{(A)} & V_i^{(A)}U_i^{(B)}\\ 0 & W_i^{(B)}\end{bmatrix}$

9: **for** $i\leftarrow N\dots 1$ **do**

10: $\quad H_i\leftarrow V_{i+1}^{(A)}P_{i+1}^{(B)}+T_{i+1}R_{i+1}^{(B)}$

11: $\quad T_i\leftarrow U_i^{(A)}H_i$

12: $\quad D_i\leftarrow D_i^{(A)}D_i^{(B)}+S_iV_i^{(B)}+T_iQ_i^{(B)}$

13: $\quad P_i^C\leftarrow\begin{bmatrix}D_i^{(A)}P_i^{(B)}+T_iR_i^{(B)} & P_i^{(A)}\end{bmatrix}$

14: $\quad T_i\leftarrow W_i^{(A)}H_i$

15: $\quad V_i\leftarrow\begin{bmatrix}V_i^{(A)}D_i^{(B)}+T_iQ_i^{(B)}\\ V_i^{(B)}\end{bmatrix}$

16: **return** SssCompression$((P_i,Q_i,R_i,U_i,V_i,W_i,D_i)_{i\in[\![1,N]\!]})$

Theorem 6.1. *Algorithm SSSxSSS computes a $2s$-SSS generator for the product of two $n\times n$ matrices given in s-SSS form in*

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

Proof. Using Lines 2 and 4 for G_i and Lines 10 and 14 for H_i, induction on i shows that

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

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

Combining these results with Line 3 for S_i, Line 11 for T_i and finally Line 12, we get that $D_i^{(C)}=\sum_{k=1}^N A_{i,k}B_{k,i}=C_{i,i}$.

When $i<j$, the products $A_{i,k}B_{k,j}$ take five shapes: lower block of $A\times$ upper block of B, diagonal block $\times$ upper block, upper $\times$ upper, upper $\times$ diagonal and upper $\times$ lower. The equality

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

and its counterpart when $i>j$ can be checked with tedious but straightforward calculations.

The cost is that of 21 products and 8 sums of $s\times s$ matrices at each of the N steps and on call to Algorithm SssCompression. ◻

Again the result of Theorem 6.1 is limited to matrices defined on the same grid and the result always has the same storage size, whatever its quasi-separability order. This is also true for product with HSS generators in numerical analysis [22]. The `Bruhat` format can avoid these issues, but to our knowledge no sub-quadratic algorithm exists for the product of two `Bruhat` generators. The method used for the sum in Section 5.2 opens the door towards a linear or quasi-linear product algorithm using Algorithm LBruhatGen.

A EXPERIMENTS

We report here on experiments of an implementation of algorithms handling SSS and `Bruhat` generators over a finite field in the `fflas-ffpack` library [11], at commit 33474b31aa. This library provides efficient dense basic linear algebra routines, such as matrix multiplication, TRSM and Gaussian elimination revealing the rank profile matrix. It was compiled with the GNU C++ compiler g++ version 9.3.0 and linked with the OpenBLAS library version 0.3.8[1].The benchmarks are run on a single core of an Intel i5-i7300U@2.6GHz running a Linux Mint-20 system.

For all experiments, the matrices have a fixed dimension $n = 3000$, over the finite field $\mathbb{Z}/131071\mathbb{Z}$. We draw the computation times depending on the quasiseprability orders, on three type of instances: having a ranks of their upper and lower triangular parts equal to 1000, 1500 and 1750.

Each point corresponds to the mean of the running times of 50 random instances with same parameters. Figure 1 compares the running times for the generation from a dense matrix. Figure 2 compares the running times for the product by a random dense $n \times 500$ block vector, using the same generators.

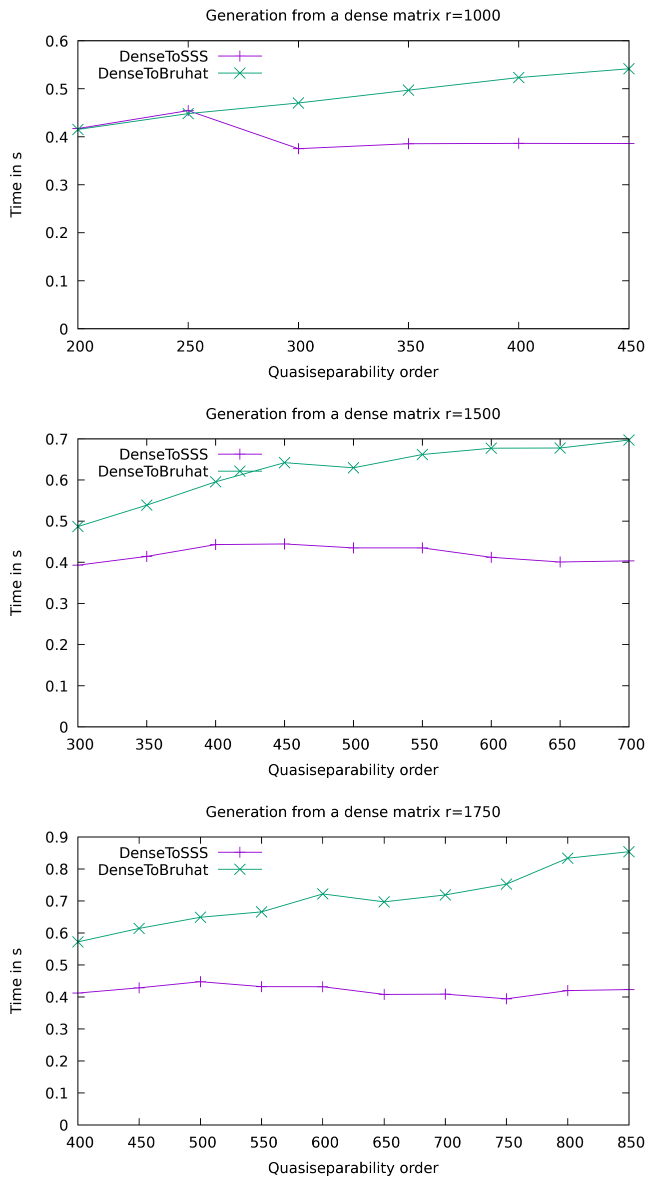

Figure 1: Experimental timings for the computation of SSS and Bruhat generators with $n = 3000$ over $\mathbb{Z}/131071\mathbb{Z}$

REFERENCES

[1] P. Boito, Y. Eidelman, and L. Gemignani. 2014. Implicit QR for rank-structured matrix pencils. *BIT Numerical Mathematics* 54, 1 (March 2014), 85–111. https://doi.org/10.1007/s10543-014-0478-0

[2] P. Boito, Y. Eidelman, and L. Gemignani. 2017. A real QZ algorithm for structured companion pencils. *Calcolo* 54, 4 (Dec. 2017), 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, and D. White. 2005. Some Fast Algorithms for Sequentially Semiseparable Representations. *SIAM J. Matrix Anal. Appl.* 27, 2 (2005), 341–364. https://doi.org/10.1137/S0895479802405884 arXiv:https://doi.org/10.1137/S0895479802405884

[4] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, and A. J. van der Veen. 2002. Fast Stable Solver for Sequentially Semi-separable Linear Systems of Equations. In *High Performance Computing — HiPC 2002*. Springer Berlin Heidelberg, 545–554. https://doi.org/10.1007/3-540-36265-7_51

[5] S. Chandrasekaran and M. Gu. 2003. Fast and Stable Algorithms for Banded Plus Semiseparable Systems of Linear Equations. *SIAM J. Matrix Analysis Applications* 25 (2003), 373–384. https://doi.org/10.1137/S0895479899353373

[6] S. Chandrasekaran, M. Gu, and T. Pals. 2006. A Fast ULV Decomposition Solver for Hierarchically Semiseparable Representations. *SIAM J. Matrix Anal. Appl.* 28, 3 (2006), 603–622. https://doi.org/10.1137/S0895479803436652

[7] Steven Delvaux and Marc Van Barel. 2008. A Givens-Weight Representation for Rank Structured Matrices. *SIAM J. Matrix Anal. Appl.* 29, 4 (2008), 1147–1170. https://doi.org/10.1137/060654967 _eprint: https://doi.org/10.1137/060654967.

[8] J.-G. Dumas, C. Pernet, and Z. Sultan. 2017. Fast computation of the rank profile matrix and the generalized Bruhat decomposition. *Journal of Symbolic Computation* 83 (2017), 187 – 210. https://doi.org/10.1016/j.jsc.2016.11.011

[9] Y. Eidelman and I. Gohberg. 1999. On a new class of structured matrices. *Integral Equations and Operator Theory* 34 (1999), 293–324. https://doi.org/10.1007/BF01300581

[10] Y. Eidelman and I. Gohberg. 2005. On generators of quasiseparable finite block matrices. *Calcolo* 42 (12 2005), 187–214. https://doi.org/10.1007/s10092-005-0102-4

[11] The FFLAS-FFPACK group. 2021. *FFLAS-FFPACK: Finite Field Linear Algebra Subroutines / Package* (v2.5.0 ed.). http://github.com/linbox-team/fflas-ffpack.

[12] W. Hackbusch. 1999. A Sparse Matrix Arithmetic Based on H-Matrices. Part I: Introduction to H-Matrices. *Computing* 62 (1999), 89–108. https://doi.org/10.1007/s006070050015

[13] W. Hackbusch. 2015. *Hierarchical Matrices: Algorithms and Analysis*. Vol. 49. Springer. https://doi.org/10.1007/978-3-662-47324-5

[14] W. Hackbusch, B. Khoromskij, and S. A. Sauter. 2000. On H2-Matrices. In *Lectures on Applied Mathematics*. Springer Berlin Heidelberg, 9–29. https://doi.org/10.1007/978-3-642-59709-1_2

[15] C.-P. Jeannerod, C. Pernet, and A. Storjohann. 2013. Rank-profile revealing Gaussian elimination and the CUP matrix decomposition. *J. Symbolic Comput.* 56 (2013), 46–68. https://doi.org/10.1016/j.jsc.2013.04.004

[16] W. Lyons. 2005. *Fast algorithms with applications to PDEs*. Ph. D. Dissertation. University of California, Santa Barbara, USA. http://scg.ece.ucsb.edu/publications/theses/Lyons_2005_Thesis.pdf

[17] W. Manthey and U. Helmke. 2007. Bruhat canonical form for linear systems. *Linear Algebra Appl.* 425, 2–3 (2007), 261–282. https://doi.org/10.1016/j.laa.2007.01.022

[18] P.G. Martinsson. 2011. A Fast Randomized Algorithm for Computing a Hierarchically Semiseparable Representation of a Matrix. *SIAM J. Matrix Analysis Applications* 32 (10 2011), 1251–1274. https://doi.org/10.1137/100786617

[19] Clément Pernet. 2016. Computing with Quasiseparable Matrices. In *Proc. ISSAC* (Waterloo, ON, Canada). ACM Press, 389–396. https://doi.org/10.1145/2930889.2930915

[20] C. Pernet, H. Signargout, and G. Villard. 2023. *Leading constants of rank deficient Gaussian elimination*. Technical Report. `hal:03976168`.

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

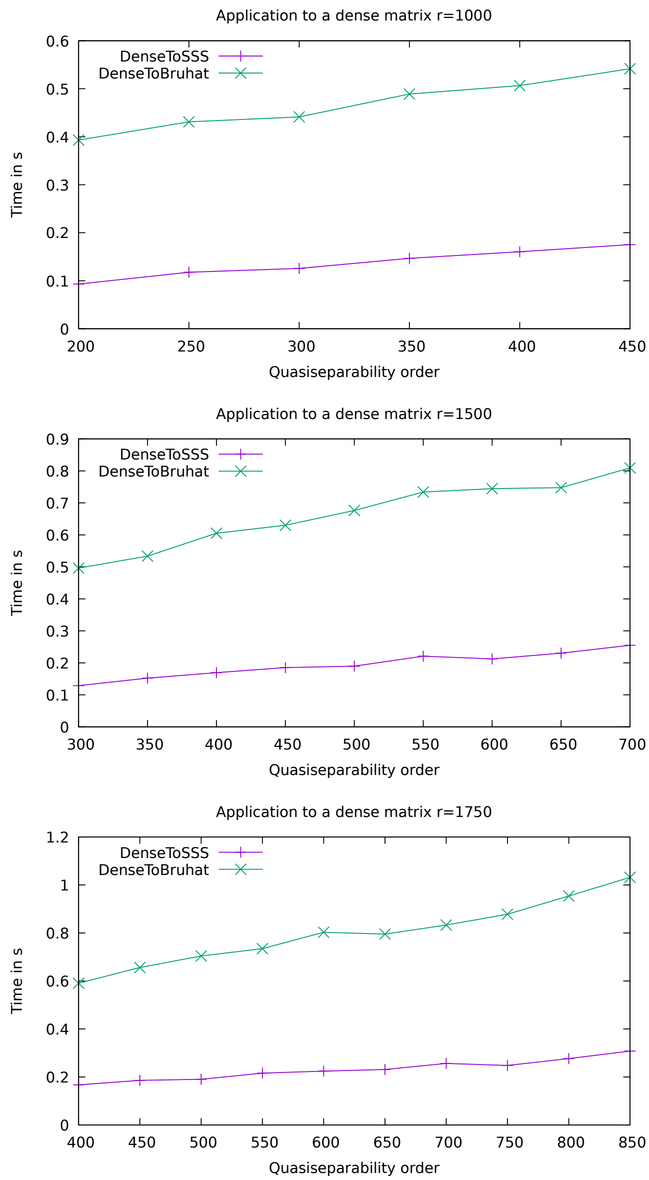

Figure 2: Experimental timings for the computation of SSS and Bruhat times a dense matrix with $n = 3000$ and $v = 500$ over $\mathbb{Z}/131071\mathbb{Z}$

[21] C. Pernet and A. Storjohann. 2018. Time and space efficient generators for quasiseparable matrices. *Journal of Symbolic Computation* 85 (2018), 224 – 246. https://doi.org/10.1016/j.jsc.2017.07.010

[22] Z. Sheng, P. Dewilde, and S. Chandrasekaran. 2007. *Algorithms to Solve Hierarchically Semi-separable Systems.* Vol. 176. 255–294. https://doi.org/10.1007/978-3-7643-8137-0_5

[23] H.P. Starr. 1992. *On the Numerical Solution of One-Dimensional Integral and Differential Equations.* Ph. D. Dissertation. Yale University, USA. https://cpsc.yale.edu/sites/default/files/files/tr888.pdf UMI Order No. GAX92-35558.

[24] A. Storjohann. 2000. *Algorithms for Matrix Canonical Forms.* Ph. D. Dissertation. Institut für Wissenschaftliches Rechnen, ETH-Zentrum, Zürich, Switzerland. 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, and N. Mastronardi. 2005. A bibliography on semiseparable matrices*. *CALCOLO* 42 (2005), 249–270. https://doi.org/10.1007/s10092-005-0107-z

[26] R. Vandebril, M. Van Barel, Gene H. Golub, and N. Mastronardi. 2008. *Matrix Computations and Semiseparable Matrices: Linear Systems.* Johns Hopkins University Press. https://doi.org/10.1353/book.16537

[27] J. Xia, S. Chandrasekaran, M. Gu, and X.S. Li. 2010. Fast algorithms for hierarchically semiseparable matrices. *Numerical Linear Algebra with Applications* 17, 6 (2010), 953–976. https://doi.org/10.1002/nla.691