Summer 8-12-2014

Row Compression and Nested Product Decomposition of a Hierarchical Representation of a Quasiseparable Matrix

Mary Hudachek-Buswell
Georgia State University

ROW COMPRESSION AND NESTED PRODUCT DECOMPOSITION OF A HIERARCHICAL REPRESENTATION OF A QUASISEPARABLE MATRIX

by

MARY HUDACHEK-BUSWELL

Under the Direction of Dr. Michael Stewart

ABSTRACT

This research introduces a row compression and nested product decomposition of an $n \times n$ hierarchical representation of a rank structured matrix A, which extends the compression and nested product decomposition of a quasiseparable matrix. The hierarchical parameter extraction algorithm of a quasiseparable matrix is efficient, requiring only $O(n\log(n))$ operations, and is proven backward stable. The row compression is comprised of a sequence of small Householder transformations that are formed from the low-rank, lower triangular, off-diagonal blocks of the hierarchical representation. The row compression forms a factorization of matrix A, where $A = QC$, Q is the product of the Householder transformations, and C preserves the low-rank structure in both the lower and upper triangular parts of matrix A. The nested product decomposition is accomplished by applying a sequence of orthogonal transformations to the low-rank, upper triangular, off-diagonal

blocks of the compressed matrix C. Both the compression and decomposition algorithms are stable, and require $O(n\log(n))$ operations. At this point, the matrix-vector product and solver algorithms are the only ones fully proven to be backward stable for quasiseparable matrices. By combining the fast matrix-vector product and system solver, linear systems involving the hierarchical representation to nested product decomposition are directly solved with linear complexity and unconditional stability. Applications in image deblurring and compression, that capitalize on the concepts from the row compression and nested product decomposition algorithms, will be shown.

KEY WORDS: Hierarchical matrices, Row compression, Nested product decomposition, Stable algorithms, Quasiseparable matrices, Image processing

ROW COMPRESSION AND NESTED PRODUCT DECOMPOSITION OF A HIERARCHICAL REPRESENTATION OF A QUASISEPARABLE MATRIX

by

MARY HUDACHEK-BUSWELL

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2014

ROW COMPRESSION AND NESTED PRODUCT DECOMPOSITION OF A HIERARCHICAL REPRESENTATION OF QUASISEPERABLE MATRICES

by

MARY HUDACHEK-BUSWELL

Committee Co-Chair: Saeid Belkasim

Committee Co-Chair: Michael Stewart

Committee: Raj Sunderraman

Yi Pan

Jon Preston

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

August 2014

DEDICATION

Dedicated to my loving parents, John and Anne Hudachek, who are watching from above; and to my very patient husband, Dan Buswell, who is ecstatic to have his wife back.

ACKNOWLEDGEMENTS

The monumental task of a doctoral degree can only be undertaken with the consistent support and encouragement of many individuals. First and foremost, I must thank Professor Michael Stewart for directing my research all these years. His instruction, patience, and support have made me the researcher I am today. Working with Professor Stewart has been a great honor, and I look forward to our continued collaboration. I wish to thank the Computer Science faculty at Georgia State University for their leadership in guiding me towards a doctoral degree in computing. Special thanks goes to my co-chair, Professor Belkasim for introducing me to image processing and encouraging me to research in this area. I express gratitude to my committee members, Professors Raj Sunderraman, Yi Pan and Jon Preston, for their valuable and constructive feedback on my research.

Many thanks go to my mentor, Professor Catherine Aust, who is responsible for starting me down this doctoral path in computer science, and assisted me in navigating the process. I am grateful to my dear colleague, Professor Catherine Matos, for her steadfast support, and boundless efforts to discuss my research at all hours of the day. I also extend my thanks to Professor Rebecca Rizzo for graciously opening her office to me this last year. I would like to acknowledge Mr. Trey Olmsted for his technical skills in maintaining my research computer, and providing me with the freedom to access my research anywhere at anytime. My sincere thanks also go to Ms. Tina Breckenridge for reminding me to laugh during the last decade, and cheering me onward.

Lastly, I thank my entire family. In the end, all you have is family; it is only through their love, care, and support that I have been able to finish this research marathon. Especially, my husband Dan for travailing this decade long journey with me, and to my brother Michael for his unwavering moral support and financial assistance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CG	Conjugate gradient
CL	Cauchy-like
DCT	Discrete cosine transform
DFT	Discrete Fourier transform
DWT	Discrete wavelet transform
FFT	Fast Fourier transform
FMM	Fast multipole method
GMRES	Generalized minimal residual
$\mathscr{H}$	Hierarchical
HSS	Hierarchically semiseparable
PDE	Partial differential equations
PSF	Point spread function
SSS	Sequentially semiseparable
SVD	Singular value decomposition

NOTATIONS

A	A general matrix A with elements $a_{i,j}$.
A_c	A column blurring matrix.
A_r	A row blurring matrix.
$\mathscr{B}$	A basis for the column space.
$B_{(l,b)}$	An off-diagonal block.
B_i	A banded lower triangular matrix with regards to the nested UBV decomposition.
$\mathbb{C}$	Denotes the set of complex numbers.
C	A compressed or Cauchy-like matrix.
D	A diagonal matrix.
$\mathscr{H}$	Hierarchical matrix.
H	Hessenberg matrix.
I_k	The identity matrix of size $k \times k$.
L	A lower triangular matrix with regards to the $LU-$factorization.
P	Permutation or transformation.
Q	A unitary (orthogonal) matrix with regards to the $QR-$factorization.
R	An upper triangular matrix with regards to the $QR-$factorization.
$\mathbb{R}$	Denotes the set of real numbers.
Σ	A diagonal matrix of singular values.
T	A Toeplitz matrix.
Z	A shift matrix.

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

Several meaningful problems in applied mathematics, engineering, and computer science have patterns or structure that translate into corresponding classes of structured matrices. Structured matrices have been studied for decades. Currently, the attention towards structured matrices has amplified due to the interdisciplinary uses and special properties of structured matrices. When the properties of structured matrices are exploited, fast numerical methods and methods for the optimization of computation storage can be developed [1, 2]. Matrix structure can be used in solving linear systems, least squares problems, integral equations, eigenvalue problems, and partial differential equations via direct and iterative solvers. Even with the considerable advances in developing fast algorithms for structured matrices, extensive work remains to be done in more challenging areas, such as multilevel structured matrices.

Structured matrices often can be described by a compact formula for their entries. Some classic examples of structured matrices are Cauchy $\left(a_{ij} = \frac{1}{b_i - c_j}\right)$, Hankel $(a_{ij} = a_{i+j})$, Toeplitz $(a_{ij} = a_{i-j})$, and Vandermonde $\left(a_{ij} = a_i^{j-1}\right)$ matrices. The structured matrix class is broad, containing several subclasses, with a prevalent subclass being that of rank structured matrices. A matrix is considered rank structured if certain submatrices have low numerical compared to the size of the matrix. However, among researchers there appears to be no uniform agreement on terminology for distinguishing certain classes within the rank structured matrix class. Various rank structured matrix classes, such as quasiseparable matrices [3–7], $\mathscr{H}$-matrices [8–10], $\mathscr{H}^2$-matrices [11, 12], sequentially semiseparable matrices [13–16], and hierarchically semiseparable matrices [17–21] may have similar attributes and

representations, but are different and the distinctions can lead to some confusion. The set of $\mathscr{H}$-matrices is comprised of dense matrices with a data-sparse representation, in which the matrix is split into a hierarchy of blocks with clusters located on the diagonal [8, 9, 22–24]. The subset of $\mathscr{H}^2$-matrices is more refined, and has a second hierarchy of clusters that exist but are not on the diagonal [8, 9, 22, 23].

Rank structured matrices are matrices for which certain blocks have rank bounded by a small constant. Data sparse parameterizations for these rank structured matrices can be as simple as a low rank factorization representing the low rank blocks. Some representations allow further compression of the matrix by exploiting common row and column spaces that occur between blocks. Quasiseparable matrices are matrices where the blocks strictly below or strictly above the diagonal have bounded rank. In other words, quasiseparable matrices are of low numerical rank in the off-diagonal blocks [4, 6, 25]. Semiseparable matrices have lower triangular and upper triangular parts that are the same as those of a low rank matrix [13, 18]. Note that semiseparable matrices are a proper subset of quasiseparable matrices. Quasiseparable matrices in this work are not to be confused with semiseparable matrices. This research is centered on quasiseparable matrices and their representations.

Rank structured matrices have emerged in many applications in computer science, systems engineering, electrical engineering and applied mathematics [1, 26]. Bridging across disciplines is what makes this class of matrices one of the hottest topics in numerical linear algebra in the last few years. Some of the more challenging problems and applications are found in multilevel representations of quasiseparable matrices. Quasiseparable matrices have a compressed representation of their off-diagonal blocks, which have low numerical rank. A matrix representation is the set of parameters that are used to represent the matrix. There are three representations discussed in this research: generator, hierarchical and nested product. Hierarchical representations of structured matrices have a structure based on the blocks of the matrix having some form of hierarchy [2, 17–19, 27]. By applying a hierarchy to a general quasiseparable matrix, the result is a hierarchical representation of a quasiseparable

matrix. Currently there is a need for stable representations of quasiseparable matrices, and fast factorization algorithms for quasiseparable matrices.

Hierarchical representations of structured matrices arise in some significant applications within computer graphics, signal processing and electromagnetics [9, 28–30]. Two image processing applications can benefit from fast, stable algorithms involving hierarchical representations of quasiseparable matrices: image compression or coding and image restoration. Wavelet transforms are used to compress images, and this research explores the possibility of a hierarchy and rank structure existing in wavelets. It is possible to further compress the image if a hierarchical representation of the wavelet exists, by performing the row compression and nested product conversion on the wavelet. A quasiseparable matrix can model satellite blur, and then the fast Fourier transform (FFT) is used to transform the blur model. When the transformed blur model is solved, the image can be restored. The hierarchical representation of the quasiseparable matrix is used as a preconditioner in an iterative technique to restore the image.

In research, it is important to form a stable hierarchical representation of the quasiseparable matrix A to exploit the low rank structure. Once formed, one must be able to perform a compression on the parameterization of A. The flexibility of converting to another representation is imperative to accessing existing algorithms for other representations. When solving matrix A, one wants not only a fast direct solver, but a proven stable solver. The major contributions of this research are the new row compression and conversion to nested product algorithms for the hierarchical representation of a quasiseparable matrix. The row compression and conversion to nested product algorithms focus on stability and have comparable computational complexity to existing algorithms.

1.2 Numerical Linear Algebra Fundamentals

In this section, we will discuss basic terminology and matrix theory to be used throughout this document. Notations for the fields of real numbers and complex numbers

are $\mathbb{R}$ and $\mathbb{C}$, respectively. Given a matrix $A \in \mathbb{R}^{n\times n}$, its (i,j)th element is denoted by $a_{i,j}$. We let A^{T} denote the *transpose* of the matrix A where $(A^{\mathrm{T}})_{i,j} = a_{j,i}$. The *conjugate* of an element is $a_{i,j} = \overline{a}_{i,j}$. A square matrix A is considered *Hermitian*, if $A^{\mathrm{H}} = A$ where A^{H} denotes the conjugate transpose of A such that $A^{\mathrm{H}} = (\overline{A})^{\mathrm{T}} = \overline{A^{\mathrm{T}}}$. Matrix $A \in \mathbb{R}^{n\times n}$ is *symmetric*, if $A = A^{\mathrm{T}}$. A matrix $Q \in \mathbb{R}^{n\times n}$ is *orthogonal* if $Q^{\mathrm{T}} = Q^{-1}$ and $QQ^{\mathrm{T}} = Q^{\mathrm{T}}Q = I$, where I is the identity matrix. The canonical unit vector, $\mathbf{e}_i$, has 1 in the ith entry and zeros elsewhere. [31–33].

Matrices with upper and lower bandwidth are special types of matrices that are known as band matrices. A matrix $A \in \mathbb{R}^{n\times n}$ has *lower bandwidth* p if $a_{i,j} = 0$ for $i > j+p$ and *upper bandwidth* q if $a_{i,j} = 0$ for $j > i+q$ [34]. The notational convention used in this proposal to display the zero pattern in a matrix is the *Wilkinson diagram* from J. H. Wilkinson [35]. The symbol $\times$ represents an element that may be nonzero, and 0 represents a zero element. Thus, a *Wilkinson diagram* for a band matrix, with lower bandwidth equal to 1 and upper bandwidth equal to 2, might look like

$$\begin{bmatrix} \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & 0 \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}.$$

The following is a Wilkinson diagram of a 5×5 *diagonal* matrix D where the matrix elements $d_{i,j} = 0$ when $i \neq j$:

$$\begin{bmatrix} \times & 0 & 0 & 0 & 0 \\ 0 & \times & 0 & 0 & 0 \\ 0 & 0 & \times & 0 & 0 \\ 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & 0 & \times \end{bmatrix}.$$

A square matrix U is *upper triangular* if $u_{i,j} = 0$ whenever $i > j$, and is of the form

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{bmatrix}.$$

Similarly, a square matrix L is *lower triangular* if $l_{i,j} = 0$ whenever $i < j$. To denote the lower triangular portion of a matrix A, we adopt the MATLAB notation of tril(A). The upper triangular portion is triu(A). A matrix A is *upper Hessenberg* if $a_{i,j} = 0$ whenever $i > j + 1$. Thus zeros fall below the subdiagonal, and such a matrix has the form

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}.$$

In all numerical algorithms, it is necessary to examine numerical stability. Let f be a function acting on data $\mathbf{d} \in S$ to produce a solution $\mathbf{y} = f(\mathbf{d})$ to some mathematical problem. For example, $f(\mathbf{d}) = A^{-1}\mathbf{y}$ in the case in which the problem is solving $A\mathbf{y} = \mathbf{d}$. Suppose $\hat{\mathbf{d}}$ is some approximation to $\mathbf{d}$. A numerical problem is said to be *well-conditioned* if $f(\hat{\mathbf{d}})$ is always close to $f(\mathbf{d})$, when $\hat{\mathbf{d}}$ is close to $\mathbf{d}$. Now in general, $f(\hat{\mathbf{d}})$ and $f(\mathbf{d})$ can differ greatly when $\hat{\mathbf{d}}$ is close to $\mathbf{d}$, and if this is true then the problem is said to be *ill-conditioned*. An algorithm is considered *unstable* if it introduces large errors in the computed solutions to a well-conditioned problem. There are two types of error to consider in analyzing the accuracy of an algorithm, forward and backward error, which can be seen in Figure 1.1. Backward error analysis is easier to carry out and is better suited for matrix algorithms

[31, 32, 35]. More importantly, backward error analysis clearly differentiates between the effects of ill-conditioning and algorithmic instability on the accuracy of a computed solution. Let $\hat{f}$ represent an algorithm that computes an approximation to f such that $\hat{f}(\mathbf{d})$ is close to $f(\mathbf{d})$. Forward error analysis finds the bounds for the errors of $\|\hat{f}(\mathbf{d}) - f(\mathbf{d})\|$. Backward error analysis examines $\|\hat{\mathbf{d}} - \mathbf{d}\| \leq O(\epsilon)$ where $\hat{\mathbf{d}}$ is the perturbed data, and the computed solution is the exact solution, $\hat{f}(\mathbf{d}) = f(\hat{\mathbf{d}})$ [32, 36, 37].

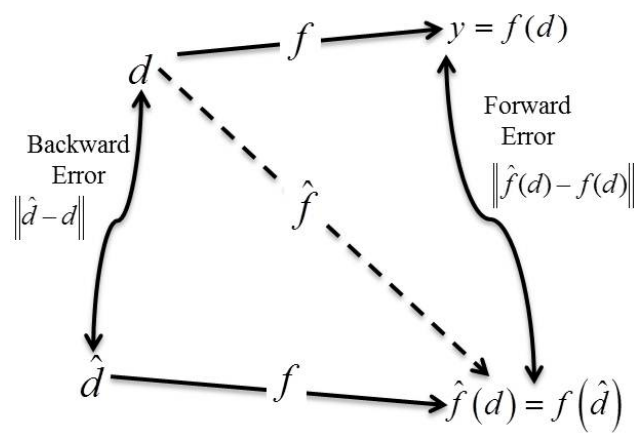


Figure 1.1. Diagram of forward and backward error analysis. Given a function, f, acting on data $\mathbf{d}$, an algorithm, $\hat{f}$ which produces an approximation to $f(\mathbf{d})$, and the approximation set of data $\hat{\mathbf{d}}$. Forward error analysis looks at the bounds for $\|\hat{f}(\mathbf{d}) - f(\mathbf{d})\|$. Backward error begins with $\hat{f}(\mathbf{d}) = f(\hat{\mathbf{d}})$ and works back towards $\|\hat{\mathbf{d}} - \mathbf{d}\|$ to determine bounds on the data set.

The backward error analysis in [3] is pertinent to the stability of the research in this dissertation. Given the theorem from [3] on errors in applying Householder transformations in the QR decomposition. Assume the usual model for floating point arithmetic with u as the unit round-off, $nu < 1$, and c is a small constant, and the bound $\tilde{\gamma}_n = \frac{cnu}{1-cnu}$. Consider $\hat{A}_{j+1} = \mathrm{fl}(\hat{Q}_j \hat{A}_j)$ where $j = 1, 2, \ldots, p$ and $A_1 = A$ is $m \times n$. Then there exists $\tilde{Q}_j$ satisfying $\tilde{Q}_j^{\mathrm{T}} \tilde{Q}_j = I$ and $\|\tilde{Q}_j - \hat{Q}_j\|_F \leq \tilde{\gamma}_m$ for which

$$\hat{A}_{p+1} = \tilde{Q}A + E, \quad \|E\|_F \leq \tilde{\gamma}_m \|A\|_F$$

where $\tilde{Q} = \tilde{Q}_p \cdots \tilde{Q}_1$.

From [3], given the row compression algorithm $A = QC$, there is error represented by

$$A + E_A = \tilde{Q}\tilde{C} \tag{1.1}$$

where $\|E_A\|_2 = O(u)\|A\|_2$ and $\frac{\|E_A\|_2}{\|A\|_2} = O(u)$. Observe that for $\|E_A\|_2$ the process begins with $C_0 = A$, and for each $k = 0, 2, \ldots, m-1$

$$\mathrm{fl}(\hat{Q}_{k+1}^{\mathrm{T}}\hat{C}_k) = \tilde{Q}_{k+1}^{\mathrm{T}}\hat{C}_k + E_{k,A} \quad \text{for} \quad \|E_{k,A}\|_F \leq d\tilde{\gamma}_n\|C_k\|_F.$$

After introducing a threshold $\|T_{k,A}\|_F \leq t = O(\|A\|)$, one obtains $\hat{C}_{k+1} = \tilde{Q}_{k+1}^{\mathrm{T}}\hat{C}_k + E_{k,A} + uT_{k,A}$. The $\hat{C}_{k+1}$ equation is iterated while applying the triangle inequality and the unitary invariance. This results in

$$\|\hat{C}_m - \tilde{Q}_m^{\mathrm{T}}\tilde{Q}_{m-1}^{\mathrm{T}} \cdots \tilde{Q}_1^{\mathrm{T}}A\|_F \leq \sum_{k=1}^{m} \|E_{k,A} + u\|_F\|T_{k,A}\|_F,$$

and from the theorem in [3] they imply $\|A - \tilde{Q}_1\tilde{Q}_2 \cdots \tilde{Q}_m\hat{C}_m\|_F \leq md\tilde{\gamma}_n\|A\|_F + mtu + O(u^2)$. Through substitution, $A + E_A = \tilde{Q}\tilde{C}$ is justified and backward error is achieved.

Unstable algorithms can produce poor results to well-conditioned problems. If solving an ill-conditioned problem with a stable algorithm, the solution is not any more or less accurate than the data warrant.

Given a matrix $A \in \mathbb{R}^{m\times n}$ for $m \geq n$. A *singular value decomposition* (SVD) of A is a factorization of the form $A = U\Sigma V^{\mathrm{T}}$ where $U \in \mathbb{R}^{m\times n}$ has orthogonal columns such that $\mathbf{u}^{\mathrm{T}}\mathbf{u} = I$, $V \in \mathbb{R}^{n\times n}$ is orthogonal with $\mathbf{v}^{\mathrm{T}}\mathbf{v} = I = \mathbf{v}\mathbf{v}^{\mathrm{T}}$, and $\Sigma \in \mathbb{R}^{n\times n}$ is diagonal. The

SVD of a 5×3 matrix A looks like

$$\overset{A}{\begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix}} = \overset{U}{\begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix}} \overset{\Sigma}{\begin{pmatrix} \times & 0 & 0 \\ 0 & \times & 0 \\ 0 & 0 & \times \end{pmatrix}} \overset{V^{\mathrm{T}}}{\begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix}}. \tag{1.2}$$

Matrix A has a *QR-factorization* $A = QR$ where $Q \in \mathbb{R}^{m \times m}$ is orthogonal and $R \in \mathbb{R}^{m \times n}$ is upper triangular with positive diagonal elements. The QR factorization is an orthogonal reduction of A to triangular form. A Wilkinson diagram follows [31, 32, 35]:

$$\overset{A}{\begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix}} = \overset{Q}{\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}} \overset{R}{\begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}}. \tag{1.3}$$

Similarly, an *LQ-factorization* of the same matrix A is $A^T = LQ$ where Q is orthonormal and L is lower triangular, and the *LQ-factorization* is the transpose of *QR-factorization*,

$$\overset{A^T}{\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}} = \overset{L}{\begin{pmatrix} \times & 0 & 0 & 0 & 0 \\ \times & \times & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 \end{pmatrix}} \overset{Q}{\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}}.$$

Matrix $A \in \mathbb{R}^{n\times n}$ has an LU factorization if the $\det(A(1:k,1:k)) \neq 0$ for $k = 1:n-1$. If the LU factorization exists and A is nonsingular, then the LU factorization is unique and is of the form $A = LU$ where L is a lower triangular matrix with zeros above the diagonal, and U is an upper triangular matrix with zeros below the diagonal [36]. The LU factorization of a 3×3 matrix A appears below:

$$\overset{A}{\begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix}} = \overset{L}{\begin{pmatrix} \times & 0 & 0 \\ \times & \times & 0 \\ \times & \times & \times \end{pmatrix}} \overset{U}{\begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{pmatrix}} \tag{1.4}$$

.

A *Householder transformation*, or elementary reflector, is a matrix of the form $P = I - 2\mathbf{u}\mathbf{u}^T$, where $\|\mathbf{u}\|_2 = 1$. Note that P is symmetric and orthogonal. Given a matrix $A \in \mathbb{R}^{n\times n}$. Typically it is computed to introduce zero elements into a particular column of A. The formula for the Householder vector, P, that transforms $\mathbf{x}$ to a multiple of $\mathbf{e}_i$ is $\mathbf{u} = \text{sign}(\mathbf{x}_i)\|\mathbf{x}\|_2\mathbf{e}_i + \mathbf{x}$ where $\mathbf{x}$ is a column of A and x_i is the ith element of $\mathbf{x}$. Householder transformations are very desirable because of their unconditional numerical stability. For example, A is a 5×5 matrix, and $\mathbf{u}$ is the Householder vector that operates on A while introducing zeros into the first column of A as seen in Householder reflector P where p is an element of $P = I - 2\mathbf{u}\mathbf{u}^T$:

$$\overset{P}{\begin{pmatrix} p & p & p & p & p \\ p & p & p & p & p \\ p & p & p & p & p \\ p & p & p & p & p \\ p & p & p & p & p \end{pmatrix}} \overset{A}{\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}} = \overset{\tilde{A}}{\begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix}}. \tag{1.5}$$

After multiple steps of the QR factorization algorithm, matrix A can be reduced to upper triangular, via a sequence of Householder transformations forming Q as a product of P, $Q = P_1 P_2 \dots P_{n-1}$, while operating on updated versions of A [31, 35, 36].

Another numerical method that introduces zeros into a matrix is *Givens rotations*. A plane rotation or Givens rotation is a matrix of the form

$$G_{i,j} = \begin{bmatrix} I_{i-1} & & & & \\ & c & & -s & \\ & & I_{j-i-1} & & \\ & s & & c & \\ & & & & I_{n-j} \end{bmatrix},$$

where $c = \cos\theta, s = \sin\theta$, and $c^2 + s^2 = 1$. Givens rotations' cost is double that of Householder reflectors in the QR decomposition [31, 35, 36].

The *column space* of a matrix $A \in \mathbb{R}^{m\times n}$ is the vector space generated by the columns of A. A vector $\mathbf{y} \in \mathbb{R}^m$ belongs to the column space of A if and only if $\mathbf{y} = A\mathbf{x}$ for some vector $\mathbf{x} \in \mathbb{R}^n$. Similarly, the rows of A generate a vector space which is called the *row space* of A, and is also referred to as the rank(A). We have rank$(A) \leq r$ if and only if there exist U and V in $\mathbb{R}^{m\times r}$ and $\mathbb{R}^{n\times r}$ respectively, with linearly independent columns, such that

$$A = UV^T.$$

The columns of both U and V contain bases for the column and row spaces of A, respectively [36]. An example follows with $A \in \mathbb{R}^{5\times 5}$ and rank$(A) = r = 3$ where $A = UV^T$ is represented

by:

$$\overset{A}{\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}} = \overset{U}{\begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix}} \overset{V^T}{\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}}. \tag{1.6}$$

It is often useful to approximate a large matrix with a simpler matrix of lesser rank. An $n \times n$ matrix of rank $r < n$ can be reconstructed from at most $2nr$ numbers. By implementing this approximation, one has *compressed* the matrix. The abatement in arithmetic matrix computations is striking when r is small, in which case the complexity of many operations changes from $O(n^2)$ to $O(nr)$.

Matrix row compression is a very useful numerical procedure that essentially factors a matrix, reduces one of the factors, and thus reduces the matrix. A simple definition for row compression is the row compression of a matrix $A \in \mathbb{R}^{m \times n}$ results in $A = QB$ where $Q \in \mathbb{R}^{m \times m}$ and $B \in \mathbb{R}^{m \times n}$. However, B is reduced since the rank$(B) = r$ and

$$QB = \begin{bmatrix} B_1 \\ 0 \end{bmatrix},$$

where $B_1 \in \mathbb{R}^{r \times n}$.

Matrices can be *partitioned* or divided into blocks. Generally, a matrix $A \in \mathbb{R}^{m \times n}$ can be rewritten as follows:

$$A = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1q} \\ A_{21} & A_{22} & \dots & A_{2q} \\ \vdots & \vdots & & \vdots \\ A_{p1} & A_{p2} & \dots & A_{pq} \end{bmatrix},$$

where $A_{ij} \in \mathbb{R}^{m_i \times n_j}$ are submatrices called *blocks* such that $m_1 + \ldots + m_p = m$ and $n_1 + \ldots + n_q = n$ [35]. We refer to A_{ij} as a block (submatrix) of matrix A.

1.3 Structured Matrices

An $m \times n$ structured matrix is a matrix with elements that can be defined in terms of substantially fewer than $m \cdot n$ parameters. Within the structured matrix class, there is the important subclass of rank structured matrices. This section looks at the definitions of different classes of rank structured matrices and the nuances of those differences. A set of structured matrices and a choice of a matrix parameterization are two distinct concepts, and in the next section we will discuss the parameterizations of matrices.

A matrix $A \in R^{n \times n}$ is rank structured if it has one or more submatrices which have a small upper bound on the ranks relative to the size of the matrix [16, 38]. The class of rank structured matrices circumscribes quasiseparable matrices and has relationships with Toeplitz and Cauchy matrices. Many fast algorithms have been developed that take advantage of the data sparse parameterizations of these matrices [13, 30, 39–41]. The relations between subclasses of rank structured matrices is shown in Figure 1.2.

The class of quasiseparable matrices is the essence of this research. Following [3], we will refer to an $n \times n$ matrix A as *quasiseparable* of order (r, s) if all partitionings are of the form

$$A = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix} \tag{1.7}$$

where A_{11} is $k \times k$ for $k = 1, 2, ..., n-1$ satisfy

$$\operatorname{rank}\left(A_{21}^{(k)}\right) = r_k(A) \leq r \qquad \text{and} \qquad \operatorname{rank}\left(A_{12}^{(k)}\right) = s_k(A) \leq s. \tag{1.8}$$

Superscripts in parentheses indicate the size of the leading principal submatrix. We refer to $r_k(A)$ as the *lower rank sequence* and $s_k(A)$ as the *upper rank sequence*. Quasiseparable matrices structure is preserved (with possible boundable increases in rank) by the following

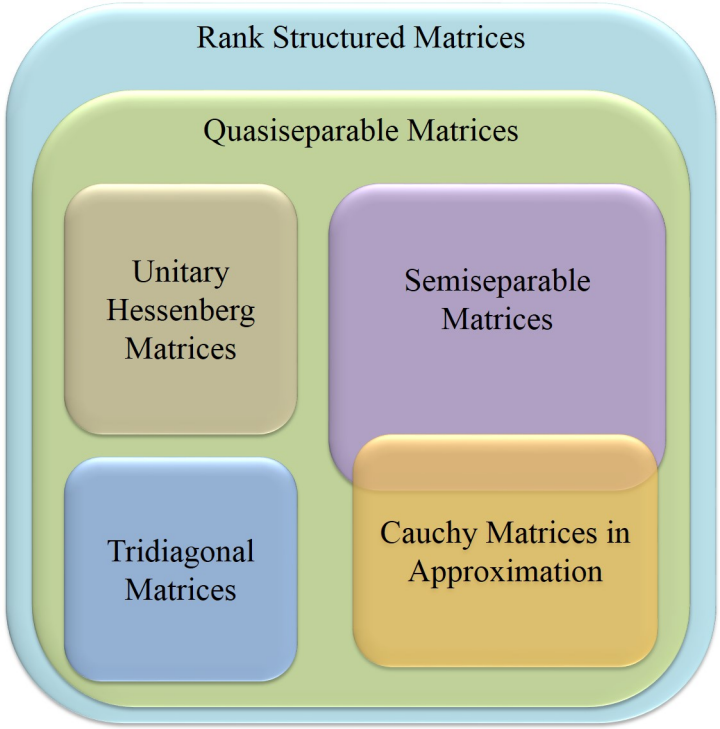


Figure 1.2. Rank structured matrices and their corresponding relationships.

operations: matrix multiplication, inversion, QR factorization, LU factorization (without pivoting), and (in special cases) the QR iteration [3]. The class of quasiseparable matrices encompasses both tridiagonal and unitary Hessenberg matrices.

A subclass of quasiseparable matrices is semiseparable matrices. An $n \times n$ matrix A is called (r_L, r_U) semiseparable if for some r_L and r_U, $A = D + \text{tril}(R_L) + \text{triu}(R_U)$, with the ranks of R_L and R_U equal to r_L and r_U respectively [42, 43]. For example,

$$R_L = \begin{bmatrix} k_1 l_1 & k_1 l_2 & k_1 l_3 \\ k_2 l_1 & k_2 l_2 & k_2 l_3 \\ k_3 l_1 & k_3 l_2 & k_3 l_3 \end{bmatrix}, \quad R_U = \begin{bmatrix} t_1 u_1 & t_1 u_2 & t_1 u_3 \\ t_2 u_1 & t_2 u_2 & t_2 u_3 \\ t_3 u_1 & t_3 u_2 & t_3 u_3 \end{bmatrix}, \quad D = \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix},$$

$$A = \begin{bmatrix} d_1 & t_1u_2 & t_1u_3 \\ k_2l_1 & d_2 & t_2u_3 \\ k_3l_1 & k_3l_2 & d_3 \end{bmatrix}.$$

As can be seen in the definitions in this section, rank structured matrices, quasiseparable matrices, and semiseparable matrices have clear differences. The class of rank structured matrices is the broadest of the three classes, and encapsulates the other two subclasses. The next largest, is the quasiseparable matrix class which in turn contains semiseparable matrices. However, underlying all the differences is the same characteristic property of small numerical ranks in the submatrices within the larger matrix. The basis of the quasiseparable matrix structure is that the low rank blocks are strictly below or above the main diagonal. There is much discussion on developing fast algorithms for quasiseparable matrices. Developing fast algorithms focuses not only on the definition of a matrix, but how it is represented or parametrized. The matrix parameterizations explored in this research are generator, hierarchical and nested product.

1.4 Representations of Matrices

Simple classes of matrices, such as banded or tridiagonal, are straightforward to represent. For rank structured matrices, which are often dense, computing a parameterization is not a trivial task. Choosing how to extract the parameters or form the parameterization of a rank structured matrix is not obvious even though the matrix may only use a few parameters. One must also be mindful of the fact that representations behave differently across different sets of matrices, and are similar in name and description only. The paper will look at three representations, and focus on their interaction with quasiseparable matrices.

Using a definition from [16, 38], a *representation* is where an element $\mathbf{v} \in \mathcal{V}$ is said to *represent* an element $\mathbf{u} \in \mathcal{U}$ if there is a map r

$$r : \mathcal{V} \subseteq \mathcal{X} \to \mathcal{U} \subseteq \mathcal{W},$$

where the sets $\mathcal{U}$ and $\mathcal{V}$ are contained in the vector spaces $\mathcal{X}$ and $\mathcal{W}$, respectively. Additionally, $\dim(\mathcal{X}) \leq \dim(\mathcal{W})$, $r(\mathcal{V}) = \mathcal{U}$ is surjective, and there exists a map $s : \mathcal{U} \to \mathcal{V}$ such that $r|_{s(\mathcal{U})}$ is bijective and $r(s(\mathbf{u})) = \mathbf{u}$ for all $\mathbf{u} \in \mathcal{U}$, such that $r(\mathbf{v}) = \mathbf{u}$. Essentially, r is a representation map of the set $\mathcal{U}$, and element $\mathbf{v} \in s(\mathcal{U}) \subseteq \mathcal{W}$ is a representation of $\mathbf{u}$ where $r(\mathbf{v}) = \mathbf{u}$ with $\mathbf{u} \in \mathcal{U}$. The choice of a representation for a class of matrices depends on the stability of the representation and how many parameters are intrinsically needed to represent the matrices [16, 38].

The first representation to look at is the generator representation. Let's begin with the representation, $\mathcal{R}$, for a semiseparable matrix A which has the mapping

$$r|_A : \mathcal{R} \subset \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \to A \qquad \text{and} \qquad (\mathbf{u}, \mathbf{v}, \mathbf{p}, \mathbf{q}) \mapsto \operatorname{tril}(\mathbf{u}\mathbf{v}^{\mathrm{T}}) + \operatorname{triu}(\mathbf{p}\mathbf{q}^{\mathrm{T}})$$

where $\mathcal{R}$ is the set of 4-tuples of the form $\mathcal{R} = (\mathbf{u}, \mathbf{v}, \mathbf{p}, \mathbf{q}) \in \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n|_{u_i v_i = p_i q_i}$, for $i = 1, \ldots, n$. In this example, $s : \mathcal{U} \to \mathcal{V}$, $r : \mathcal{V} \to \mathcal{U}$,

$$\mathcal{U} = \left\{ A \in \mathbb{R}^{n \times n} |\ \exists\ \mathbf{u}, \mathbf{v}, \mathbf{p}, \mathbf{q} \in \mathbb{R}^n \text{ such that } \ a_{i,j} = \left\{ \begin{array}{ll} u_i v_j, & i \geq j, \\ p_i q_j, & i < j. \end{array} \right\} \right\} \quad \text{and}$$

$$\mathcal{V} = \{(\mathbf{u}, \mathbf{v}, \mathbf{p}, \mathbf{q}) \in \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n | \text{ such that } \ u_i v_i = p_i q_i \text{ for } i = 1, \ldots, n\}.$$

From [16], the representation of a semiseparable matrix in which the lower rank structure extends to the diagonal will look like:

$$A = \begin{bmatrix} u_1 v_1 & p_2 q_1 & p_3 p_1 & \cdots & p_n q_1 \\ u_2 v_1 & u_2 v_2 & p_3 p_2 & \cdots & p_n q_2 \\ u_3 v_1 & u_3 v_2 & u_3 v_3 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & p_n q_{n-1} \\ u_n v_1 & u_n v_2 & u_n v_3 & \cdots & u_n v_n \end{bmatrix}.$$

This particular example is not general enough to parameterize quasiseparable matrices.

Quasiseparable matrices have the property that their off-diagonal blocks have low rank with the provision the off-diagonal blocks do not intersect the diagonal. The generator representation of a quasiseparable matrix A has vectors that generate the lower and upper off-diagonal elements of A, and a vector that produces the elements on the diagonal [44]. An example of a generator representation of a quasiseparable matrix is

$$\mathcal{A} = \left\{ A \in \mathbb{R}^{n\times n} | \ \exists \ \mathbf{u}, \mathbf{v}, \mathbf{p}, \mathbf{q}, \mathbf{d} \in \mathbb{R}^n \text{ such that } \ a_{i,j} = \left\{ \begin{array}{ll} u_i v_j, & i > j, \\ d_i, & i = j, \\ p_i q_j, & i < j \end{array} \right\} \right\}$$

where i and $j = 1, \ldots, n$. Some quasiseparable matrices lack a generator representation, and this can be addressed with a pair of inverse bidiagonal factors. In the above example, if A is amended where $n = 5$, the generators $\mathbf{u}, \mathbf{v}, \mathbf{p}, \mathbf{q}, \mathbf{d} \in \mathbb{R}^n$, and the inverse bidiagonals $\mathbf{t}, \mathbf{r} \in \mathbb{R}^{n-1}$ [44], then the generator representationof the quasiseparable matrix is

$$\mathcal{A} = \begin{bmatrix} d_1 & p_1 r_1 q_2 & p_1 r_1 r_2 q_3 & p_1 r_1 r_2 r_3 q_4 & p_1 r_1 r_2 r_3 r_4 q_5 \\ u_2 t_1 v_1 & d_2 & p_2 r_2 q_3 & p_2 r_2 r_3 q_4 & p_2 r_2 r_3 r_4 q_5 \\ u_3 t_2 t_1 v_1 & u_3 t_2 v_2 & d_3 & p_3 r_3 q_4 & p_3 r_3 r_4 q_5 \\ u_4 t_3 t_2 t_1 v_1 & u_4 t_3 t_2 v_2 & u_4 t_3 v_3 & d_4 & p_4 r_4 q_5 \\ u_5 t_4 t_3 t_2 t_1 v_1 & u_5 t_4 t_3 t_2 v_2 & u_5 t_4 t_3 v_3 & u_5 t_4 v_4 & d_5 \end{bmatrix}.$$

Regrettably, it must be noted that some particular sets of generators for quasiseparable matrices display instability in matrix-vector multiplication which is not the fault of the algorithm.

The class of $\mathscr{H}$-matrices and hierarchical representations of rank structured matrices are examined. These data sparse parameterizations are used to represent the matrix, its inverse and decompositions of the matrix. With these particular matrices, they are characterized by a hierarchy in the low-rank off-diagonal blocks. Thus the matrix is hierarchically partitioned into blocks of low rank at multiple levels of the hierarchy. The partitioning of these blocks

continues recursively until the lowest level diagonal block is reached. The hierarchical representation and associated operations enable the reuse of data, and the sharing of information across different levels, which can be extremely efficient. An example of such a matrix is shown in Figure 1.3. Algorithms for rank structured matrices frequently make use of a hierarchical block representation [17–19, 45–47]. The fast multipole literature [28, 29, 48] uses binary tree structures and parameterizations for the hierarchical representation of rank structured matrices. The research in this dissertation also utilizes the binary tree to navigate the hierarchy and parameterizations for the hierarchical representation. This will be discussed in Subsection 2.5.2 and Chapter 3.

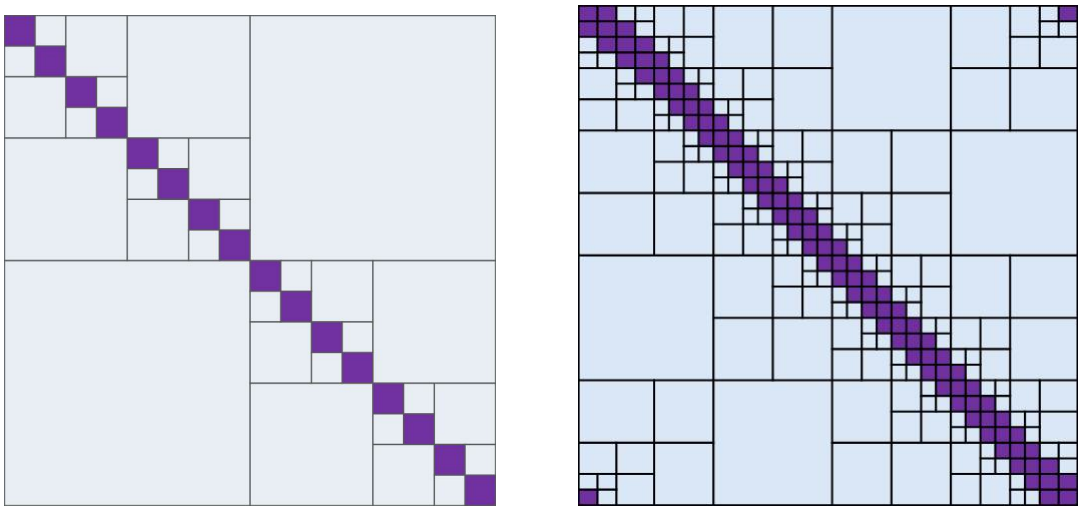


Figure 1.3. The hierarchical representations of the matrices display the full-rank and low-rank blocks. The submatrices in dark color are full rank. All other submatrices have low numerical rank.

The nested product representation for the class of quasiseparable matrices is a series of multiplications and subtractions that are nested within each other [3]. Given a quasiseparable matrix H, the nested product representation multiplies the left and right hand sides of H by orthogonal transformations, then zeros out the first of column of H with matrix subtraction. The matrix can now be rewritten in terms of the sequence of nested orthogonal

transformations and matrix addition. The process is repeated until the terminating matrix H_k is sufficiently small. Therefore, a matrix with band structure in its lower triangular part and rank structure in its upper triangular part when the nested product is applied gives

$$H_0 = U_1 \left(L_1 + U_2 \left(L_2 + U_3 \left(L_3 + \ldots U_k \left(L_k + \begin{bmatrix} 0 & 0 \\ 0 & H_k \end{bmatrix} \right) V_k^{\mathrm{T}} \ldots \right) V_3^{\mathrm{T}} \right) V_2^{\mathrm{T}} \right) V_1^{\mathrm{T}}. \tag{1.9}$$

It is important to note that research groups working with low rank structured matrices favor a particular definition, parameterization, and representation of these matrices. Comparisons of results are difficult between the different groups due to the diverse parameterizations and representations for the same matrix.

1.5 Fast Solvers

A fundamental problem in numerical linear algebra is solving linear systems. There are four major factors in choosing a linear system solver: preservation of structure in the matrix, stability, accuracy, and computational costs. Traditional Gaussian elimination without pivoting is, in general, not stable. However, Gaussian elimination with partial pivoting is known as the standard backward stable solver [37]. Unfortunately, these pivoting techniques in Gaussian elimination fail to preserve the structure of quasiseparable matrices. The unpivoted LU factorization method preserves the data sparsity of rank structured matrices, but is based on Gaussian elimination and has stability issues [37, 49]. Table 1.1 summarizes the traditional algorithms, and how they perform with respect to stability and data sparsity.

There is a group of solvers which keep the data sparse structure of a quasiseparable matrix intact. Within this group, the focus now narrows towards stability and accuracy [37]. One particular type algorithm rises to the surface where it preserves the sparsity in the rank structure, is backward stable, and is accurate. The $QR-$based system

Table 1.1. Algorithm Comparison for Rank Structured Matrices

Traditional Algorithm	Matrix Structure	Stability
Gaussian elimination without pivoting	Preserves	Unstable
Gaussian elimination with partial pivoting	Destroys	Maybe Stable
SVD decomposition	Destroys	Stable
QR factorization	Preserves	Stable

solvers, for some classes of rank structured matrices, have shown promise [47]. There are some very fast $QR-$based algorithms for solving structured matrices. The current $QR-$based system solvers for structured matrices, that address some form of hierarchical representation, perform a decomposition implicitly or explicitly, where the decomposition is comprised of orthogonal matrices, and a lower or upper triangular matrix [50]. Block solvers focus on solving matrix representations such as sequentially semiseparable (SSS) and hierarchically semiseparable (HSS). However, this comes at an additional cost in computing the parameterization of matrix which varies from $O(nr)$ to $O(rn^2)$ where n is the size of a square matrix and r is the rank of the low-rank blocks. Given a hierarchically semiseparable representation of structured matrix A, the solvers in [13, 14, 18] do orthogonal eliminations on both sides of A to transform A into a lower triangular matrix. The resulting matrix, being lower triangular, can be solved directly through substitution. These block solvers computation costs range from $O(rn\log^2(n))$ to $O(n^2)$. It is important to note that even though the parameterization and block solver algorithms state they are stable, none of them have been proven stable.

Algorithms to solve quasiseparable systems use factorizations, such as the LU or ULV [20]. In [18], an implicit ULV decomposition of A is computed, where U and V are orthogonal matrices, and L is a lower-triangular matrix. The algorithm operates on low-rank blocks sequentially to solve recursively for $\mathbf{x}$. The fast ULV solver algorithm can be adapted to operate on a generator representation of a quasiseparable matrix where the triangular parts of the rank structured matrix A are decomposed as nested products of sums. The

computational complexity of the ULV solver algorithms ranges from solvers computation costs range from $O(nk^2)$ to $O(n^2)$ where n is the size of a square matrix and k is the size of a submatrix. Again, none of these algorithms have been proven stable.

The fast nested product solver presented by Bella, Olshevsky and Stewart operates on a quasiseparable matrix A to avoid the generator representation, and decomposes A into a nested UBV product. The problem solved in [3] is $A\mathbf{x} = \mathbf{b}$ where A is a quasiseparable matrix. Matrix A is first row compressed, $A = QC$ or $Q^{\mathrm{T}}A = C$ where C has band structure in the lower triangular part and rank structure in the upper triangular part. What follows is $Q^{\mathrm{T}}A\mathbf{x} = Q^{\mathrm{T}}\mathbf{b}$ becomes $C\mathbf{x} = Q^{\mathrm{T}}\mathbf{b}$. Matrix C is then decomposed by the UBV algorithm, and the fast nested product solver operates on $C\mathbf{x} = Q^{\mathrm{T}}\mathbf{b}$. The solver in [3] is the only *proven stable* solver for these classes of matrices with computational complexity of $O(n)$.

1.6 Problem Statement and Contributions

Rank structured matrices have arisen in an assorted variety of applications from image processing to electromagnetics, and their importance has intensified in recent years. The recent activity surrounding quasiseparable matrices is their emergence as a new superclass for the structured matrix class [47]. As a result, investigating different representations, conversions, decompositions, computations, and solvers for quasiseparable matrices is taking on more importance in numerical linear algebra. There still isn't much consensus on terminology even though some algorithms have been developed more than 25 years ago. Much early research was done on algorithms and solvers for the semiseparable matrix class which is a subclass of the quasiseparable matrix class [51]. Representations of rank structured matrices are as diverse as their applications with representations ranging from unitary-weight, to generator, to hierarchical [20, 52, 53]. A number of the existing algorithms are centered on the QR factorization techniques, and generator representations. The nested product representation and the accompanying solver algorithm for quasiseparable matrices introduced in [3] are proven stable with the solver computational cost being $O(n)$.

The $\mathscr{H}$ matrix and corresponding hierarchical representations can be used to characterize quasiseparable matrices. Numerical backward stability in existing decompositions and fast solver algorithms other than [3] is a major sticking point.

There is a substantial gap in the research on representations of rank structured matrices, and the conversion between representations for quasiseparable matrices. Currently, there is only one solver proven to be stable, and the solver operates on the nested product representation for quasiseparable matrices. There is no specific conversion for a quasiseparable matrix from a hierarchical representation to a nested product representation. In image processing, iterative methods for deblurring are still preferred so preconditioners are necessary in these methods [34, 54]. Most methods use two-dimensional (2-D) circulant preconditioners for Toeplitz systems where there is much work done in clustering of eigenvalues. Wavelets are used to compress images, and wavelets have a hierarchical structure. If a hierarchical representation of the wavelet is compressed, then the image can be compressed even further. The research done by Bella, Olshevsky and Stewart introduces a new fast stable solver based on the nested product representation of the class of quasiseparable matrices, and, for some problems, can allow us to solve the deblur problem stably circumventing the costly iterative methods.

The research proposed in this dissertation extends the nested product decomposition and solver from [3] by converting a hierarchical representation of a quasiseparable matrix to a nested product. We present a new hierarchical parameterization algorithm for quasiseparable matrices. A major contribution of this work is the row compression of the hierarchical structure which is then prepared for conversion to another representation. We introduce a new conversion algorithm transforming a hierarchical representation of a quasiseparable matrix into a nested product representation. The key component in both the compression and conversion algorithms is the Householder transformation, which results in the algorithms having backward stability. With a full nested product complete, the matrix-vector product and fast solver in [3] can be used to stably solve large systems with $O(n)$ cost. Quasiseparable

matrices, combined with an already computed hierarchical representation, can now have access to a fast stable solver via our parameterization, row compression, and nested product conversion. The algorithms of this study are applied to image compression problems that use wavelets, and image restoration problems that involve of atmospheric turbulence of satellite images using preconditioners.

The remainder of the dissertation is organized as follows: In Chapter 2, related works are surveyed. The new algorithms for the parameterization and row compression of a hierarchical representation of a quasiseparable matrix are detailed in Chapter 3. Chapter 4 lays out the conversion of the hierarchical representation to the nested product representation found in [3]. The matrix-vector multiplication and fast solver from [3], is discussed in Chapter 5. Chapter 6 presents an application of the algorithms in image deblurring. Chapter 7 presents an application of the algorithms in image compression. Comparisons, complexity, conclusions and future work with respect to the row compression and nested product conversion are presented in Chapter 8.

CHAPTER 2

RELATED WORK

Rank structured matrices are matrices for which certain blocks have ranks that are bounded by a small constant. In Chapter 1, the rank structured matrix class was introduced and a more formal definition was given. A diagram of the rank structured matrix class in Figure 1.2 illustrates the relationships of quasiseparable matrices and its subclasses [2]. Quasiseparable matrices have blocks strictly above or strictly below the diagonal that have bounded rank, and have emerged as a tractable class for development of fast algorithms. Quasiseparable matrices are cross disciplinary in their uses with applications throughout applied mathematics, engineering and computer science. Hence, it is not surprising that quasiseparable matrices, as well as operations to manipulate them, are receiving a great deal of attention from several groups in Belgium (Dewilde, Van Barel et al.) [4, 5, 15, 16, 55], Israel (Eidelman, Gohberg) [6, 53, 56], Italy (Bini, Gemignani, Mastronardi) [30, 39, 57], the USA (Chandrasekaran, Gu, Olshevsky, Stewart) [3, 18], etc. Hierarchical matrices form a hierarchy of their submatrices from a rank structured matrix, and play an important role in multipole methods and electromagnetic [8, 10, 11, 23, 28, 29, 48, 58]. Research on $\mathscr{H}$-matrices is taking place in the USA (Greengard, Rokhlin, Strain et al.), and in Germany (Hackbusch, Börm, Grasedyck, Khoromskij).

There has been confusion among groups on the nomenclature for rank structured matrices. The book and papers by Dewilde, van der Veen, and Alijagic [4, 5, 7] treat quasiseparable systems, which they refer to as systems with low Hankel rank, using an approach based on linear systems theory. Their work was the first to begin closing the gap between function theory and numerical linear algebra where, instead of operating on matrices of scalars or variables, the theories operated on vector representations. Dewilde and van der

Veen developed several classical results on algorithms for time-varying systems. Especially, inner-outer factorization as a generalization of the QR factorization.

The work by Van Barel et al. discusses various rank structured matrix types such as tridiagonal and semiseparable matrices in full generality which are subclasses of the larger quasiseparable matrix class [15, 55]. They carry out efficient matrix operations on numerical examples of subclasses of the quasiseparable matrix class. The books by Vandebril et al. examine computations and algorithms for semiseparable matrices and their representations [16, 38]. The Vandebril representations are centered on the Givens vector and the use of rotations in the construction of the different representations. Research by the Chandrasekaran group explores the sequential representations of semiseparable matrices, and uses generators in forming their representations [13, 14]. In all three bodies of work, the derivation of algorithms for rank structured matrices capitalized on the block quasiseparable or sequentially semiseparable (SSS) representations. Definitions for quasiseparable matrices are given, and by definition does contain the semiseparable matrix class. The research in this dissertation is on hierarchical representations of quasiseparable matrices (and not SSS), performs a factorization of blocks and not generators for the elements, and uses Householder transformations in the row compression and nested product algorithms (and not Givens rotations).

The hierarchical matrix class, the $\mathscr{H}-$matrix class, is the class of dense matrices with the characteristic of being represented by only a few parameters and is therefore considered data-sparse. Matrices in the FMM, or that are used in elliptic boundary value problems, have data clusters around the diagonal, and a hierarchy of the submatrices can be constructed from the matrix [8, 23, 28, 48, 58]. The representation of an $\mathscr{H}-$matrix requires only in $O(n\log(n))$ computation, and the algorithms that operate on the new truncated $\mathscr{H}-$matrix format do so in $O(n\log(n))$ operations.

The quasiseparable matrix definition used in this research comes from the work by Eidelman and Gohberg, and is similar to the definition found in these papers [3, 43, 47, 49,

59]. Eidelman and Gohberg reinterpreted the algorithm of Dewilde and van der Veen in linear algebraic terms. The purpose of this dissertation is to take a quasiseparable matrix already partitioned into a hierarchical representation and transform it into a nested product representation, then solve the system.

2.1 Representations of Quasiseparable Matrices

There are three parameterizations of a quasiseparable matrix that are involved in this dissertation: generator, hierarchical, and nested product. Many algorithms in numerical linear algebra have been modified to work with the generator representation of a quasiseparable matrix instead of the elements within the matrix [4–6, 13]. The generator representation is absent as a subject of research in this work, but is surveyed as background. All three representations can represent a quasiseparable matrix with $O(n)$ parameters.

2.1.1 Generator Representation

The work proposed in this dissertation transforms a hierarchical representation into a nested product representation which is based on generators. The generator representation is one of the first representations for quasiseparable matrices, and standard numerical linear algebra algorithms have been amended to operate on generators instead of matrix elements. The generator representation presented as background in Bella, Olshevsky and Stewart [3] takes the upper and lower triangular parts of the quasiseparable matrix, and applies products of generators to form the elements. Given an $n \times n$ quasiseparable matrix A with 2×2 block partitionings of the form

$$A = \begin{bmatrix} A_{11}^{(j)} & A_{12}^{(j)} \\ A_{21}^{(j)} & A_{22}^{(j)} \end{bmatrix} \tag{2.1}$$

where A_{11} is $j \times j$ for $j = 1, 2, \ldots, n-1$. The *lower rank sequence* of A is $r_j(A) = \text{rank}(A_{21}^{(j)}) \leq r$, and the *upper rank sequence* of A is $s_j(A) = \text{rank}(A_{12}^{(j)}) \leq s$. The parameterization of

quasiseparable matrix with generators is formed in the example below, where A is a 5×5 quasiseparable matrix,

$$A = \begin{bmatrix} d_1 & \mathbf{g}_1^{\mathrm{T}}\mathbf{h}_2 & \mathbf{g}_1^{\mathrm{T}}B_2\mathbf{h}_3 & \mathbf{g}_1^{\mathrm{T}}B_2B_3\mathbf{h}_4 & \mathbf{g}_1^{\mathrm{T}}B_2B_3B_4\mathbf{h}_5 \\ \mathbf{p}_2^{\mathrm{T}}\mathbf{q}_1 & d_2 & \mathbf{g}_2^{\mathrm{T}}\mathbf{h}_3 & \mathbf{g}_2^{\mathrm{T}}B_3\mathbf{h}_4 & \mathbf{g}_2^{\mathrm{T}}B_3B_4\mathbf{h}_5 \\ \mathbf{p}_3^{\mathrm{T}}A_2\mathbf{q}_1 & \mathbf{p}_3^{\mathrm{T}}\mathbf{q}_2 & d_3 & \mathbf{g}_3^{\mathrm{T}}\mathbf{h}_4 & \mathbf{g}_3^{\mathrm{T}}B_4\mathbf{h}_5 \\ \mathbf{p}_4^{\mathrm{T}}A_3A_2\mathbf{q}_1 & \mathbf{p}_4^{\mathrm{T}}A_3\mathbf{q}_2 & \mathbf{p}_4^{\mathrm{T}}\mathbf{q}_3 & d_4 & \mathbf{g}_4^{\mathrm{T}}\mathbf{h}_5 \\ \mathbf{p}_5^{\mathrm{T}}A_4A_3A_2\mathbf{q}_1 & \mathbf{p}_5^{\mathrm{T}}A_4A_3\mathbf{q}_2 & \mathbf{p}_5^{\mathrm{T}}A_4\mathbf{q}_3 & \mathbf{p}_5^{\mathrm{T}}\mathbf{q}_4 & d_5 \end{bmatrix} \tag{2.2}$$

where $A_j \in \mathbb{R}^{r_j(A)\times r_{j-1}(A)}$, $B_j \in \mathbb{R}^{s_j(A)\times s_{j-1}(A)}$, $d_j \in \mathbb{R}$, $\mathbf{p}_j \in \mathbb{R}^{r_{j-1}(A)}$, $\mathbf{q}_j \in \mathbb{R}^{r_j(A)}$, $\mathbf{g}_j \in \mathbb{R}^{s_j(A)}$, and $\mathbf{h}_j \in \mathbb{R}^{s_{j-1}(A)}$ [3].

When examining the off-diagonal subblocks of the generator representation of A,

$$M = \begin{bmatrix} \mathbf{p}_3^{\mathrm{T}}A_2\mathbf{q}_1 & \mathbf{p}_3^{\mathrm{T}}\mathbf{q}_2 \\ \mathbf{p}_4^{\mathrm{T}}A_3A_2\mathbf{q}_1 & \mathbf{p}_4^{\mathrm{T}}A_3\mathbf{q}_2 \\ \mathbf{p}_5^{\mathrm{T}}A_4A_3A_2\mathbf{q}_1 & \mathbf{p}_5^{\mathrm{T}}A_4A_3\mathbf{q}_2 \end{bmatrix} \quad \text{and} \quad N = \begin{bmatrix} \mathbf{g}_1^{\mathrm{T}}B_2h_3 & \mathbf{g}_1^{\mathrm{T}}B_2B_3\mathbf{h}_4 & \mathbf{g}_1^{\mathrm{T}}B_2B_3B_4\mathbf{h}_5 \\ \mathbf{g}_2^{\mathrm{T}}\mathbf{h}_3 & \mathbf{g}_2^{\mathrm{T}}B_3\mathbf{h}_4 & \mathbf{g}_2^{\mathrm{T}}B_3B_4\mathbf{h}_5 \end{bmatrix},$$

the blocks can be rewritten as the following product,

$$M = \begin{bmatrix} p_3^{\mathrm{T}} \\ p_4^{\mathrm{T}}A_3 \\ p_5^{\mathrm{T}}A_4A_3 \end{bmatrix} \cdot \begin{bmatrix} A_2q_1 & q_2 \end{bmatrix} \quad \text{and} \quad N = \begin{bmatrix} g_1^{\mathrm{T}}B_2 \\ g_2 \end{bmatrix} \cdot \begin{bmatrix} h_3 & B_3h_4 & B_3B_4h_5 \end{bmatrix}.$$

This factorization shows M has rank at most $r_2(A)$ and N has rank at most $s_2(A)$. Similar generator parameterizations have been used in a series of papers by Chandrasekaran et al. [13, 14, 17–19]. It is important to point out a drawback to the generator representation. When small perturbations are introduced into some particular sets of generators of a quasiseparable matrix A and algorithms are applied to the generators, this leads to large perturbations of A, where the instability is caused by the generators. This can be a source of instability

in algorithms that use the generator representation. The cost of extracting a generator representation of the matrix is $O(n^2)$ [4, 5].

2.1.2 Nested Product Representation

The paper by Bella, Olshevsky and Stewart [3] introduced a new parameterization for quasiseparable matrices, the nested product representation. The algorithm given in [3] computes a nested UBV decomposition. Given a matrix C with a banded structure in the lower triangular part, and a quasiseparable structure in the upper triangular part. The UBV decomposition of C is

$$C = U_{k_0}\left(B_{k_0} + U_{k_1}\left(B_{k_1} + \ldots + U_{k_{p-1}}\left(B_{k_{p-1}} + D_{k_p}\right)\ldots\right)\right)V_{k_{p-1}}^{\mathrm{T}}V_{k_1}^{\mathrm{T}}V_{k_0}^{\mathrm{T}} \tag{2.3}$$

where $0 = k_0 < k_1 < \ldots < k_p$ is an increasing sequence, both U_{k_j} and V_{k_j} are sequences of orthogonal transformations for $j = 0, \ldots, p-1$, and B_{k_j} is a sequence of banded lower triangular matrices with only a few nonzero columns. The form of matrix D_{k_p} is

$$D_{k_p} = \begin{bmatrix} 0_{k_p \times k_p} & 0 \\ 0 & D_{k_p,22}^{(k_p)} \end{bmatrix}$$

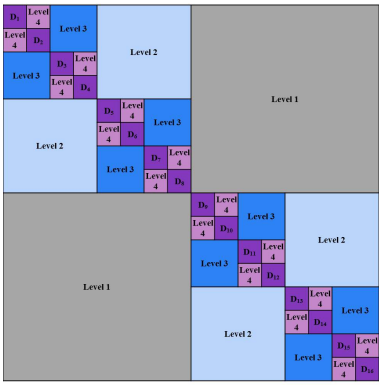
such that the superscript (k_p) indicates the size of the leading principal submatrix, and the small size of $D_{k_p,22}^{(k_p)}$ makes exploiting the rank structure in this matrix is unnecessary. The pattern of two-sided unitary transformations in (2.3) was first used to take advantage of rank structured matrices in a succession of papers [13, 18, 60]. If A has rank structure in both its lower and upper triangular parts, a row compression can be used to introduce a band structure in the lower triangular part of A. The decomposition in [3] has the row compression introducing a band structure in the lower triangular part of A which results in a sparse orthogonal decomposition of A. The nested UBV decomposition is stable, and uses $O(n^2)$ operations on a general rank structured matrix, which is comparable to the generator representation of a quasiseparable matrix.

The similarity between [3] and our work is that both begin with a quasiseparable matrix A and use row compression methods to form $A = QC$. Our work starts with a hierarchical representation of A, then, via hierarchical row compression and nested product algorithms, transforms A to C and C into UBV decomposition. Whereas in [3], the row compression forms the nested product representation directly from A. The hierarchical to nested product conversion enables use of the fast solver and matrix-vector product algorithms presented in [3].

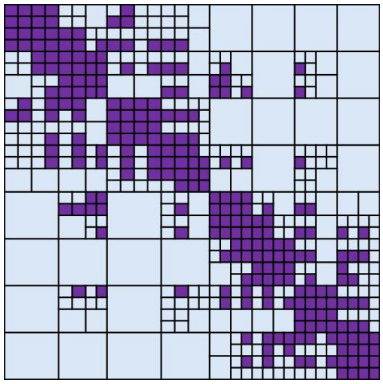
2.1.3 Hierarchical Representation

Hierarchical matrices have been studied explicitly since the works of Hackbusch [8], Hackbusch and Khoromskij [22, 23], and Hackbusch, Grasedyck and Börm [9] on $\mathscr{H}$-matrices. The fast multipole method (FMM) of Carrier, Greengard, and Rokhlin [48], and Greengard and Rokhlin [28], used ideas related to rank structure to speed up the processing of computationally intensive problems. However, FMM literature is not concerned with matrices, but with the evaluation of functions The hierarchical representation is created from a superimposed mesh on a function, and forms partitions which range from simple to complex as seen in Figure 2.1. The dark blocks represent full rank blocks that might admit further subdivision. the light blocks represent blocks with bounded (low) rank.

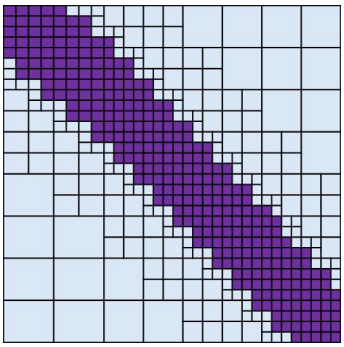
A hierarchical representation of a quasiseparable matrix is a recursive low rank factoring and partitioning of the blocks. The algorithm for partitioning SSS matrices in [17–19] directly relates to the work in this dissertation, and has been adapted for the quasiseparable matrices we consider in our research. The SSS and HSS matrices considered in the works by the Chandrasekaran group are different representations of the quasiseparable matrices used in our research. As the matrix is partitioned into blocks at multiple levels, the blocks are described by a formula defined by an associated binary tree. This enables reuse and sharing of information across different levels to achieve high efficiency. The cost to compute a hierarchical representation of a general rank structured matrix is $O(n^2)$ [18]. Table 2.1 shows the computational complexity for extracting the parameterization from an $n \times n$



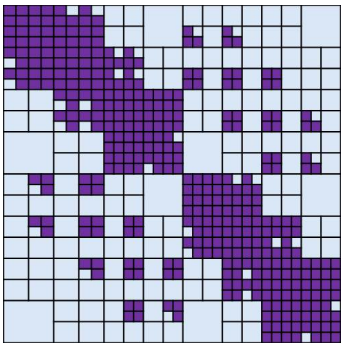
(a) Partitioning done in this research denoting levels in hierarchy.



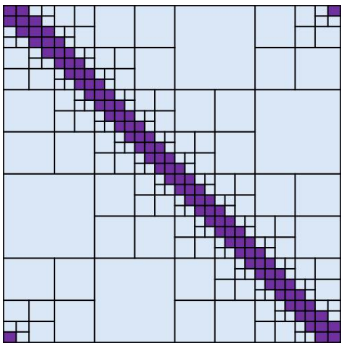
(b) Partitions with boundary concentrations [10].



(c) Partitions with edge concentrations [10].



(d) Partitions with uniform concentrations [10].



(e) Partitions with dyadic clustering [9].

Figure 2.1. Examples of hierarchical block partitioning where the dark (purple) blocks are full rank and all other blocks have low numerical rank. Figures 2.1(b) 2.1(c) and 2.1(d) are from [10], and Figure 2.1(e) is from [9].

matrix represented by its elements. In the presence of sparsity or other structure, faster algorithms are possible.

The conversion of a hierarchical representation to a nested product representation requires that the hierarchical representation have a factored blocks that can be compressed to form a nested product representation. Hence, the factorization of the low rank off-diagonal blocks in the quasiseparable matrix is included in the partition algorithm. The factorization in this research follows similar procedures as in [18]. As the matrix is partitioned

Table 2.1. Summary of Computational Complexity for Parameterizations

Parameterization	Computational Cost
Generator representation[4]	$O(n^2)$
Nested Product representation[3]	$O(n^2)$
Hierarchical representation[18]	$O(n^2)$

and factored, a binary tree is the natural choice for representing the partitions of the hierarchical representation. Sequentially and hierarchically semiseparable matrices are particular representations of a quasiseparable matrix, and contain generators for the elements. These representations have some similarities with the hierarchical representation presented in this dissertation; however, our hierarchical representation is a block factorization which is much more efficient.

2.2 Compression and Decomposition for Representation Conversion

The row compression algorithm presented in [3] compresses an $m \times n$ submatrix block C in a quasiseparable matrix A. The algorithm introduces zero rows into C by computing Householder transformations. The basic procedure for introducing zeros into a matrix is as follows. Let X be an $m \times d$ matrix with columns that form an orthonormal basis for the left nullspace of C. Zeros are introduced into the last d rows of C by computing Householder transformations Q_k where

$$Q_d^{\mathrm{H}} \dots Q_2^{\mathrm{H}} Q_1^{\mathrm{H}} X = \begin{bmatrix} 0 \\ I_d \end{bmatrix}$$

and

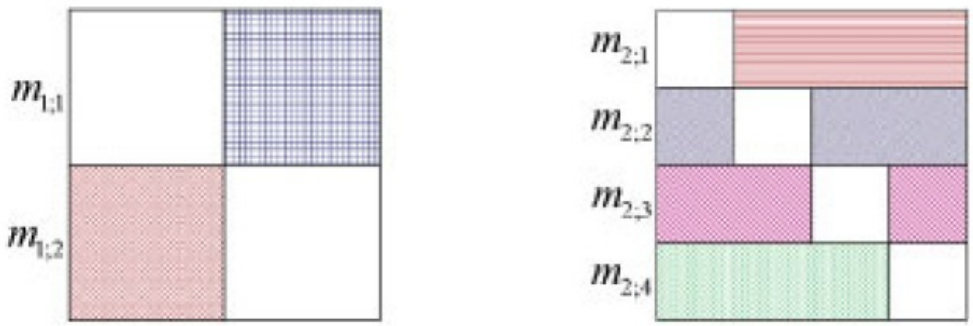
$$0 = X^{\mathrm{H}} C = \begin{bmatrix} 0 & I_d \end{bmatrix} Q_d^{\mathrm{H}} \dots Q_2^{\mathrm{H}} Q_1^{\mathrm{H}} C = \begin{bmatrix} 0 & I_d \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = C_2 = 0.$$

The computation of the nullspace X is accomplished using the SVD. The row compression of the hierarchical representation in this dissertation is similar to that in [3] by using

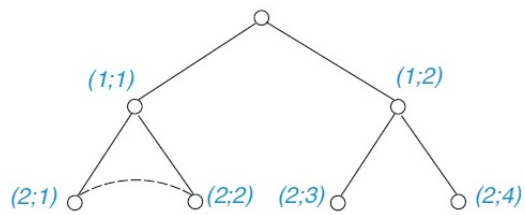
Householder transformations. However, our row compression algorithm operates on multiple block levels in the quasiseparable matrix, transforming the multiblock region into a low rank block columns space basis for compression.

One decomposition that this research can be compared with is the implicit ULV decomposition where U and V are orthogonal matrices, and L is a lower -triangular matrix. In Chandrasekaran, Gu and Pals [18] and Xia, Chandrasekaran, Gu and Li [21], both papers form the ULV decomposition of semiseparable matrices. The research presented in [18] is closely related to what is proposed in this dissertation where the ULV decomposition is formed from hierarchically semiseparable (HSS) dense matrices. Their factors are not computed and stored explicitly, U and V are represented as a product of elementary Gaussian transforms and permutation matrices. In [21], they take HSS matrices and implicitly form $ULV-$type factorizations where U and V are orthogonal matrices and L is lower triangular. Figure 2.2 shows the HSS matrix structure and corresponding binary tree. The UBV decomposition used in our research is on quasiseparable matrices, the factors are computed explicitly, and it becomes a nested product. The decomposition done in our research is to transform the hierarchical representation of quasiseparable matrices into a nested product, and the factors are computed explicitly.

The QR decomposition of quasiseparable matrices has become a building block in a lot of other decomposition algorithms, and is still being modified and improved. Eidelman and Gohberg [6] did a study on the class of block structured matrices analyzing QR factorization and inversion algorithms related to those from [4]. In [6], they modified and simplified existing algorithms to be more transparent. They give a VUS decomposition algorithm of generators where V is a block lower triangular orthogonal matrix, U is a block upper triangular orthogonal matrix, and S is a block upper triangular matrix with square invertible blocks on the diagonal. The VUS algorithm incorporates a matrix-vector product of triangular systems. The fast QR iteration method introduced by Eidelman, Gohberg and Olshevsky [56] exploits the structure of a Hermitian quasiseparable matrix. The algorithm



(a) Two levels of HSS off-diagonal blocks: left matrix is first-level, and right matrix is second-level.



(b) Binary tree of HSS showing two levels.

Figure 2.2. Examples of the HSS matrix and its corresponding tree representations, from [18].

operates on generators, a linear set of parameters, defining the quasiseparable matrix. The paper by Bini, Eidelman, Gemignani and Gohberg [39] shows a fast shifted QR decomposition to compute the eigenvalues of a Hessenberg matrix. The basic idea of the work in [39] is to find a compact representation of the matrices A_k for $k = 0, 1, ...,$ that are generated by the QR iteration. Benner and Mach [61] presented an efficient, recursive, block column wise QR decomposition of $\mathscr{H}$- matrices by implementing the standard $QR-$ factorization algorithm for dense blocks as often as possible. Although, the work done in [4, 6, 39, 56, 61] is on various $QR-$decomposition methods, it is relevant to ours since we are adapting QR factorization techniques in some parts of our row compression conversion algorithm.

There are a few algorithms for representation conversion of structured matrices. The Givens-weight representation by Delvaux and Van Barel [52] for rank structured

matrices is where representation conversion is mentioned. Obviously from the title, this representation uses Givens rotations to form a representation of a rank structured matrix. The research on representation conversion in [52], extends the work by Delvaux, Frederix and Van Barel [62], and illustrates a hierarchical to unitary weight representation conversion. The algorithm transforms an HSS matrix into a Hessenberg matrix via orthogonal transformations using QR factorization. The algorithm operates on the binary tree associated with the HSS, and converts the levels of the 2-D row tree via Givens rotations into a column tree while obtaining the elementary orthogonal operations for the unitary-weight representation. Our hierarchical representation conversion algorithm is more transparent and simple, it is stable, and of course converts a hierarchical representation to a nested product representation of a quasiseparable matrix.

2.3 Generator Representation and Matrix-Vector Products

In computer science and scientific computing, the product of a matrix and a vector is a fundamental operation. For matrix-vector multiplication, let $\mathbf{x} \in \mathbb{R}^n, A \in \mathbb{R}^{m\times n}$, then $\mathbf{b} = A\mathbf{x}$ is a linear combination of the columns of A. Thus, a formula for $\mathbf{b}$ with the $j-$th column of A can be written as $\mathbf{b} = \sum_{j=1}^{n} x_j \mathbf{a}_j$ and is interpreted as $\mathbf{x}$ acting on A to produce $\mathbf{b}$ [32]. The simplest example of exploiting the quasiseparable structure of a matrix is one with $O(n)$ operations. The key idea of matrix-vector products is to take a rank structured matrix A in Equation 2.1.1, and rewrite it as a sum, $A = L + D + U$, where

$$L = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ \mathbf{p}_2^{\mathrm{T}}\mathbf{q}_1 & 0 & 0 & 0 & 0 \\ \mathbf{p}_3^{\mathrm{T}}A_2\mathbf{q}_1 & \mathbf{p}_3^{\mathrm{T}}\mathbf{q}_2 & 0 & 0 & 0 \\ \mathbf{p}_4^{\mathrm{T}}A_3A_2\mathbf{q}_1 & \mathbf{p}_4^{\mathrm{T}}A_3\mathbf{q}_2 & \mathbf{p}_4^{\mathrm{T}}\mathbf{q}_3 & 0 & 0 \\ \mathbf{p}_5^{\mathrm{T}}A_4A_3A_2\mathbf{q}_1 & \mathbf{p}_5^{\mathrm{T}}A_4A_3\mathbf{q}_2 & \mathbf{p}_5^{\mathrm{T}}A_4\mathbf{q}_3 & \mathbf{p}_5^{\mathrm{T}}\mathbf{q}_4 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} d_1 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 & 0 \\ 0 & 0 & d_3 & 0 & 0 \\ 0 & 0 & 0 & d_4 & 0 \\ 0 & 0 & 0 & 0 & d_5 \end{bmatrix}$$

$$U = \begin{bmatrix} 0 & \mathbf{g}_1^{\mathrm{T}}\mathbf{h}_2 & \mathbf{g}_1^{\mathrm{T}}B_2h_3 & \mathbf{g}_1^{\mathrm{T}}B_2B_3\mathbf{h}_4 & \mathbf{g}_1^{\mathrm{T}}B_2B_3B_4\mathbf{h}_5 \\ 0 & 0 & \mathbf{g}_2^{\mathrm{T}}\underline{\mathrm{h}}_3 & \mathbf{g}_2^{\mathrm{T}}B_3h_4 & \mathbf{g}_2^{\mathrm{T}}B_3B_4\mathbf{h}_5 \\ 0 & 0 & 0 & \mathbf{g}_3^{\mathrm{T}}\mathbf{h}_4 & \mathbf{g}_3^{\mathrm{T}}B_4\mathbf{h}_5 \\ 0 & 0 & 0 & 0 & \mathbf{g}_4^{\mathrm{T}}\mathbf{h}_5 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Then when multiplying A by vector x, it becomes $A\mathbf{x} = (L + D + U)\mathbf{x} = L\mathbf{x} + D\mathbf{x} + U\mathbf{x}$. The matrix $D\mathbf{x}$ is straight forward, and the focus is on the matrix-vector products of the triangular parts $L\mathbf{x}$ and $U\mathbf{x}$ to reformulate them as a nested products of sums. The matrix-vector product algorithm works with the equation $\mathbf{y} = L\mathbf{x}$ and the algorithm is shown in the 5×5 example below

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_2^{\mathrm{T}}\mathbf{q}_1 & 0 & 0 & 0 \\ \mathbf{p}_3^{\mathrm{T}}A_2\mathbf{q}_1 & \mathbf{p}_3^{\mathrm{T}}\mathbf{q}_2 & 0 & 0 \\ \mathbf{p}_4^{\mathrm{T}}A_3A_2\mathbf{q}_1 & \mathbf{p}_4^{\mathrm{T}}A_3\mathbf{q}_2 & \mathbf{p}_4^{\mathrm{T}}\mathbf{q}_3 & 0 \\ \mathbf{p}_5^{\mathrm{T}}A_4A_3A_2\mathbf{q}_1 & \mathbf{p}_5^{\mathrm{T}}A_4A_3\mathbf{q}_2 & \mathbf{p}_5^{\mathrm{T}}A_4\mathbf{q}_3 & p_5^{\mathrm{T}}\mathbf{q}_4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}. \tag{2.4}$$

The matrix-vector algorithm rewrites $\mathbf{y} = L\mathbf{x}$ as a nested products of sums with the initial values of the first equation are $y_1 = 0, \mathbf{z}_1 = 0, k = 2, \ldots, 5$ such that $\mathbf{z}_k = A_{k-1}\mathbf{z}_{k-1} + \mathbf{q}_{k-1}x_{k-1}$ and $y_k = \mathbf{p}_k\mathbf{z}_k$. Hence, the algorithm appears in the sequence of equations below,

$$\mathbf{z}_2 = A_1\mathbf{z}_1 + \mathbf{q}_1x_1 = A_1 \cdot 0 + \mathbf{q}_1x_1,\ y_2 = \mathbf{p}_2\mathbf{z}_2 = \mathbf{p}_2\mathbf{q}_1x_1$$

$$\mathbf{z}_3 = A_2\mathbf{z}_2 + \mathbf{q}_2x_2 = A_2 \cdot \mathbf{q}_1x_1 + \mathbf{q}_2x_2,\ y_3 = \mathbf{p}_3\mathbf{z}_3 = \mathbf{p}_3\left(A_2 \cdot \mathbf{q}_1x_1 + \mathbf{q}_2x_2\right)$$

$$\mathbf{z}_4 = A_3\mathbf{z}_3 + \mathbf{q}_3x_3 = A_3\left(A_2 \cdot \mathbf{q}_1x_1 + \mathbf{q}_2x_2\right) + \mathbf{q}_3x_3,\ y_4 = \mathbf{p}_4\mathbf{z}_4 = \mathbf{p}_4\left(A_3\left(A_2 \cdot \mathbf{q}_1x_1 + \mathbf{q}_2x_2\right) + \mathbf{q}_3x_3\right)$$

$$\mathbf{z}_5 = A_4\mathbf{z}_4 + \mathbf{q}_4x_4 = A_4\left(A_3\left(A_2 \cdot \mathbf{q}_1x_1 + \mathbf{q}_2x_2\right) + \mathbf{q}_3x_3\right),$$

$$y_5 = \mathbf{p}_5\mathbf{z}_5 = \mathbf{p}_5\left(A_4\left(A_3\left(A_2 \cdot \mathbf{q}_1x_1 + \mathbf{q}_2x_2\right) + \mathbf{q}_3x_3\right)\right).$$

The matrix-vector product algorithm in Chandrasekaran, Gu and Pals [18] uses a HSS matrix A where the product is of the form $\mathbf{z} = A\mathbf{b}$. In Figure 2.3, block row partitioning is denoted by $\mathbf{b}_{k,i}$ where the rows whose indices of $\mathbf{b}_{k,i}$ belong to $\text{Node}(k, i)$. Each node stores the factorization of the off-diagonal block. The multiplication of $U_{1;1}B_{1;1,2}V_{1;2}^{\text{H}}\mathbf{b}_{1;2}$ leads to the matrix-vector product

$$\mathbf{z}_{1;1} = D_{1;1}\mathbf{b}_{1;1} + U_{1;1}B_{1;1,2}V_{1;2}^{\text{H}}\mathbf{b}_{1;2} + R_{1;1}$$

where $\mathbf{b}_{k;i}, \mathbf{b}_{k;i+1}$ are vectors, $D_{k;i}$ is the diagonal block, $U_{k;i}, V_{k;i+1}$ are unitary matrices from the decomposition, $B_{k;i}$ is the matrix of singular values, and $R_{k;i}$ is the carry over. Refer to Figure 2.3 to see how these components appear in the tree. It is observed that the intermediate quantities of the computation at higher levels of the HSS can be computed recursively [18].

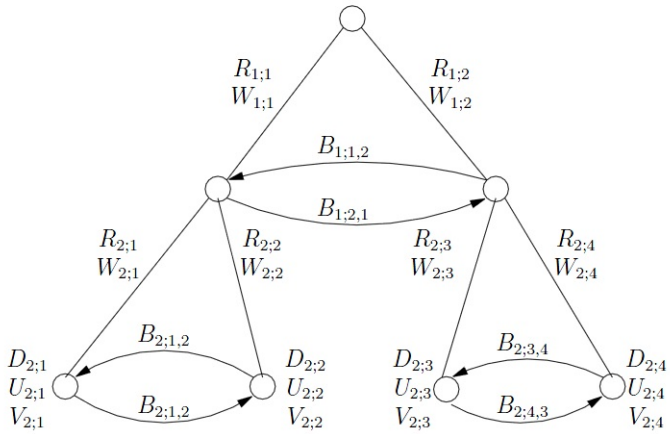


Figure 2.3. Three-level HSS representation on a binary tree displaying components used in matrix-vector products, from [18].

The matrix-vector product presented in Delvaux, Frederix and Van Barel [62] uses a column tree and row tree to access the components of a hierarchically rank structured matrix

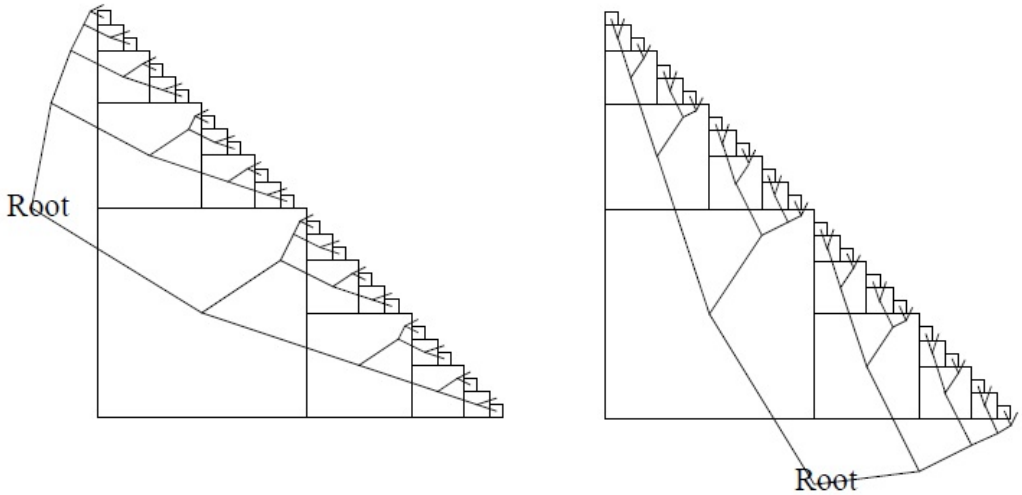


Figure 2.4. Hierarchically semiseparable (HSS) structure with underlying 2-D row tree, and column tree, from [62].

for the computation which is seen in Figure 2.4. Given the multiplication $\mathbf{y} = H\mathbf{x}$. At node k, $\mathbf{x}_k$ denotes the part of vector $\mathbf{x}$ corresponding to the indices of the vertical shaft. Similarly, $\mathbf{y}_k$ is the part of the matrix-vector product $\mathbf{y}$ that corresponds to the indices of the horizontal shaft. The first phase computes the matrix-vector product for the column tree $\mathbf{w}_k = B_k V_k \mathbf{x}_k$ for each node k. The second phase computes matrix-vector product for the leaves of the row tree $\mathbf{z}_k = \mathbf{z}_k + \mathbf{z}_{k-1}$ and $\mathbf{y}_k = U_k \mathbf{z}_k$. Thus the full matrix-vector product $\mathbf{y} = H\mathbf{x}$ is complete at the end of the second phase.

The matrix-vector product used in our research comes directly from the one given in [3]. It is designed to work specifically with a nested product representation of a quasiseparable matrix. Thus, the reasoning behind the conversion of representation from hierarchical to nested product. This allows the row compression $A = QC$ from our work to be used in the [3] matrix-vector product algorithm. The matrix-vector product algorithm operates on the equation $\mathbf{y} = A\mathbf{x} = QC\mathbf{x}$. More in depth information on the algorithm is shown in Chapter 5.

2.4 Fast System Solvers

The nested product solver introduced by Bella, Olshevsky and Stewart is the only proven stable solver, and requires $O(n)$ operations. Accessing this particular solver and using it to solve a hierarchical representation of a system is one of the goals for this research. The nested product solver adaptation and implementation for a hierarchical representation of a quasiseparable matrix is mentioned later in the section and detailed in Chapter 5. Other solvers that are useful to the research in this dissertation have some aspect in common with the hierarchical representation of a quasiseparable matrix. For the most part, the solvers surveyed are designed for hierarchical, HSS, or SSS matrices, with one on rank structured matrices and another for Vandermonde matrices.

The class of $\mathscr{H}$-matrices were introduced by Greengard and Rohklin [28] in their work on FMM. Fast solvers for $\mathscr{H}$-matrices were explored by Hackbusch in [22]. The matrix-vector multiplication algorithm for rank r $\mathscr{H}$-matrices has linear-logarithmic complexity. The solver operates on the block partitionings and achieves a complexity of $O(p^2n)$ where p is the number of partitions and n is the size of the matrix. The procedure computes an inverse approximation using LU factorization, and recursively iterates through each partition. Börm refines this solver in paper [63].

The SSS solver introduced by Chandrasekaran, Dewilde, Gu, Pals and van der Veen [13] is a fast backward stable algorithm solving $AX = B$ where A and B are given in SSS form. The "one-pass and top-down" algorithm carries out orthogonal eliminations to both sides of A by computing QL and LQ factorizations. The algorithm in [13] finds the SSS form of X thereby transforming the unknowns in X. The SSS solver algorithm follows the sequence of blocks and operates on a single block at a time. The solver disclosed in Chandrasekaran, Gu and Pals [18] is connected to our solver in that the hierarchical structure is most similar to ours. The HSS ULV solver in [18] is a fast, recursive solver that computes a ULV decomposition implicitly. The HSS ULV algorithm is derived from the SSS solver in [13], and was amended to operate on all block rows at the same time. Then Chandrasekaran, Dewilde,

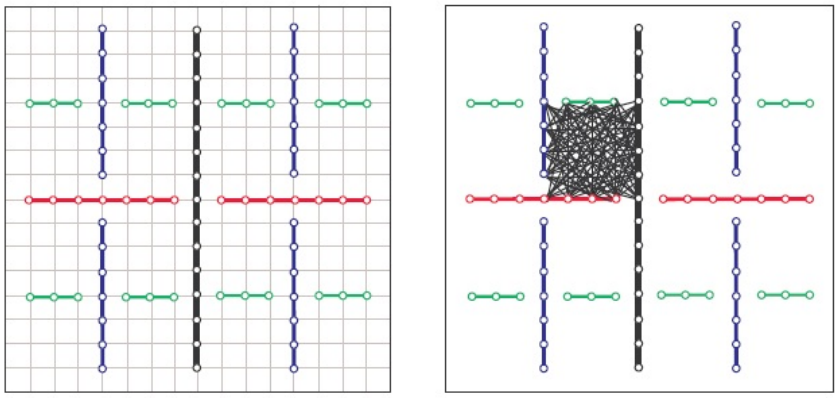
Gu, Lyons and Pals [19] took the HSS structure for a dense matrix and converted it into a larger sparse system of equations. The solver algorithm in [19] extends the HSS structure to take advantage of an efficient direct Gaussian elimination solver that is used on sparse systems. Lastly, in this group of solvers, is the work done by Xia, Chandrasekaran, Gu and Li [45] developing a fast direct solver for large discretized linear systems representing partial differential equations. The multifrontal solver covers a HSS structure with a mesh, the nodes are categorized into separators with nested dissection. The [45] algorithm then a supernodal multifrontal method eliminates the separators and accumulates updates locally following an elimination tree as illustrated in Figure 2.5. Table 2.2 summarizes solver algorithms for $n \times n$ rank structured matrices that have parameterizations, and r represents the rank of the parameterized blocks.

Table 2.2. Computational Complexity for Solvers of Rank Structured Matrices

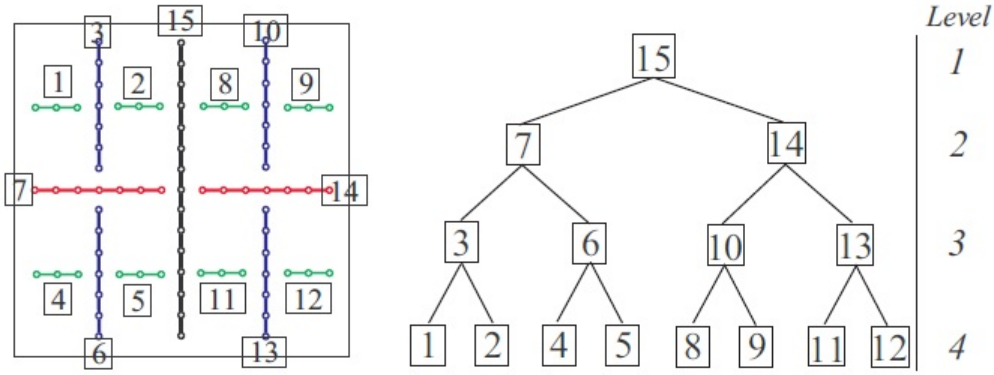
Parameterization, Solver Algorithm	Parameterization Cost	Solver Cost
SSS, SVD based solver [13]	$O(n^2)$	$O(nlog^2(n))$
SSS, SVD based solver [14]	$O(n^2)$	$O(n^2)$
HSS, ULV based solver [18]	$O(n^2)$	$O(n^2)$
HSS, ULV based solver [20]	$O(nr)$	$O(nr^3)$
Unitary weight, QR based solver [50]	$O(nr)$	$O(nr^2)$

The QR based solver by Delvaux and Van Barel [50] uses the Givens-weight representation of rank structured matrix A. The solver computes the QR factorization of the linear system $A\mathbf{x} = \mathbf{b}$, and uses the factorization $A = QR$ to transform the linear system to $R\mathbf{x} = Q^{\mathrm{H}}\mathbf{b} = (RV_l)(V_l^{-1}\mathbf{x}) = \tilde{\mathbf{b}}$ where V_l is an orthogonal operation. The equation continues to be transformed due to the Givens-weight representation, using $L = JRJ$ where J is an antidiagonal matrix such that $R\mathbf{x} = \tilde{\mathbf{b}}$ becomes $LJ\mathbf{x} = J\mathbf{b}$ and [50] finally solves for $L\mathbf{x} = \mathbf{b}$.

At the end of Section 1.5, the Bella, Olshevsky, Stewart solver was first mentioned. The fast UBV solver in [3] assumes that a quasiseparable matrix has been compressed into an upper triangular form and then further decomposed into a nested product. Prior to entering



(a) Partition with separators in nested dissection and the connections of mesh points during elimination.



(b) Ordering separators and corresponding separator tree/nested dissection elimination tree.

Figure 2.5. Example of stages in dissection of HSS in the multifrontal method, from [21].

the solver, the quasiseparable matrix must take the form

$$U_{k_0}\left(B_{k_0}+U_{k_1}\left(B_{k_1}+\cdots+U_{k_{p-1}}\left(B_{k_{p-1}}+D_{k_p}\right)V_{k_{p-1}}^{\mathrm{T}}\cdots\right)V_{k_1}^{\mathrm{T}}\right)V_{k_0}^{\mathrm{T}}$$

where U_k and V_k are Householder unitary transformations, B_k is a nonzero column, and D_k is a small diagonal block with only a few nonzero elements. When they solve the linear system, $A\mathbf{x}=\mathbf{b}$, the vector $\mathbf{y}$ represents the sequence of unitary transformations V^{T} times

the vector $\mathbf{x}$. Then they just peel apart the nested product beginning from the outside to solve the system. The solver from [3] is presented in detail in Section 5.2.

2.5 Applications of Quasiseparable Matrices

Image processing is becoming more instrumental in contemporary science and technology. The two areas in image processing that are applicable to the research in this dissertation are image restoration and image compression. In image restoration, we are particularly interested in satellite images that have been degraded due atmospheric turbulence because the blur can modeled by a degradation function which can be represented by a quasiseparable matrix. In image compression, we examine wavelets where a wavelet transform is applied the image and then the transform coefficients are compressed, thus compressing the image. In this application, the wavelet can be modeled by a hierarchical matrix

2.5.1 Image Deblurring

Image degradation is often modeled as a linear convolution of the original image with a point spread function (PSF) representing the blur [64, 65]. Toeplitz matrices are often used in forming the matrix of the PSF [66]. A standard technique in restoring an image is to use a preconditioner to approximate the blur operator. Given the blurred image equation $A_c X A_r^{\mathrm{T}} = B$ where X is the restored image, B is the blurred noise-free image, and the two structured matrices matrices A_c and A_r represent blurring in the directions of columns and rows of the image, respectively [67, 68]. The matrix representing the blur is transformed into a rank structured matrix where the off-diagonal blocks have low-rank, can be approximated quickly then exploited for fast image restoration [51]. With some approximation to A, we can obtain $X = A_c^{-1} B (A_r^{\mathrm{T}})^{-1}$ [32]. This is a separable blur model. More general linear transformations also arise in blur models. Techniques for addressing matrix noise and ill-conditioning include regularization, and iterative methods such as conjugate gradient and generalized minimal residual. Such approximations can be efficiently computed, efficiently

inverted, and can be applied as a preconditioner to enhance the convergence of an iterative method [69].

Benzi and Ng [70] consider two types of preconditioners for weighted Toeplitz matrices in an iterative solution. The two preconditioners are, a variant of constraint preconditioning, and the Hermitian/skew-Hermitian splitting preconditioner which are both rank structured. In [70] they employ circulant matrices, C, to precondition weighted Toeplitz matrices, T. More efficient tools for iterative procedures, preconditioners and matrix-vector multiplications in the restoration of images are explored by Nagy, Plamer and Perrone [71]. The work by Hansen and Jensen [72] examines the 2-D discrete cosine transform (DCT), and how noise, from both the signal and components of the solution, affects the reconstruction of images computed by regularizing iterations. They determine that the generalized minimal residual and minimal residual method are not suited for image deblurring.

The research proposed in this dissertation is a stable linear time algorithm for the solution of a rank structured system which is applied to a PSF represented by a Toeplitz matrix. The PSF is transformed into a rank structured Cauchy-like matrix. The rank structure is exploited by extracting the parameters and computing a hierarchical representation of the matrix. The resulting blur matrix is compressed, decomposed, and solved using nested products, the only algorithm proven to be backward stable. Once the blur system is solved, the solution is applied to the blurred image restoring it.

2.5.2 Image Compression via Wavelets

Mathematical wavelet transforms are used extensively in image compression. The huge volume of data in a direct image spatial domain is imprudent for transmission or storage. The wavelet transform maps the spatial domain of an image to a frequency domain, then excessive redundancies in the image are exploited and removed [64, 73, 74]. Wavelet transformations make it easier to compress, transmit and analyze images. The transform coding process is done in four major steps: apply the wavelet transform, detect the threshold, entropy code the quantized transform coefficients, and apply an inverse transform as shown in Figure 2.6

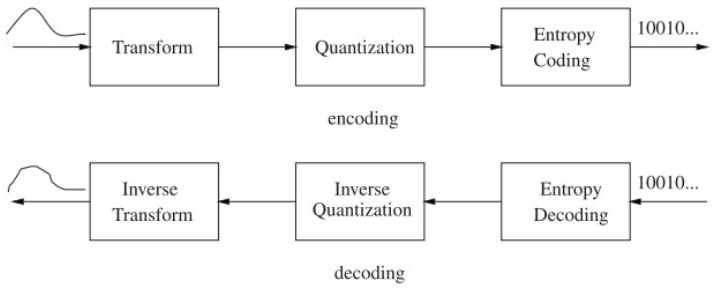


Figure 2.6. Diagram of the typical wavelet transform encoding and decoding process [75].

[75]. This method of lossy compression of the image is acceptable since the reconstruction of the image need not be exact. This dissertation focuses on the discrete wavelet transform (DWT) which computes the series expansion coefficients for a function that is comprised of a wavelet function, $\psi(x)$, and a scaling function, $\varphi(x)$.

A separable 2-D orthogonal Daubechies wavelet decomposition is computed of the image. The decomposed image forms a 2×2 block partitioned matrix W where the upper diagonal block approximates the image, the lower diagonal block locates the image's edges, the lower left off-diagonal block contains the vertical edges, and the upper right off-diagonal block contains the horizontal edges [64]. Once the wavelet compresses the image, a hierarchy is formed by the coefficients. The wavelet process can be repeated on the low frequency diagonal block and the result is a multiresolution hierarchical structure. The research in this dissertation experiments with further compression of wavelet transforms which have a somewhat hierarchical structure.

2.6 Overview

Rank structured matrices are coming to the forefront of research today and are copiously studied by a variety of groups exploring many different approaches. These matrices are significant in research for the interdisciplinary and complex problems they model, and the

exploitable data sparse characteristic that exists within the matrix. Confusion exists on the nomenclature of the subclasses of rank structured matrices, and their representations. The research in this dissertation deals with quasiseparable matrices explicitly. Some parameterizations or representations have been developed which allow solver algorithms to operate on the quasiseparable matrices with linear complexity. However, there are gaps in the types of parameterizations which leaves opportunity for new approaches to representations to be explored. Some research has been devoted to representation conversion which would allow the flexibility of moving from one representation to another; thus providing access to other algorithms. Furthermore, some of the parameterizations are costly to compute, and only one of the parameterization algorithms is proven to be stable.

This research presents a parameter extraction algorithm of a hierarchical representation for a quasiseparable matrix in Chapter 3. The computational complexity is of the hierarchical extraction is $O(n\log(n))$ which is improved or comparable to other parameterizations. The primary contribution of this research is the conversion of a hierarchical representation of a quasiseparable matrix to a nested product representation. The first step to the representation conversion is accomplished by way of the new row compression algorithm which is detailed in Chapter 3. The second phase, in the conversion to nested product, decomposes the hierarchical compressed matrix forming a UBV decomposition and is elaborated on in Chapter 4. The conversion algorithm has computation cost of $O(n^2)$ and is still comparable to other representationconversions.

The importance of the conversion, from hierarchical to nested product representation, is to take advantage of the only proven stable solver presented in [3]. Once the hierarchical representation has been converted to a nested decomposition, the research delves into the adaptation of the decomposition for implementation into the two algorithms presented in [3], the matrix-vector product and fast system solver discussed in Chapter 5. All the algorithms in this research focus on increased numerical stability while maintaining comparable speed

to those found in other previous works. A full error analysis was presented in [3] and shows the algorithms for nested products to be backward stable.

Two direct applications of the new algorithms are image restoration and compression. The image restoration in this research continues the initial work of [66] by directly deblurring an image, as opposed to approximating the image using iterative methods. The Toeplitz matrices represent the blur operators, and can be decomposed into nested products. It is well known that wavelet transforms have a hierarchical structure. In our experiments, an image is first compressed with the DWT, and the resulting coefficients are then sent through the new row compression algorithm to see if they can be compressed further. Currently, no other work in image processing is using a hierarchical row compression or a nested product solver approach to directly deblur or compress an image. The image processing applications are discussed in Chapter 5.

CHAPTER 3

ROW COMPRESSION ALGORITHM

The row compression algorithm and nested product decomposition of an $n \times n$ hierarchical representation of a rank structured matrix A, extend the compression and nested product decomposition of a quasiseparable matrix represented by the matrix elements. Once matrix A is decomposed into a nested product, then the proven stable matrix-vector multiplier and solver algorithms from [3] can be applied. The algorithms for the UBV decomposition in [3] carry out an $O(n^2)$ row compression and an $O(n^2)$ nested product decomposition by applying small Householder transformations directly to the matrix.

There are some matrices where low rank factorizations of off-diagonal blocks can be obtained directly by the truncation of a series, such as in the FMM from [48], or the transformation of a Toeplitz to a Cauchy-like matrix in [51]. Hierarchical representations of these types of matrices can be computed with far fewer operations than the $O(n^2)$ in [3]. The conversion of a hierarchical representation to a nested product as presented in this research is an attractive development in working with these types of rank structured matrices.

The row compression operates on matrix A in a bottom-up approach where the first compression operates on the rows associated with the last diagonal block at the bottom of matrix A, then sequentially moves up to the next diagonal block until it reaches the top. The row compression introduces as many zero rows into the lower triangular part of A as the ranks of the off-diagonal blocks will allow. There are two stages to the proposed parameterization: row compression and conversion to nested product of a rank structured matrix. An outline of the major components of the row compression algorithm follows:

Row Compression Algorithm

The algorithm input is the hierarchical representation of a rank structured matrix.

1. Perform a bottom-up recursive row compression, navigating the binary tree, and acting on the rows associated with each diagonal block.

2. Form a basis from a collection of the factors to the left of each diagonal block, and let the rank, s, of the basis determine the size of the repartition that adds extra nonzero rows below the diagonal to the upper left of the current block.

3. Compute a sequence of Householder vectors to introduce zeros into the basis, and place the vectors in a linked list.

4. Compress the lower left off-diagonal block region by applying the Householder transformations.

5. Repartition any left off-diagonal blocks, where a boundary will be crossed in the next compression, by merging the nonzero rows in the current blocks up into the blocks above.

6. Update the diagonal block and the right off-diagonal blocks by applying the Householder transformations, tag the top s nonzero rows for repartitioning, and repartition by merging these tagged rows from the diagonal block and/or the off-diagonal blocks into the block above.

The first and obvious step is to begin with a hierarchically partitioned and factored rank structured matrix. The overarching, recursive compression algorithm controls the interaction of the stages, and calls each procedure. The second stage is to collect the factors associated with a subset of the selected rows of A, into which zeros are introduced. Next follows the formation of a basis for the column space of the selected rows from the collected factors. The basis stage applies Householder transformations to the collection

in order to introduce zeros into the basis. From this basis, a Householder linked list is created to store the transformations applied to matrix A to compress the matrix. The linked list is stored and then will be later applied to update upper right off-diagonal blocks of matrix A when performing the nested product decomposition. Once zeros have been introduced into the selected rows of A, the matrix must be repartitioned to have the same overall structure but with different block sizes, taking into account the additional nonzero rows below each diagonal block. After the left off-diagonal blocks are compressed, the Householder transformations are applied to the diagonal and right off-diagonal blocks, and they are also repartitioned to be consistent with the partitioning of the rest of the matrix. The algorithm is then applied recursively to the leading principal submatrix. When the algorithm reaches the top of the matrix, the row compression is complete, and the matrix is ready for the nested product algorithm. Notations important in Chapter 3 are: 1) The factors $U_{(l,b)}, V_{(l,b)}, W_{(l,b)}$, and $X_{(l,b)}$ represent the factorization of off-diagonal blocks in matrix A. 2) The set $\mathscr{B}_k$ denotes the basis for the column space of the low rank blocks, whereas $B_{(l,b)}$ represents an off-diagonal block of matrix A . 3) The diagonal block of A is D_i.

3.1 Hierarchical Representation of the Quasiseparable Matrix

Matrix A is an $n \times n$ rank structured matrix, and a hierarchical parameterization of A must be given. A partition-factor procedure parameterizes the matrix forming a hierarchical representation. The partition-factor procedure performs a 2×2 block partitioning of the matrix, then does a low rank factoring of the off-diagonal blocks as shown in Equations 1.7 and 1.8. A binary tree is formed at the outset of the procedure. The parameterization procedure is performed recursively on the diagonal blocks until a terminal block size is reached. The terminal block size is determined by the rank of the off-diagonal blocks in conjunction with the smallest size acceptable for use with low rank factoring. The maximum size of the terminal blocks is set to $m_b \leq 4 \cdot \text{rank}$ at the start of the partition-factor

procedure, and is used as a parameter to determine depth, navigation, and termination of methods within the row compression algorithm. Some specific notation for the recursive block partitioning is necessary, and an example of this notation for the first level partitioning is

$$A = A_{1,1} = \begin{bmatrix} A_{1,1;(11)} & A_{1,1;(12)} \\ A_{1,1;(21)} & A_{1,1;(22)} \end{bmatrix}, \tag{3.1}$$

where $A_{1,1}$ represents level 1, block 1. The off-diagonal blocks $A_{1,1;(12)}$ and $A_{1,1;(21)}$ are factored, and the diagonal blocks $A_{1,1;(11)}$ and $A_{1,1;(22)}$ continue to be partitioned. The hierarchical representation of the rank structured matrix is constructed from the recursive partitions and creates a bidirectional, binary tree. The binary tree is illustrated by a block 8×8 example in Figure 3.1.

The binary tree describes how the rows and columns of A have been partitioned. The tree has an overall depth, d, based on parameter m_b, such that $d = \log_2(\frac{n}{m_b})+1$, where level 1 is the root of the tree and level d contains the leaves of the tree. The leaves of the tree contain the full rank diagonal blocks D_i where $i = 1, \ldots, 2^{d-1}$, and i denotes the position of the block along the diagonal from the top-left to the bottom-right. The diagonal block D_i is $m_i \times m_i$ where $m_i \leq m_b$. The nodes in a single level of the binary tree are numbered consecutively from left to right, and represent the number of blocks being partitioned at that level. The nodes in a single level correspond to the off-diagonal blocks oriented along the sub and super diagonal. The off-diagonal blocks are factored as $U_{(l,b)}V_{(l,b)}^{\mathrm{T}}$. The factors $U_{(l,b)}$ and $V_{(l,b)}$ have s columns where s is the rank of the off-diagonal block obtained from the singular value decomposition with the singular values included in $U_{(l,b)}$. The columns of $V_{(l,b)}$ are orthonormal, but those of $U_{(l,b)}$ are not. Each node of the binary tree stores comprehensive information about the new hierarchical representation that can be easily accessed, modified, and manipulated during the compression algorithm. The most crucial information is the two pairs of factors for the upper and lower off-diagonal blocks of the larger diagonal block that was partitioned. The upper right off-diagonal blocks have factors $W_{(l,b)}$ and $X_{(l,b)}$, and lower left off-diagonal blocks have factors $U_{(l,b)}$ and $V_{(l,b)}$. The notation used to identify the

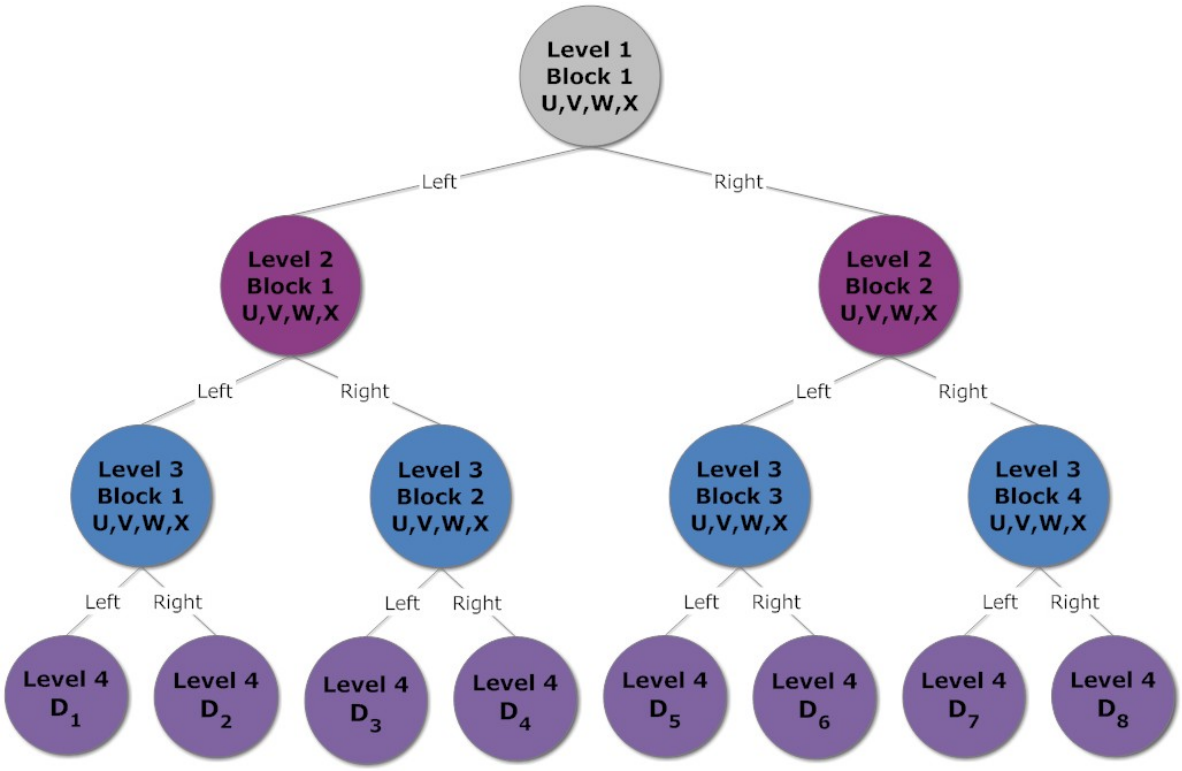


Figure 3.1. Binary tree of the hierarchical representation for a block 8×8 example of the rank structured matrix A. The tree has 8 leaves (diagonal blocks) and a depth of 4. Each node stores pointers to the parent and/or child nodes, indices for the matrix, and flags for navigating the binary tree.

location of factors in the rank structured matrix is $U_{(l,b)}$ where $l = 1, \ldots, d$ represents the level and $b = 1, \ldots, 2^{l-1}$ is the block and node number within that level.

Further explanation of the notation in Equation 3.1 continues with the two off-diagonal blocks defined as $A_{1,1;(12)} = W_{(1,1)}X_{(1,1)}^{\mathrm{T}}$ and $A_{1,1;(21)} = U_{(1,1)}V_{(1,1)}^{\mathrm{T}}$, and the diagonal blocks defined as $A_{1,1;(11)} = A_{2,1}$ and $A_{1,1;(22)} = A_{2,2}$. Thus,

$$A = \begin{bmatrix} A_{2,1} & A_{1,1;(12)} \\ A_{1,1;(21)} & A_{2,2} \end{bmatrix} = \begin{bmatrix} A_{2,1} & W_{(1,1)}X_{(1,1)}^{\mathrm{T}} \\ U_{(1,1)}V_{(1,1)}^{\mathrm{T}} & A_{2,2} \end{bmatrix} \tag{3.2}$$

Continued partitioning of A yields

$$A = \begin{bmatrix} \begin{bmatrix} A_{3,1} & W_{(2,1)}X_{(2,1)}^{\mathrm{T}} \\ U_{(2,1)}V_{(2,1)}^{\mathrm{T}} & A_{3,2} \end{bmatrix} & W_{(1,1)}X_{(1,1)}^{\mathrm{T}} \\ U_{(1,1)}V_{(1,1)}^{\mathrm{T}} & \begin{bmatrix} A_{3,3} & A_{2,2;(12)} \\ U_{(2,2)}V_{(2,2)}^{\mathrm{T}} & A_{3,4} \end{bmatrix} \end{bmatrix}.$$

Note that block $A_{3,4}$ in the 8×8 partitioning of A belongs to node 4 on level 3 of the binary tree in Figure 3.1. In general terms, block $A_{l,b}$ is defined as node b at level l, and brings about

$$A_{l,b} = \begin{bmatrix} A_{l+1,2b-1} & A_{l,b;(12)} \\ A_{l,b;(21)} & A_{l+1,2b} \end{bmatrix}. \tag{3.3}$$

Every off-diagonal block $A_{l,b;(12)}$ can be written as

$$A_{l,b;(12)} = W_{(l,b)}X_{(l,b)}^{\mathrm{T}}, \tag{3.4}$$

Every off-diagonal block $A_{l,b;(21)}$ can be written as

$$A_{l,b;(21)} = U_{(l,b)}V_{(l,b)}^{\mathrm{T}}. \tag{3.5}$$

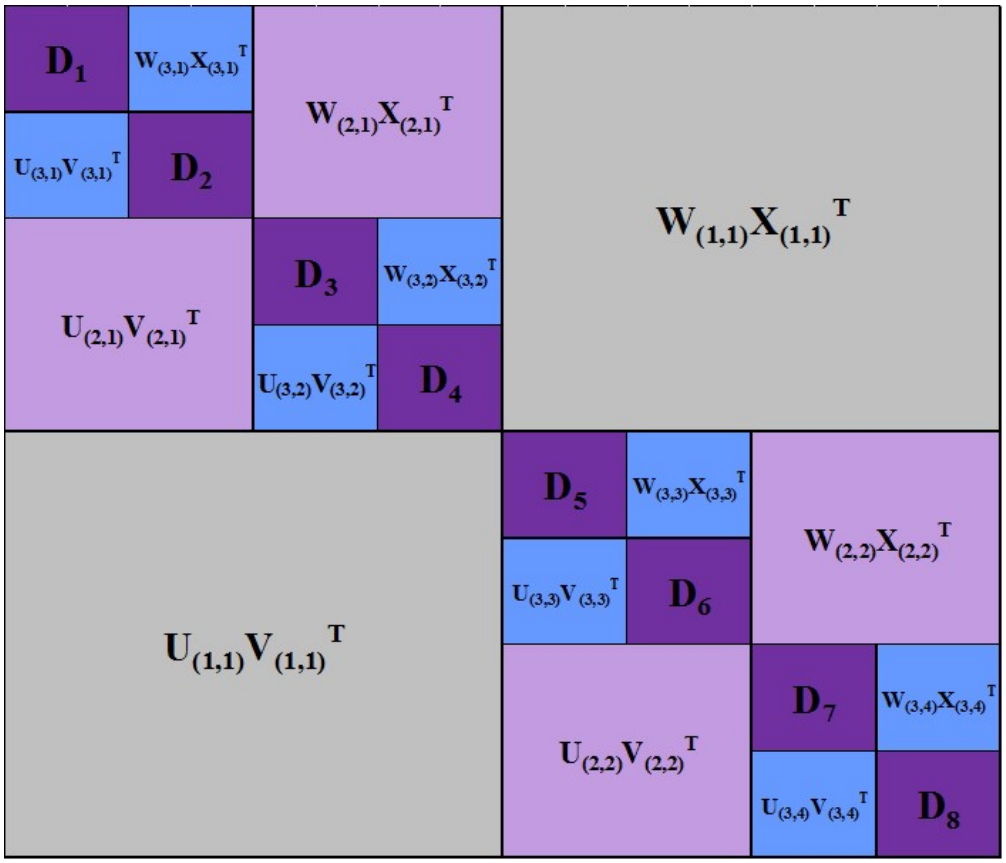


Figure 3.2. The quasiseparable matrix after it is partitioned and factored into a hierarchical representation. The D_i diagonal blocks are full rank where $i = 1, \ldots, 2^{d-1}$ and $d = \log_2(\frac{n}{m_b}) + 1$ is the depth of the binary tree. The off-diagonal blocks $U_{(l,b)}V_{(l,b)}^{\mathrm{T}}$ and $W_{(l,b)}X_{(l,b)}^{\mathrm{T}}$ are low rank where $l = 1, \ldots, d$ denotes the level and $b = 1, \ldots, 2^{l-1}$ is the block number within that level.

Another view of the hierarchical representation, recursive block partitioning corresponding to the binary tree in Figure 3.1, is shown in Figure 3.2. The off-diagonal blocks in this view of the representation are denoted by $U_{(l,b)}, V_{(l,b)}, W_{(l,b)}$, and $X_{(l,b)}$.

In Figure 3.2, the block containing $U_{(3,2)}V_{(3,2)}^{\mathrm{T}}$ is located in the lower triangular part of the matrix, block 2 in level 3, along the subdiagonal of matrix A. The off-diagonal blocks, $B_{(l,b)}$, in the lower triangular region of the matrix $B_{(l,b)} = U_{(l,b)}V_{(l,b)}^{\mathrm{T}}$ and in the upper triangular region of the matrix $B_{(l,b)} = W_{(l,b)}X_{(l,b)}^{\mathrm{T}}$.

The nodes contain information on parent and child nodes, right or left branching down and up the tree, flags indicating whether it is root, leaf or branch in the tree, and index information for location in original matrix A. In preparation for compression, data structures are created noting which levels and off-diagonal blocks are directly associated with a particular diagonal block. A basic list of the hierarchical information stored in a node is:

- Numeric representation of node level, size, and type (i.e. branch, leaf or root).
- The U, V, W, X factors of the off-diagonal blocks for the level
- Indices from the original matrix A.
- Boolean flags for the root, branch, leaf, left move, and right move in the binary tree.
- Pointers to the left, right and/or parent nodes of the binary tree.
- If the node is a leaf or diagonal block, then the original matrix data, diagonal block number, and pairing of the level and path movement from the root to the leaf.

Examination of binary tree structure reveals that all the right branches from the root to a specific leaf node, or diagonal block, correspond to the left adjacent off-diagonal blocks, in the lower left part of the matrix, to the diagonal block.

3.2 Control Algorithm for Row Compression

The control algorithm for row compression begins at the last diagonal block, D_{n/m_b}, at the bottom of matrix A. The design of the control algorithm is to recursively compress the matrix in a sequential order from the bottom to the top. The algorithm uses D_i as the key for where the row compression operates in matrix A, and updates the linked list of Householder vectors at each stage. The control algorithm feeds the binary tree into the various procedures during compression to modify the data. Additional data structures are created, for the branching levels and indices, from the information stored in the leaf node

for D_i of the tree. These data structures direct which off-diagonal blocks are operated on in forming a low rank block column space basis and the subsequent computation of Householder vectors. Crucial to the control algorithm is navigating the binary tree from node to node, and acting on the rows associated with each diagonal block throughout the matrix. Both capabilities facilitate the compression and access to data in the tree.

In order to navigate between leaves, breadth movement, and between levels on a path to a leaf, depth movement, a handful of traversal functions are utilized. All of the traversal methods capitalize on the boolean flags in the nodes allowing the compression algorithm to move within the binary tree from D_i up to the root. The methods use a level-path pairing data structure that stores the path of each diagonal block D_i from the root to the leaf, pairing the level with a right or left branch flag. Hence, navigating the tree can be done by accessing the level-path pairing to move from the root back down the tree according the path of a particular diagonal block, either D_{i-1} or D_{i+1}. A more important traversal method is the movement to different hierarchical levels in the tree stopping to access the factors along the path towards a diagonal block. Again, the level-path pairing is used; however, the method stops a specified levels, not at leaves, which requires tracking.

The control algorithm directs all the other major algorithms or procedures in compressing matrix A. The row compression applies transformations from the left and is accomplished in three stages: the left off-diagonal region with $U_{(l,b)}$ and $V_{(l,b)}$ factors, the diagonal block, and the right off-diagonal region with factors $W_{(l,b)}$ and $X_{(l,b)}$. The single most difficult piece of the compression is repartitioning the affected blocks at the end of each compression stage. Repartitioning is the removal of nonzero rows from the already compressed area, and merging these rows up above into the area yet to be compressed. This requires intricate manipulation of the binary tree and the data structures.

3.3 Collection of Factors and Formation of Basis

Once the rank structured matrix is hierarchically partitioned and factored, the factors for a region associated with diagonal block, D_i, and located in the off-diagonal blocks to the left of D_i are collected to form a basis for the column space of the low rank blocks currently being compressed. This signals the preparatory phase of the row compression algorithm. The selection of the off-diagonal blocks associated with a specific diagonal block, D_i, for compression, and their corresponding $U_{(l,b)}$ and $V_{(l,b)}$ factors, is determined by two equivalent characteristics. The first characteristic is that the off-diagonal block lie strictly to the left of the diagonal block, and the second one is that the off-diagonal block is a right branch in the binary tree path towards the diagonal block to the right of the rows being compressed. The column space for the basis is determined by $U_{(l,b)}$. The rows in D_i are identified, and determine which rows in each $U_{(l,b)}$ factor of the off-diagonal blocks will be in the column space. As the off-diagonal blocks are selected, only the pertinent rows in each $U_{(l,b)}$ are collected. The possible number of rows to be acted on is z where $m_b \leq z \leq 2 \times m_b$. The column span used in the collection region is based on the target D_i, and all columns to the left of D_i. These columns span across different levels of the partitioned, factored hierarchical representation of the quasiseparable matrix. Therefore, all columns and rows of V are collected. Hence, the size of the collection is just a few rows tall and across all columns strictly to the left of the diagonal block. We will refer to the entire region as $Collection_{D_i}$ where i refers to the index of the diagonal block. The collection of factors for D_7 found in the block 8×8 example is shown in Figure 3.3.

The collection method examines the branching from path information stored in D_i. The method loops through each level from root to leaf, and only collects the $U_{(l,b)}$ and $V_{(l,b)}$ factors of right branches. As the factors are collected, they are placed in a secondary smaller data structure which is passed into the procedure that forms the basis. The collection of factors data structure is illustrated in Figure 3.4 for the region associated with diagonal block D_7 and is referred to as $Collection_{D_7}$. The D_7 collection selects U and V factors only

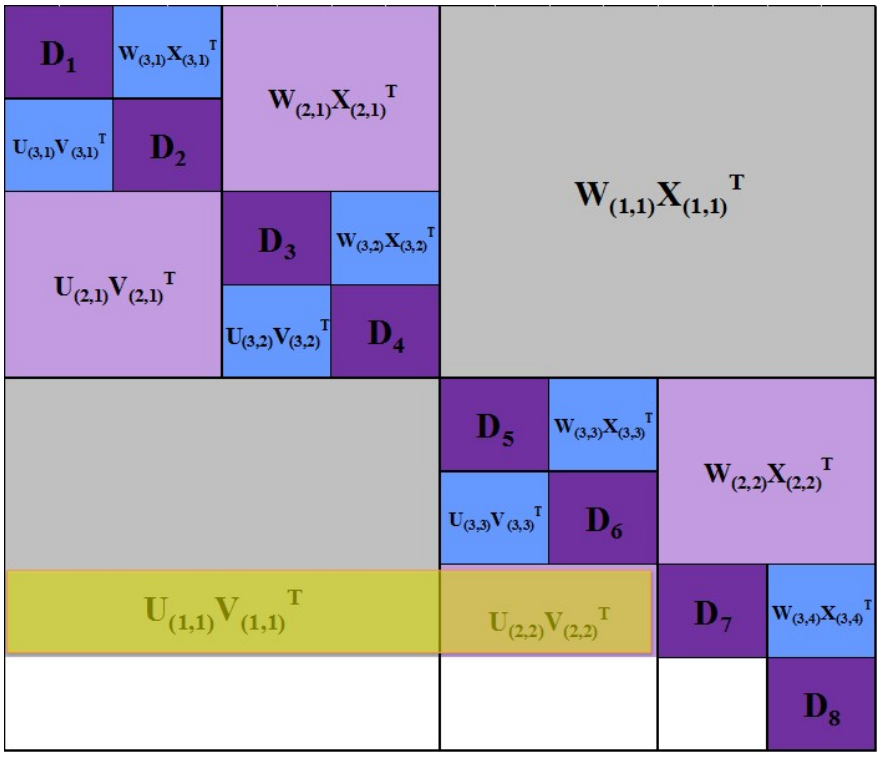


Figure 3.3. The figure highlights the collection of factors for diagonal block D_7, $Collection_{D_7}$, in the hierarchical representation of the matrix.

from levels 1 and 2 of the tree. These two levels are where right branches occur in the binary tree along the path from the root to diagonal block D_7 as seen in Figure 3.1.

In general, a basis $\mathscr{B}$ is a set of linearly independent vectors that span a given subspace. Every element in the subspace is expressed uniquely as a finite linear combination of basis vectors. Forming a basis, $\mathscr{B}_i$, for the column space of the rows in each $Collection_{D_i}$ eliminates extra multiplication operations in computing the Householder vectors of the much larger overall matrix A. The basis formation method operates on the collection of factors data structure as illustrated in Figure 3.4. In Figure 3.4, the rank of the lower triangular blocks is s. Thus, the number of columns in each factor, $U_{(l,b)}$ and $V_{(l,b)}$, are based on the rank s. This basis formation procedure isolates the $V_{(l,b)}$ column factors. The $V_{(l,b)}$ column factors create a tall skinny matrix.

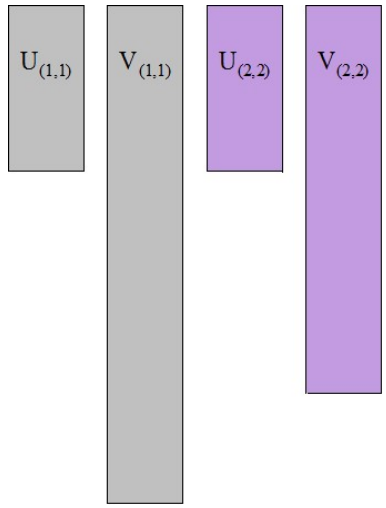


Figure 3.4. The factor data structure for $Collection_{D_7}$ where $U_{(1,1)}$ and $U_{(2,2)}$ are $m_b \times s$; and $V_{(1,1)}$ is $\frac{n}{2} \times r$, and $V_{(2,2)}$ is $\frac{n}{4} \times s$; and s represents the rank of the off-diagonal blocks.

An LQ factorization method is applied to each $V^{\mathrm{T}}_{(l,b)}$ matrix from a single level in $Collection_{D_i}$ to reduce the size of each $V^{\mathrm{T}}_{(l,b)}$. The LQ factorization is a transposed variant of the QR decomposition, and it is obtained from Householder transformations applied directly to $V^{\mathrm{T}}_{(l,b)}$. The LQ factorization of $V^{\mathrm{T}}_{(l,b)}$ yields a reduced matrix which is then transposed into in lower triangular block form.

$$\tilde{V}^{\mathrm{T}}_{(l,b)} = \begin{bmatrix} V^{\mathrm{T}}_{(l,b)} \end{bmatrix} P = \begin{bmatrix} L_{(l,b)} & 0 \end{bmatrix}. \tag{3.6}$$

A subblock of the basis is thus formed by multiplying $U_{(l,b)}$ times $L_{(l,b)}$ as follows,

$$\begin{bmatrix} U_{(l,b)} \cdot L_{(l,b)} & 0 \end{bmatrix}.$$

An intermediate matrix $\mathscr{B}$ is formed by appending the products of select rows of each U and reduced V from the off-diagonal blocks beside each other. This intermediate basis formation

of $Collection_{D_7}$, from Figure 3.4, is

$$\mathscr{B} = \left[\begin{array}{cc|cc} U_{(1,1)} \cdot L_{(1,1)} & 0 & U_{(2,2)} \cdot L_{(2,2)} & 0 \end{array} \right]. \tag{3.7}$$

An efficient SVD factorization operates on the intermediate matrix $\mathscr{B}$, for example $\mathscr{B} = U\Sigma V^{\mathrm{T}}$ (refer to Section 1.1). The efficient SVD first performs a QR decomposition of $\mathscr{B}$ to obtain R, and stores the sequence of Householder vectors in a linked list. Next, R is decomposed with a regular SVD algorithm where the rank of R, r, is determined by the number singular values larger than a given tolerance level. The orthogonal matrix U, from the SVD of R, is reduced to only the first r columns, and $\tilde{U} = U(:, 1:r)$. The final step is to apply the Householder transformations from the linked list to $\tilde{U}$ producing a basis, $\mathscr{B}_i$, for $Collection_{D_i}$. The basis formation method returns the basis $\mathscr{B}_i$ and basis rank r.

3.4 Householder Transformation Computation from Basis

When dealing with a tall matrix A that is $m \times n$ where $m \geq n$, and its QR decomposition, explicitly forming Q requires storing m^2 elements. It is faster to store only the Householder transformations, since only n Householder vectors are required. The Householder vectors computed from the reduced matrix basis, $\mathscr{B}_i$, are equivalent to those computed from the larger submatrix $Collection_{D_i}$ and the original matrix A. In this procedure, a Householder bidirectional linked list data structure, Q_i^{T}, is created which stores the Householder vectors and offsets for the location of submatrix $Collection_{D_i}$. Note that i refers to z number of rows in diagonal block, D_i, and indicates to which rows the Householder transformation is to be applied. Additionally, other pertinent information is saved that is necessary in subsequent updating and compression procedures.

The procedure for computing the sequence of Householder vectors begins with $\mathscr{B}_i$, and is essentially performing the QR factorization of B_i, which was introduced in Section 1.2. The recursive procedure introduces zeros in to each successive column of the matrix, then passes in

the next smaller submatrix. The Wilkinson diagram below shows how the procedure operates on an example of a basis that is 6×4 with a sequence of four Householder transformations,

$$\mathscr{B}_i = \left[\begin{array}{c|c|c|c} \times & \times & \times & \times \\ \hline \otimes_1 & \times & \times & \times \\ \hline \otimes_1 & \otimes_2 & \times & \times \\ \hline \otimes_1 & \otimes_2 & \otimes_3 & \times \\ \hline \otimes_1 & \otimes_2 & \otimes_3 & \otimes_4 \\ \hline \otimes_1 & \otimes_2 & \otimes_3 & \otimes_4 \end{array}\right].$$

The factorization is

$$\mathscr{B}_i = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix}. \tag{3.8}$$

The column space of Q_1 is equivalent to the basis, $\mathcal{R}(Q_1) = \mathcal{R}(\mathscr{B})$. The sequence of four Householder transformations, P, form Q where $Q = P_1 P_2 P_3 P_4$. It follows that

$$Q_1 = Q \begin{bmatrix} I \\ 0 \end{bmatrix} = P_1 P_2 P_3 P_4 \begin{bmatrix} I \\ 0 \end{bmatrix}. \tag{3.9}$$

For the purposes of the compression algorithm, each Householder vector is inserted into the tail of the linked list as it is computed. The linked list is represented by Q_i^{T} where i is the index of diagonal block D_i, and operates on the rows from D_i. The linked list itself marks the head and tail of list to allow for traversing the list both forwards or backwards. When the Householder vectors are applied from the left of submatrix $\mathit{Collection}_{D_i}$, zeros are introduced into rows of the off-diagonal blocks to the left of D_i. The diagonal block D_i, although unaffected at this point, is dense and potentially full rank. Thus, the computed Householder linked list from the smaller basis matrix B_i will compress the rows of matrix A corresponding to $\mathit{Collection}_{D_i}$. It is imperative to track the Householder linked lists, Q_i^{T}, for each $\mathit{Collection}_{D_i}$, and create a comprehensive Householder data structure, Q_i^{T}, for use

in the compression stages of the control algorithm which actually apply the Householder to A.

After Q_i^{T} is returned from the Householder computation procedure to the control algorithm, the indices of the diagonal block D_i are recalculated based on rank r from basis $\mathscr{B}_i$. The control algorithm then invokes a traversal method to access node D_{i-1}, and prepares to call the first of the three compression stages.

3.5 Compression of the Lower Left

The compression of the off-diagonal blocks in the lower left of the hierarchical representation begins with the $U_{(l,b)}$ factors in all right branch nodes corresponding to leaf D_i. The Householder transformation Q_i^{T}, obtained from the basis of the factors with respect to D_i, will introduce zeros into each $U_{(l,b)}$ factor directly to the left of D_i. Thus, the process of Householder transformations quickly introduces zeros into the off-diagonal blocks of the hierarchical representation of the matrix. Figure 3.5 is a diagram of a block 8×8 hierarchical factored matrix prior to compression. Let U_f represent a factor in a right branch in the path to D_i, and assume U_f has size $m_f \times k_f$ where $k_f \leq m_f$. When U_f is multiplied by the Householder transformation,

$$U_f = Q_i \begin{bmatrix} \tilde{U}_f \\ 0 \end{bmatrix} \tag{3.10}$$

then the last $m_f - k_f$ rows of the off-diagonal block become zero. An illustration of the first and second compression of the block 8×8 introducing zeros into the off-diagonal region directly to the left of D_8 and D_7 is shown in Figure 3.6.

The compression of the $U_{(l,b)}$ and $V_{(l,b)}$ factors in the off-diagonal blocks to the left of D_i is the most straight forward of the three stages in compression and repartitioning. The compression of the matrix is done sequentially beginning at the bottom of the matrix and moving up. It is important to note that after every compression there is a repartitioning of the last left off-diagonal block involved in the compression. Therefore, the algorithm must

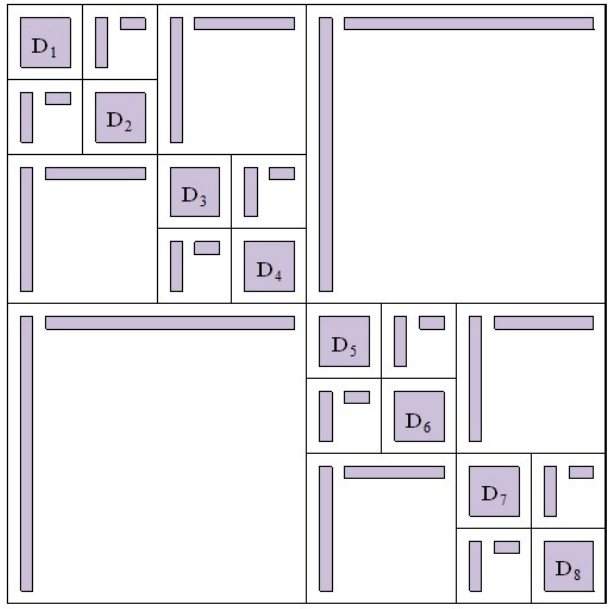
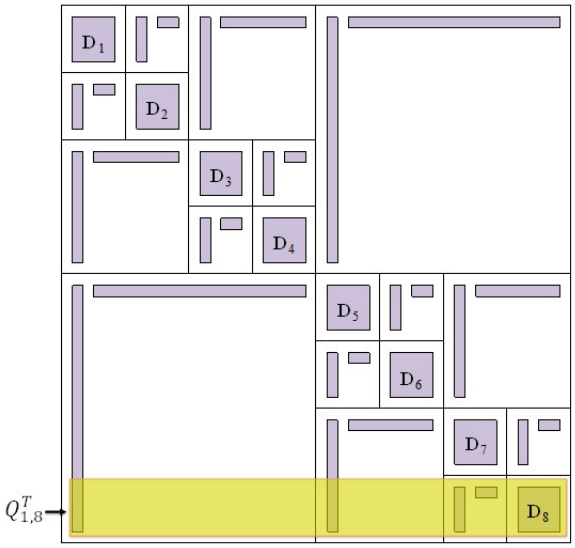


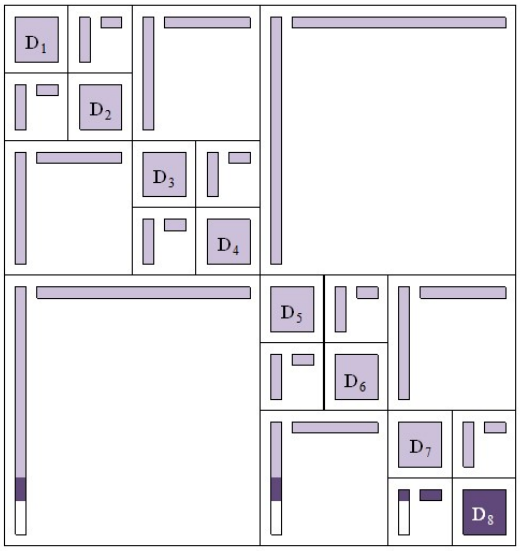
Figure 3.5. Diagram of a block 8×8 example displaying the factors in a hierarchical representation of a rank structured matrix. The shaded rectangles represent the nonzero portions of the factors in the off-diagonal blocks and the dense diagonal blocks.

tag the nonzero portion of the factors to be repartitioned. The last right branch node, or off-diagonal block closest to D_i, in the current compression will have the first s rows in U tagged for repartitioning where s is the rank of the basis formed from $Collection_{D_i}$.

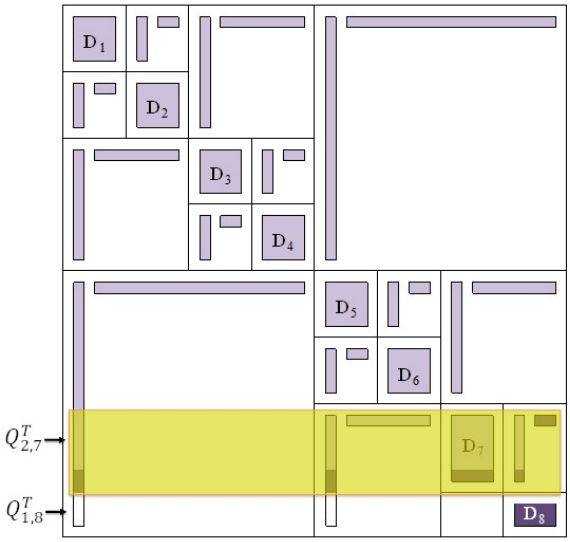
Let $B_{(q,k)}$ be the left off-diagonal block closest to D_i where q is the level and k is the block number. Repartitioning will occur at level q. The tagged part of the $U_{(q,k)}$ factor is referred to as U_r, and is what remains of factor $U_{(q,k)}$. Factor $U_{(q,k)}$ has zeros introduced into the bottom of the factor, so essentially the top s rows of $U_{(q,k)}$ are nonzero. At level q, all of $V_{(q,k)}$ is tagged for repartitioning, and is denoted by V_r. Off-diagonal block $B_{(q,k)}$ contains both U_r and V_r. In Figure 3.6(b), the dark shaded part of block $U_{(3,4)}V_{(3,4)}^{\mathrm{T}}$ is tagged. Likewise, in Figure 3.6(d), the dark shaded part of block $U_{(2,2)}V_{(2,2)}^{\mathrm{T}}$ is tagged. Both dark shaded tagged portions are used in repartitioning before the next compression. The repartitioning of blocks is presented the next section.



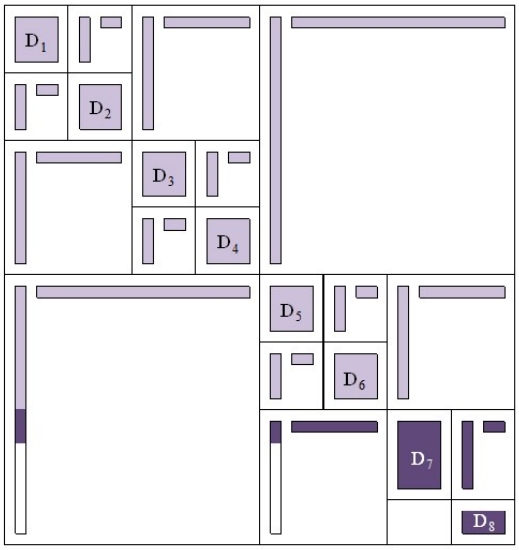
(a) The yellow highlights the area of the first compression. The Householder transformation $Q^{\mathrm{T}}_{(1,8)}$, first compression on 8th block, is applied to the area in yellow.



(b) The white portion of the rectangles is where zeros have been introduced. The dark shaded portions are where $Q^{\mathrm{T}}_{(1,8)}$ has been applied. The light shaded areas are nonzero portions yet to be compressed.



(c) The second region in the sequence of compressions is highlighted. The transformation $Q^{\mathrm{T}}_{(2,7)}$ represents the second compression on 7th block. This region has been repartitioned prior to the compression.



(d) The dark shaded portions are where the Householder transformations have been applied.

Figure 3.6. An illustration for the compression of the lower left off-diagonal blocks.

3.6 Repartition Blocks

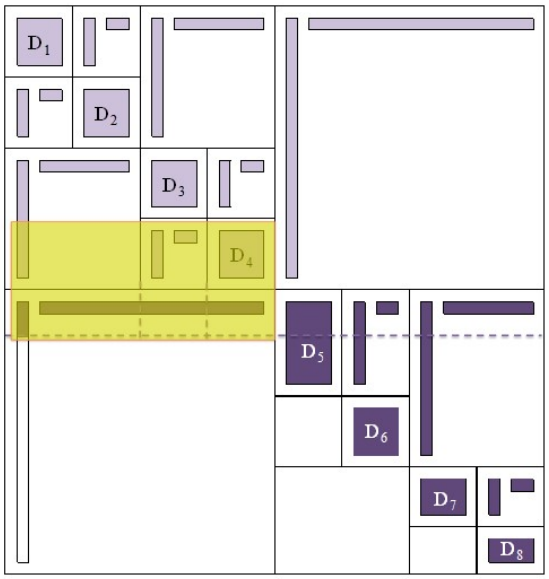
The repartition algorithm is comprised of a factor selection procedure and a merge procedure. The merge procedure combines two pairs of factors into a single pair, and has a detailed mathematical explanation presented later in this section. The $U_{(l,b)}V_{(l,b)}$ factor selection procedure begins with the tagged pair of factors, U_r and V_r, found in off-diagonal block $B_{(q,k)}$ associated with diagonal block D_i where q represents the level and k is the block number. The selection procedure moves up to the D_{i-1} diagonal block region locating the block(s) which lay directly above the off-diagonal block $B_{(q,k)}$. The columns of block $B_{(q,k)}$ will span one diagonal block D_{i-1}, or span D_{i-1} plus additional off-diagonal blocks above $B_{(q,k)}$. If block $B_{(q,k)}$ spans only block D_{i-1}, then $U_rV_r^{\mathrm{T}}$ is simply appended to the bottom of D_{i-1} as follows

$$\tilde{D}_{i-1} = \begin{bmatrix} D_{i-1} \\ U_rV_r^{\mathrm{T}} \end{bmatrix}. \tag{3.11}$$

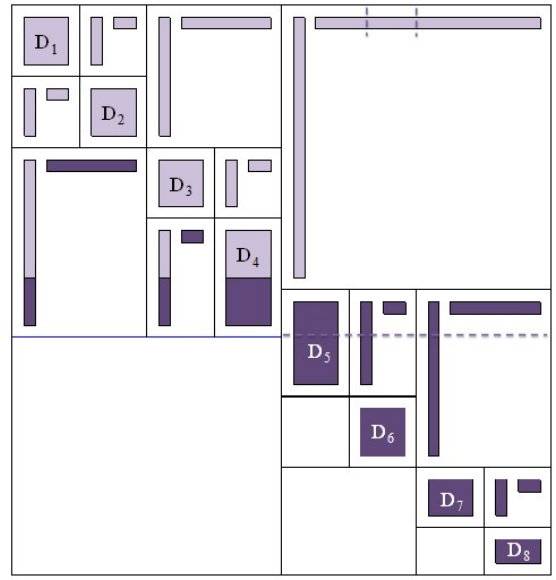
If block $B_{(q,k)}$ spans multiple blocks directly above it, then factor V_r is partitioned to align with the columns in the blocks above. In the D_{i-1} diagonal block region, suppose blocks $U_{(l,b)}V_{(l,b)}^{\mathrm{T}}$ are directly above $B_{(q,k)}$ where $l = q+1, \ldots, d$; q is the level of the off-diagonal block below; and d is the depth of the tree. Additionally, b is the block number at level l of a right branch in the path to diagonal block D_{i-1}. Given that $V_{r_i}^{\mathrm{T}}$ is a subblock of V_r^{T} such that the columns align with the off-diagonal block above, and $i = 1, \ldots, d-q$. Thus,

$$\left[\begin{array}{c|c|c|c} U_{(r+1,b)}V_{(r+1,b)}^{\mathrm{T}} & \cdots & U_{(d,c)}V_{(d,c)}^{\mathrm{T}} & D_{i-1} \\ \multicolumn{4}{c}{U_rV_r^{\mathrm{T}}} \end{array}\right] = \left[\begin{array}{c|c|c|c} U_{(r+1,b)}V_{(r+1,b)}^{\mathrm{T}} & \cdots & U_{(d,c)}V_{(d,c)}^{\mathrm{T}} & D_{i-1} \\ U_rV_{r_1}^{\mathrm{T}} & \cdots & U_rV_{r_{d-q-1}}^{\mathrm{T}} & U_rV_{r_{d-q}}^{\mathrm{T}} \end{array}\right]. \tag{3.12}$$

A diagram of V_r being partitioned in the selection procedure is shown in Figure 3.7(a). In Figure 3.7(a), $U_{(1,1)}V_{(1,1)}^{\mathrm{T}}$ is $U_rV_r^{\mathrm{T}}$, and is being merged into blocks $U_{(2,1)}V_{(2,1)}^{\mathrm{T}}$, $U_{(3,2)}V_{(3,2)}^{\mathrm{T}}$, and D_4. The columns of $V_{r_1}^{\mathrm{T}}$ align with $V_{(2,1)}^{\mathrm{T}}$, the columns of $V_{r_2}^{\mathrm{T}}$ align with $V_{(3,2)}^{\mathrm{T}}$, and the columns of $V_{r_3}^{\mathrm{T}}$ align with D_4.



(a) The yellow highlights the area for repartitioning after compressing the left off-diagonal blocks associated with diagonal block D_5. The dark shaded factors on the left are to be merged above. Factor V_r^{T} is partitioned into three portions to merge with D_4 and the two off-diagonal blocks above.



(b) The dark shaded portions to the left of D_5 have been merged. It is easy to see where $U_r V_{r_1}^{\mathrm{T}}, U_r V_{r_2}^{\mathrm{T}}$, and $U_r V_{r_3}^{\mathrm{T}}$ are now merged above into $U_{2,1}V_{2,1}^{\mathrm{T}}, U_{3,2}V_{3,2}^{\mathrm{T}}$, and D_4 respectively. All affected blocks are now repartitioned.

Figure 3.7. Repartitioning of the left off-diagonal block closest to D_5.

The most important procedure to the repartitioning of the blocks is the vertical merge of factored blocks together into a single block represented by one pair of factors. Given two factored blocks B_1 and B_2 such that B_1 is $m_1 \times n$ and B_2 is $m_2 \times n$, and B_1 is directly above B_2 as shown below

$$\begin{bmatrix} B_1 \\ B_2 \end{bmatrix} = \begin{bmatrix} U_1 V_1^{\mathrm{T}} \\ U_2 V_2^{\mathrm{T}} \end{bmatrix}. \tag{3.13}$$

We assume that the U's have orthogonal columns, and write

$$\begin{bmatrix} U_1 V_1^{\mathrm{T}} \\ U_2 V_2^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} \begin{bmatrix} V_1^{\mathrm{T}} \\ V_2^{\mathrm{T}} \end{bmatrix} \tag{3.14}$$

where the U factor matrix is $(m_1+m_2)\times s_1$ and the V factor matrix is $(m_1+m_2)\times s_2$. Next, the V factor matrix is reduced by a series of Householder transformations which results in

$$\begin{bmatrix} V_1^{\mathrm{T}} \\ V_2^{\mathrm{T}} \end{bmatrix} Q = \begin{bmatrix} L & 0 \end{bmatrix}.$$

Here, L is $s_2 \times s_2$, and Q represents the Householder transformation of the V factor matrix. Then by using the SVD, L is broken down so that a rank decision can be made. Thus

$$L = U_L \begin{bmatrix} \Sigma_L & 0 \\ 0 & 0 \end{bmatrix} V_L^{\mathrm{T}} \tag{3.15}$$

where U_L is $s_2 \times s_2$, Σ_L is $r \times r$, and U_L is $s_2 \times s_2$. From Equations 3.14 and 3.15, one obtains

$$\begin{bmatrix} U_1 V_1^{\mathrm{T}} \\ U_2 V_2^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} \begin{bmatrix} U_L \begin{bmatrix} \Sigma_L & 0 \\ 0 & 0 \end{bmatrix} V_L^{\mathrm{T}} & 0 \end{bmatrix} Q^{\mathrm{T}}. \tag{3.16}$$

Lastly, the U_1 and U_2 are combined to form U_M, and V_M is formed from $\Sigma_L, V_L^{\mathrm{T}}$, and Q^{T} seen below

$$U_M V_M^{\mathrm{T}} = \begin{bmatrix} \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} & U_L \begin{bmatrix} I \\ 0 \end{bmatrix} \end{bmatrix} \begin{bmatrix} \Sigma_L \begin{bmatrix} I & 0 \end{bmatrix} & \begin{bmatrix} V_L^{\mathrm{T}} & 0 \end{bmatrix} Q^{\mathrm{T}} \end{bmatrix}. \tag{3.17}$$

An example of this is portrayed in Figure 3.7(b) where $U_r V_r^{\mathrm{T}}$ is merged into blocks $U_{(2,1)}V_{(2,1)}^{\mathrm{T}}, U_{(3,2)}V_{(3,2)}^{\mathrm{T}}$, and D_4. The example in Figure 3.7(b) shows how the block sizes have been repartitioned, and from Equations 3.12 and 3.17, the merged expression is

$$\left[\begin{array}{c|c|c} U_{(M2,1)}V_{(M2,1)}^{\mathrm{T}} & U_{(M3,2)}V_{(M3,2)}^{\mathrm{T}} & D_{M4} \end{array}\right] = \left[\begin{array}{c|c|c} U_{(2,1)}V_{(2,1)}^{\mathrm{T}} & U_{(3,2)}V_{(3,2)}^{\mathrm{T}} & D_4 \\ U_r V_{r_1}^{\mathrm{T}} & U_r V_{r_2}^{\mathrm{T}} & U_r V_{r_3}^{\mathrm{T}} \end{array}\right].$$

This merge procedure is used again in repartitioning the diagonal and right off-diagonal blocks in a D_i region.

3.7 Update and Repartitioning Upper Right

Once the lower left off-diagonal blocks are compressed and repartitioned, the Householder transformation is applied to diagonal block D_i, and to all right off-diagonal blocks $W_{(l,b)}X^{\mathrm{T}}_{(l,b)}$. In the previous section, block $B_{(q,k)}$ is the left off-diagonal block that is repartitioned where q is the level and k is the block number. When D_i is multiplied by the Householder transformation, it yields $\hat{D}_i = Q_i^{\mathrm{T}} D_i$. Since the diagonal block is full rank, there is no increase in rank and the block remains dense. Next, the method loops through each level from leaf to root, and all left branches in the path from D_i to the root have their $W_{(l,b)}$ factor multiplied by the Householder transformation. The application of Householder transformations to the right off-diagonal blocks is much the same as to $\hat{D}_i$. The W factor becomes $\hat{W}_{(l,b)} = Q_i^{\mathrm{T}} W_{(l,b)}$, and the block in turn is $\hat{W}_{(l,b)}X^{\mathrm{T}}_{(l,b)} = Q_i^{\mathrm{T}} W_{(l,b)} X^{\mathrm{T}}_{(l,b)}$. The upper right off-diagonal blocks are of low rank, and there will be some increase in the ranks after Householder transformation are applied. In Figure 3.6(c), the pale shaded areas have not had Householder transformations applied to them. However, in Figure 3.6(d), the dark shaded areas illustrate where the Householder transformations were applied in blocks $\hat{D}_7$ and $\hat{W}_{(3,4)}X^{\mathrm{T}}_{(3,4)}$.

Now begins the DWX repartitioning procedure for the diagonal and right off-diagonal blocks which is distinctly different from the $U_{(l,b)}V_{(l,b)}$ selection for the lower left off-diagonal blocks. Diagonal block $\hat{D}_i$ will have the first s rows tagged for repartitioning and housed in D_r, where s is the rank from the basis in Section 3.3. If Q_i^{T} is applied to the first m_b rows of any $W_{(l,b)}$ factors to the right of D_i, then the first s rows of $\hat{W}_{(l,b)}$ are tagged for repartitioning in W_{r_i} where $i = 1, \ldots, d-q$. Therefore, repartitioning will involve only D_r, or D_r plus $W_{r_1}, \ldots, W_{r_{d-q}}$ blocks. If only block D_r is to be repartitioned, then D_r is vertically merged with $X^{\mathrm{T}}_{(q,k)}$. The size of I is $s \times s$ and amends with the $W_{(q,k)}$ factor. This process will maintain the orthogonality of the W factors. Then the dense block D_r is vertically merged

with $X^{\mathrm{T}}_{(q,k)}$ where $W_{(q,k)}$ and $X^{\mathrm{T}}_{(q,k)}$ are from the D_{i-1} region above. Thus,

$$\tilde{W}_{(q,k)}\tilde{X}^{\mathrm{T}}_{(q,k)} = \begin{bmatrix} W_{(q,k)} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} X^{\mathrm{T}}_{(q,k)} \\ D_r \end{bmatrix}. \tag{3.18}$$

In the case where multiple blocks are to be repartitioned directly above, then D_r and $W_{r_1}X^{\mathrm{T}}_{r_1}$ to $W_{r_{d-q}}X^{\mathrm{T}}_{r_{d-q}}$ are appended together forming a single block before merging. A new $\tilde{D}_r$ is created such that

$$\tilde{D}_r = \left[\begin{array}{c|c|c|c} D_r & W_{r_1}X^{\mathrm{T}}_{r_1} & \cdots & W_{r_{d-q}}X^{\mathrm{T}}_{r_{d-q}} \end{array}\right].$$

The combined block $\tilde{D}_r$ is vertically merged with $W_{(q,k)}X^{\mathrm{T}}_{(q,k)}$ in the D_{i-1} region above. Similar to Equation 3.18, we see that

$$\tilde{W}_{(q,k)}\tilde{X}^{\mathrm{T}}_{(q,k)} = \begin{bmatrix} W_{(q,k)}X^{\mathrm{T}}_{(q,k)} \\ I \quad \tilde{D}^{\mathrm{T}}_r \end{bmatrix}.$$

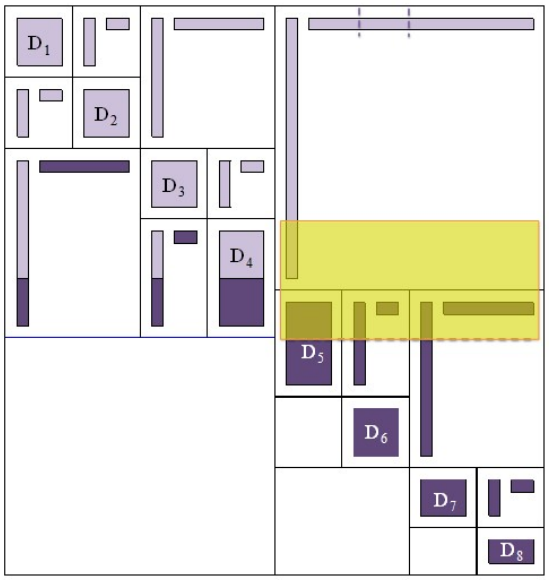
A diagram of $D_5, W_{(3,3)}X^{\mathrm{T}}_{(3,3)}$, and $W_{(2,2)}X^{\mathrm{T}}_{(2,2)}$ being repartitioned up into $W_{(1,1)}X^{\mathrm{T}}_{(1,1)}$ is shown in Figure 3.8. On the left, Figure 3.8(a) has the tagged area for merging highlighted in yellow. In this example,

$$\tilde{D}_r = \left[\begin{array}{c|c|c} D_r & W_{r_1}X^{\mathrm{T}}_{r_1} & W_{r_2}X^{\mathrm{T}}_{r_2} \end{array}\right]$$

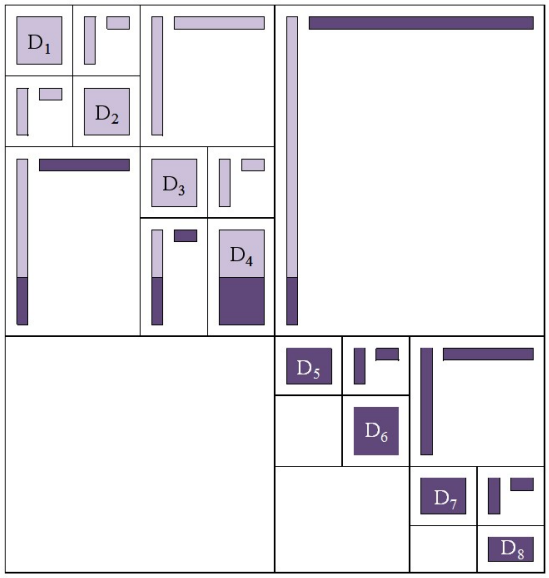
where $W_{r_1}X^{\mathrm{T}}_{r_1}$ is in block $W_{(3,3)}X^{\mathrm{T}}_{(3,3)}$ and $W_{r_2}X^{\mathrm{T}}_{r_2}$ is in block $W_{(2,2)}X^{\mathrm{T}}_{(2,2)}$. The merge of these blocks into $W_{(1,1)}X^{\mathrm{T}}_{(1,1)}$ above, results in

$$\tilde{W}_{(1,1)}\tilde{X}^{\mathrm{T}}_{(1,1)} = \begin{bmatrix} W_{(1,1)} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} X^{\mathrm{T}}_{(1,1)} \\ D_r \end{bmatrix}.$$

Figure 3.8(b) displays the completed repartitioning of the two areas. The dark shaded areas are where Householder transformations have been applied, the pale shaded regions have not



(a) The yellow highlights the area for repartitioning after Q_5^{T} has been applied to blocks $\hat{D}_5, \hat{W}_{(3,3)}X_{(3,3)}^{\mathrm{T}}$, and $\hat{W}_{(2,2)}X_{(2,2)}^{\mathrm{T}}$. The dark shaded factors on the right are to be merged above. Factors $D_r, W_{r_1}X_{r_1}^{\mathrm{T}}$, and $W_{r_2}X_{r_2}^{\mathrm{T}}$ joined together to form $\tilde{D}_r$.



(b) The dark shaded portions to the right of and including D_5 have been merged. It is easy to see where I and $\tilde{D}_r^{\mathrm{T}}$ are now merged above into $W_{(1,1)}X_{(1,1)}^{\mathrm{T}}$. All affected blocks are now repartitioned.

Figure 3.8. Repartitioning of $\hat{D}_5, \hat{W}_{(3,3)}X_{(3,3)}^{\mathrm{T}}$, and $\hat{W}_{(2,2)}X_{(2,2)}^{\mathrm{T}}$ up into $W_{(1,1)}X_{(1,1)}^{\mathrm{T}}$.

been compressed, and the white blocks have been reduced to zero. Figure 3.9 displays the matrix before, then after the row compression and repartitioning.

3.8 Summary of Row Compression Algorithm

The proposed row compression algorithm of the hierarchical representation of rank structured matrices contains many intricate and some interdependent algorithms. The encompassing control algorithm directs the flow of and interaction between the various data structures and procedures. It is a given that the row compression algorithms operates on a hierarchical representation of a rank structured matrix with a bidirectional binary tree and factored off-diagonal blocks. The control algorithm moves to last diagonal block to prepare

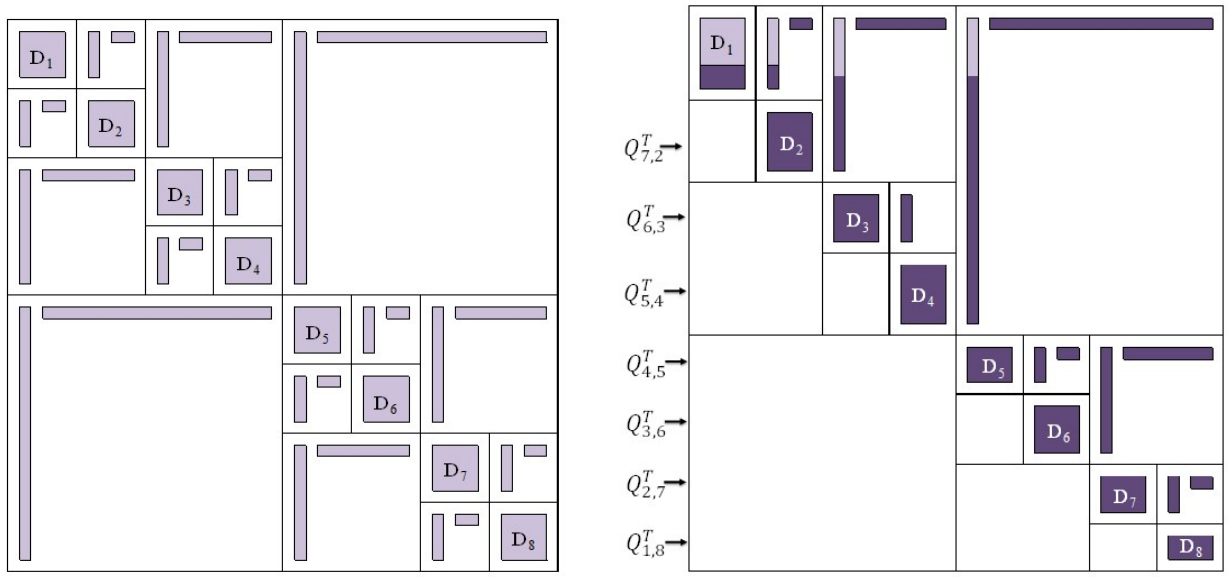


Figure 3.9. Diagram of a block 8×8 example displaying the hierarchical representation of a rank structured matrix before and after the row compression and repartitioning. The left matrix is prior to compression and is used for comparison to the compressed matrix on the right. The matrix on the right is compressed with the application of Householder transformations, Q_i^{T} where $i = 8, \ldots, 2$. All lower left off-diagonal blocks are white, and zeros were introduced into these blocks. The light shaded areas represent the nonzero portions of the factors in the off-diagonal blocks and the dense diagonal blocks. The dark shaded areas are where Householder transformations were applied and repartitioning took place.

for recursive compression. The preliminary compression stage begins with traversing the binary tree and locating the left off-diagonal blocks associated with diagonal block D_i. The collection of factors procedure then accesses the nodes pulling the $U_{(l,b)}$ and $V_{(l,b)}$ factors to produce a new data structure $Collection_{D_i}$ required for the basis. The basis formation procedure manipulates components of $Collection_{D_i}$ to yield a basis, $\mathscr{B}_i$, that is a subset of the larger vector space. Householder vectors are computed from the basis, $\mathscr{B}_i$, using QR factorization. The Householder transformation, Q_i^{T}, is inserted into the bidirectional linked list data structure for compression of the larger off-diagonal blocks. At this point, compression of the left off-diagonal blocks can take place.

The goal is to efficiently compress the left off-diagonal blocks in the diagonal block D_i region. Given off-diagonal block $B_{(q,k)}$, at level q and block number k, containing factor $U_{(q,k)}$. Compressing $U_{(q,k)}$ is an efficient way to compress block $B_{(q,k)}$. Therefore, $U_{(q,k)}$ is multiplied by Q_i^{T} which results in

$$Q_i^{\mathrm{T}} U_{(q,k)} = \begin{bmatrix} \tilde{U}_{(q,k)} \\ 0 \end{bmatrix}.$$

Once all the $U_{(l,b)}$ factors in the D_i region are compressed, then the last $U_{(l,b)}$ and $V_{(l,b)}$ factors adjacent to D_i are repartitioned. The repartition procedure is complex and involves merging two pairs of factors together. The merge procedure uses an LQ factorization and implements an efficient SVD to accomplish the merge. The columns of $V_{(l,b)}^{\mathrm{T}}$ are partitioned to align with the blocks above, then the tagged part of U_r and subblock $V_{R_i}^{\mathrm{T}}$ are merged into the off-diagonal and diagonal blocks that lay directly above. The last phase of compression is to apply the Householder transformations to diagonal block D_i and the right off-diagonal blocks $W_{(q,k)} X_{(q,k)}^{\mathrm{T}}$. The rank of D_i does not increase, and the ranks of the right off-diagonal blocks may see some bounded increase. After D_i and the right off-diagonal blocks are updated by Householder transformations, they are also repartitioned. However, the tagged parts of D_i and any additional $W_{(q,k)} X_{(q,k)}^{\mathrm{T}}$ blocks are appended to form a single block. The now single pair of factors is merged with the right off-diagonal block above.

The compression algorithm is recursive and continues until the second diagonal block, D_2, is reached, and the algorithm terminates after completing that compression. The end result is that the left off-diagonal blocks are compressed. The algorithm returns the compressed matrix C, and the Householder vector linked list. The quasiseparable matrix A has been row compressed to form $A = QC$, and C is ready for decomposition by the nested product algorithm in Chapter 4.

CHAPTER 4

NESTED PRODUCT DECOMPOSITION OF A RANK STRUCTURED MATRIX

The row compression algorithm in Chapter 3 introduces zeros in the lower left off-diagonal blocks of matrix A in a sequential sweep up from the bottom of the matrix. Given the row compression $A = QC$ in Chapter 3 where A is a matrix, Q is a sequence orthogonal transformations, and C is the row compression of A. The resulting off-diagonal block-compressed matrix C, shown in Figure 4.1, is used to extend the new sparse orthogonal nested product decomposition for quasiseparable matrices presented in the paper by Bella, Olshevsky and Stewart [3]. There is a distinct difference between the factorization of A that is a product of matrices, $A = UBV^T$ where B is banded and lower triangular, and the nested product decomposition of A that is a representation using simple matrices in combination to form sums and products.

The nested UBV decomposition by Bella, Olshevsky and Stewart begins with a matrix C that has a banded structure in the lower triangular part, and a quasiseparable structure in the upper triangular part. The decomposition creates two sequences of unitary transformations U_{k_j} and V_{k_j}, and two sequences of matrices B_{k_j} and D_{k_j}. A single stage in the nested decomposition has the components in the above sequences form the equation

$$D_{k_{j+1}} = U_{k_j}^{\mathrm{T}} D_{k_j} V_{k_j} - B_{k_j}.$$

The UBV decomposition of C is

$$C = U_{k_0}\left(B_{k_0} + U_{k_1}\left(B_{k_1} + \ldots + U_{k_{p-1}}\left(B_{k_{p-1}} + D_{k_p}\right)\ldots\right)\right) V_{k_{p-1}}^{\mathrm{T}} V_{k_1}^{\mathrm{T}} V_{k_0}^{\mathrm{T}} \tag{4.1}$$

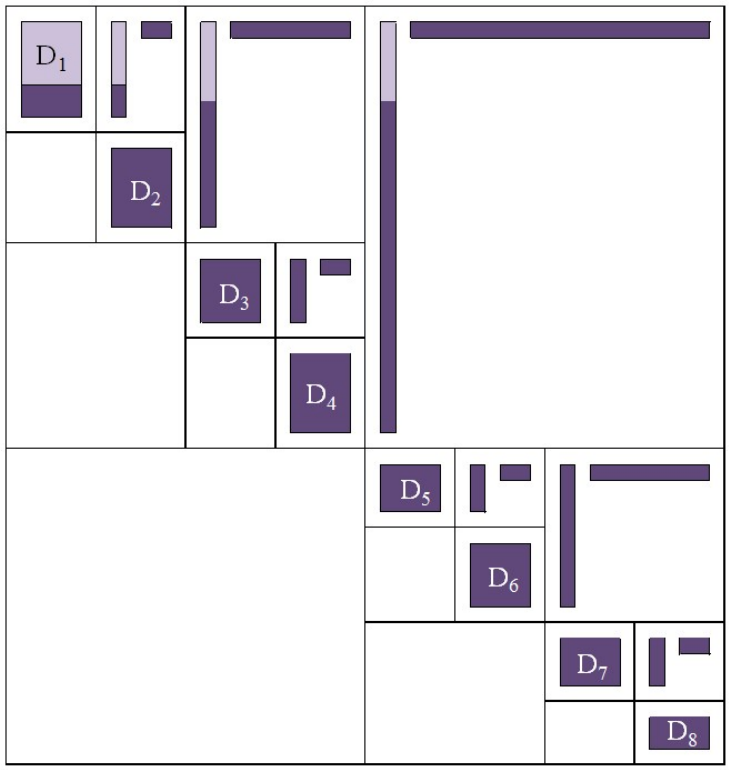


Figure 4.1. Diagram of a block 8×8 example of compressed matrix C from the hierarchical representation of a rank structured matrix A where $A = QC$, and Q represents the sequence of compression transformations. White off-diagonal blocks are where zeros have been introduced. Shaded blocks represent the nonzero blocks either as factors or full rank dense blocks.

where $0 = k_0 < k_1 < \ldots < k_p$ is an increasing sequence. The matrices B_{k_j} are a sequence of banded lower triangular matrices with only a few nonzero columns, and have form

$$B_{k_j} = \begin{bmatrix} I_{k_j \times k_j} & 0 & 0 \\ 0 & B^{(k_j)}_{k_j,11} & 0 \\ 0 & B^{(k_j)}_{k_j,21} & 0_{n-d_j-k_j \times n-d_j-k_j} \end{bmatrix} \tag{4.2}$$

for $j = 1, \ldots, p$. The matrix D_{k_p} is a sequence of zero-bordered matrices, and have the form

$$D_{k_j} = \begin{bmatrix} 0_{k_{jp} \times k_{jp}} & 0 \\ 0 & D^{(k_{jp})}_{k_{jp},22} \end{bmatrix} \tag{4.3}$$

for $j = 1, \ldots, p$ such that the superscript (k_{j_p}) indicates the size of the leading principal submatrix, and the small size of $D^{(k_{jp})}_{k_{jp},22}$ makes exploiting the rank structure unnecessary. This is the point at which the algorithm terminates. Both U_{k_j} and V_{k_j} sequences of orthogonal transformations of the form

$$U_{k_j} = \begin{bmatrix} I_{k_j} & 0 & 0 \\ 0 & U^{(k_j)}_{k_j,22} & 0 \\ 0 & 0 & I_{n-k_j-\delta_j} \end{bmatrix}, \text{ and } \quad V_{k_j} = \begin{bmatrix} I_{k_j} & 0 & 0 \\ 0 & V^{(k_j)}_{k_j,22} & 0 \\ 0 & 0 & I_{n-k_j-\delta_j} \end{bmatrix} \tag{4.4}$$

for $j = 1, \ldots, p$ such that $U^{(k_j)}_{k_j,22}$ and $V^{(k_j)}_{k_j,22}$ are $\delta_j \times \delta_j$. The pattern of two-sided unitary transformations exploited in [3] were first used to take advantage of rank structured matrices in a succession of papers [13, 18, 60]. A key feature of the decomposition in [3] has the row compression introducing a band structure in the lower triangular part of A, which results in a sparse orthogonal decomposition of A. The nested UBV decomposition is stable, and can be computed from a general rank structured matrix using $O(n^2)$ operations, which is comparable to the computation of a generator representation of a quasiseparable matrix represented by n^2 matrix elements.

The connection between [3] and our work is that both begin with a rank structured matrix A and use compression methods to form $A = QC$. Our work, however, starts with a hierarchical representation of A, then via a row compression conversion algorithm transforms C into a nested UBV decomposition. Whereas in [3], the row compression forms the nested product representation directly from A. Once half of the decomposition is done with the computation of matrix C, the next stage is to perform a sweep down the matrix to complete the nested UBV decomposition operating on blocks of data.

The row compression algorithm methods and procedures, outlined in Chapter 3, are amended to accommodate direction, orientation, and region changes required to compute a nested UBV decomposition. An outline of the major procedures in the hierarchical UBV decomposition algorithm are:

Hierarchical Nested UBV Decomposition Algorithm

1. Perform a top-down sequential UBV decomposition by acting on the rows associated with each diagonal block, and all columns to the right of and including the diagonal block.

2. Form a basis from a collection of the W and X factors to the right of each diagonal block, and let the rank, s, of the basis determine the size of the repartition.

3. Repartition the right off-diagonal blocks that fall directly below the diagonal block, and merge the nonzero rows into the blocks below.

4. Update the diagonal block by applying the Householder transformations to the diagonal block from the left.

5. Compute an LQ factorization of the diagonal block, forming B and V of the nested product.

The result, after applying the hierarchical UBV decomposition algorithm to C, is a nested product representation of the quasiseparable matrix [3]. The matrix,

$$B = B_{k_1} + \cdots + B_{k_{p-1}} + D_{k_p},$$

will look like a banded diagonal matrix [76] similar to

$$\begin{bmatrix} \times & 0 & 0 & 0 & 0 & 0 & 0 \\ \times & \times & 0 & 0 & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & 0 & 0 & \times & \times \end{bmatrix}.$$

The main difference between the UBV decomposition in [3] and the one presented here is that we start with a hierarchical representation and entire blocks are decomposed into the nested product representation, not just a single vector at a time. The nested product notation is somewhat the same, and is amended to reflect that blocks are being transformed instead of vectors. Two important notation clarifications for Chapter 4 must be made: 1) The factors $U_{(l,b)}$ and $V_{(l,b)}$ represent the factorization of lower off-diagonal blocks in compressed matrix C, and U_k and V_k indicate sequence of Householder transformations in nested UBV decomposition. 2) The set $\mathscr{B}_k$ denotes the basis for the column space of the low rank blocks, whereas B_k represents the lower triangular diagonal block in the nested UBV decomposition.

4.1 Algorithms for Nested UBV Decomposition

The UBV nested product algorithm begins with a hierarchical representation of the compressed matrix C that is obtained from the hierarchically partitioned quasiseparable matrix A, as seen in Figure 4.1. The control algorithm utilizes the binary tree structure of C and creates a new linked list UBV data structure. Each node of the UBV comprehensive data structure will house each of the U_i and V_i sequences of orthogonal transformations, and the B_i lower triangular matrix associated with the corresponding diagonal block, D_i. The

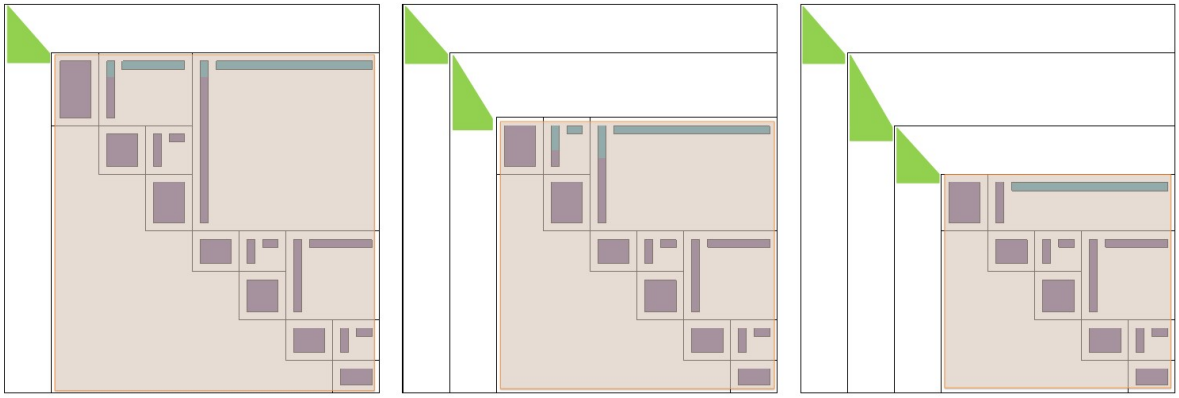


Figure 4.2. Illustration of a block 8×8 nested product decomposition. From left to right, the shaded areas in each of the decompositions in the sequence are nested within the current decomposition of the diagonal block.

same binary tree traversal routines that are implemented in the row compression algorithm are also used in the nested product algorithm to access data in the tree. The nested product control algorithm is recursive, and at each recursive call adds a node to the UBV data structure. These navigation methods are presented in detail in Section 3.2.

The control algorithm for the nested product decomposition starts at the first diagonal block, D_1, and recursively decomposes matrix C until it reaches the last diagonal block. This movement creates a nesting effect where everything to the right and below the current diagonal block, D_i, is encapsulated in a hierarchical parameterization as seen in Figure 4.2. Everything in the upper left is represented by a banded matrix B. The algorithm uses D_i as the key for where the UBV decomposition operates on matrix C. The decomposition acts on the rows in and to the right of each D_i and forms a basis from the off-diagonal blocks to the right of each diagonal block. As the algorithm navigates through the binary tree for C, it introduces zeros into the off-diagonal blocks to the right of D_i. All elements directly below the diagonal block already contain zeros from the previous row compression algorithm. The decomposition is accomplished in two stages: the introduction of zeros into the right off-diagonal region with $W_{(l,b)}$ and $X_{(l,b)}$ factors using a transformation U_k, followed

by the reduction of the diagonal block into a lower triangular block using transformation V_k. Repartitioning does occur in the blocks between the two stages, and the processes to repartition blocks are the same as described in Section 3.6.

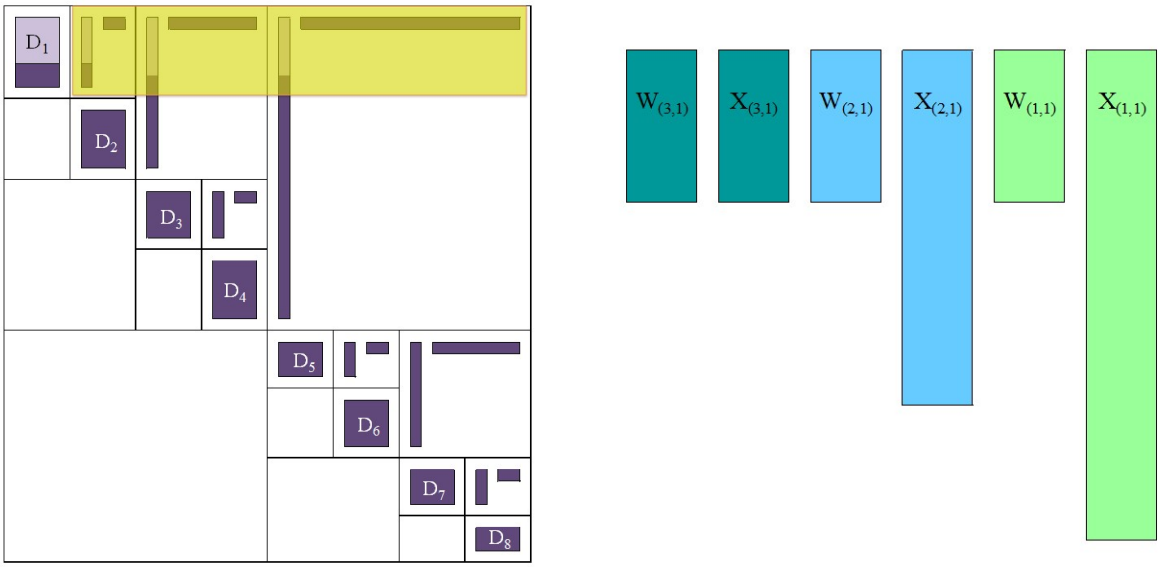
The selection of the off-diagonal blocks for the basis formation associated with diagonal block D_i is determined by two equivalent characteristics: the off-diagonal block must lie strictly to the right of D_i, and the off-diagonal block should be a left branch in the binary tree path for D_i. The column space for the basis is determined by the appropriate $W_{(l,b)}$. The rows in D_i are identified, and determine which rows in each $W_{(l,b)}$ factor of the off-diagonal blocks will be in the column space. As the off-diagonal blocks are selected, only the pertinent rows in each W are collected. All columns and rows of each $X_{(l,b)}$ are collected. Hence, the size of the collection is just a few rows tall and across all columns strictly to the left of the diagonal block. The collection of factors for D_1, both in the matrix and as the $Collection_{D_1}$, are illustrated in the block 8×8 example in Figure 4.3.

The basis $\mathscr{B}_i$ is a set of linearly independent vectors which span the given subspace. The basis procedure is exactly the same as presented in Section 3.3 where Householder transformations are utilized in an LQ factorization of the $Collection_{D_1}$. The number of columns in each factor, $W_{(l,b)}$ and $X_{(l,b)}$, are based on the rank of the lower triangular blocks. This basis formation procedure isolates the $W_{(l,b)}$ column factors that are relevant to determining the column space of the region of C into which zeros are to be introduced. An LQ factorization method is applied to each $X_{(l,b)}$ factor in $Collection_{D_i}$ to reduce the size of each $X_{(l,b)}$. The LQ factorization of $X_{(l,b)}$ yields a reduced matrix which is then transposed into lower triangular block form

$$\mathscr{B}_1 = \left[\begin{array}{cc|cc|cc} W_{(3,1)} \cdot L_{(3,1)} & 0 & W_{(2,1)} \cdot L_{(2,1)} & 0 & W_{(1,1)} \cdot L_{(1,1)} & 0 \end{array}\right]. \tag{4.5}$$

.

An efficient SVD factorization operates on the intermediate matrix for the basis by first performing a QR decomposition in order to obtain the factor R. Figure 4.3(b) displays



(a) The 8×8 block example highlights the collection of factors for diagonal block D_1, $Collection_{D_1}$, in the compressed matrix C.

(b) The factor data structure in the 8×8 block example for $Collection_{D_1}$ of the off-diagonal blocks $W_{(3,1)}, X_{(3,1)}, W_{(2,1)}, X_{(2,1)}, W_{(1,1)}$, and $X_{(1,1)}$.

Figure 4.3. Illustration of a block 8×8 formation of basis $\mathscr{B}_i$ from the $W_{(l,b)}$ and $X_{(l,b)}$ factors associated with diagonal block D_i.

$Collection_{D_1}$ that is used to compute the basis $\mathscr{B}_1$. The sequence of Householder vectors from Q are stored in a linked list. The orthogonal matrix U, from the SVD of R, is reduced to only the first r columns, where r is the rank the R, resulting in $\tilde{U} = U(:, 1 : r)$. The final step is to apply the Householder transformations from the linked list to $\tilde{U}$, producing a basis, $\mathscr{B}_i$, for $Collection_{D_i}$. The basis procedure method returns the basis, $\mathscr{B}_i$, and basis rank, s. The basis and rank are used to compute unitary transformations that introduce zeros in the upper right off-diagonal blocks of the larger matrix C.

4.2 Forming U of UBV and Reducing Upper Right

The basis $\mathscr{B}_i$ is a reduced form of a collection of factors associated with each diagonal block D_i discussed in Section 4.1. The Householder vectors computed from the reduced matrix basis, $\mathscr{B}_i$, are equivalent to those computed from the larger submatrix $Collection_{D_i}$

and the compressed matrix C. The sequence of Householder vectors computed from $\mathscr{B}_i$ forms the U_k from the UBV decomposition. A new nested product class data structure is created to harbor each of the three data structures in the nested UBV decomposition. In this stage, the sequence of Householder vectors are stored in a bidirectional linked list data structure, U_i^{T}, along with the offsets for the location of submatrix $Collection_{D_i}$. The procedure for computing the sequence of Householder vectors performs the QR factorization of $\mathscr{B}_i$, and is detailed in Section 3.4. However, the QR factorization places zeros in the lower left,

$$\mathscr{B}_i = Q \begin{bmatrix} R \\ 0 \end{bmatrix},$$

and the nested product requires zeros placed in the upper right.

If the basis $\mathscr{B}_i$ is flipped upside down prior to computing the Householder vectors, then the unitary transformations are essentially introducing zeros in the top rows of the basis. Thus, the permutation, P, is applied to the basis, $\mathscr{B}_i$, resulting in

$$P\mathscr{B}_i = \breve{\mathscr{B}}_i, \quad P\breve{\mathscr{B}}_i = \mathscr{B}_i, \qquad \text{and} \qquad PP = I$$

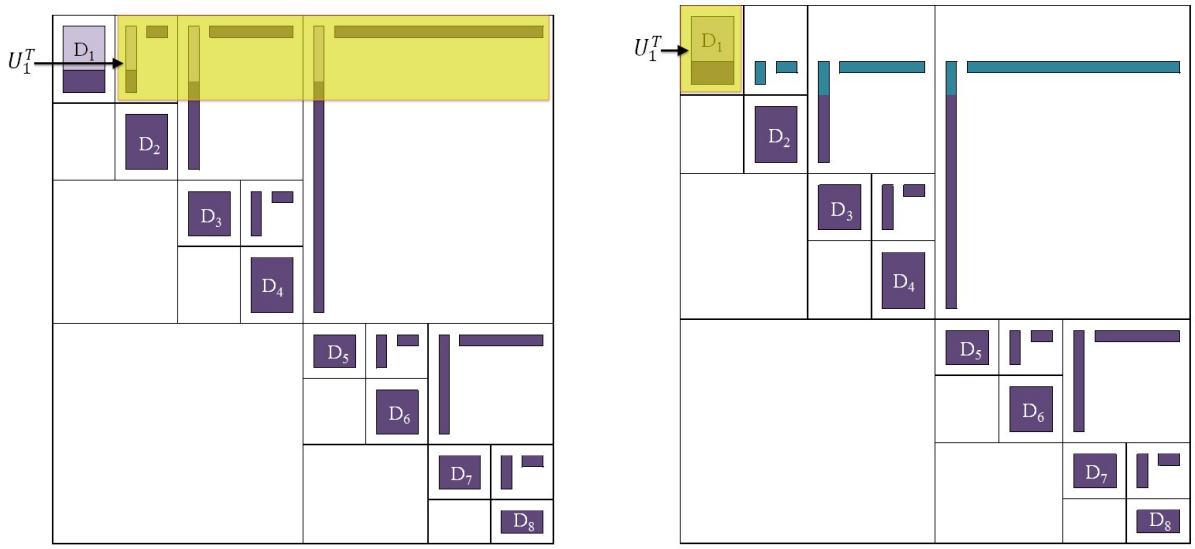
where $\breve{\mathscr{B}}_i$ is the flipped matrix $\mathscr{B}_i$. The QR factorization of $\breve{\mathscr{B}}_i$ in this section is notated as

$$\breve{\mathscr{B}}_i = \breve{U}_i \begin{bmatrix} R \\ 0 \end{bmatrix} = \breve{U}_{i,1}\breve{U}_{i,2}\ldots\breve{U}_{i,k} \begin{bmatrix} R \\ 0 \end{bmatrix},$$

where the sequence of Householder transformations is represented by $\breve{U}_i = \breve{U}_{i,1}\breve{U}_{i,2}\ldots\breve{U}_{i,k}$ such that i refers to the diagonal block D_i and $1 \leq k \leq \mathrm{rank}(\mathscr{B}_i)$. Next, each Householder vector is permuted, flipping them upside down, producing

$$(P\breve{U}_{i,1}P)(P\breve{U}_{i,2}P)\ldots(P\breve{U}_{i,k}P) = PU_{i,1}U_{i,2}\ldots U_{i,k}P.$$

Hence, when the Householder sequence, U_i, is applied to $\mathscr{B}_i$ zeros are introduced into the



(a) The yellow highlights the area of where the sequence of Householder transformations, U_1^{T}, are applied to the $W_{(3,1)}, W_{(2,1)}$, and $W_{(1,1)}$ factors. All white rectangles denotes zeros.

(b) The yellow highlights the area of where U_1^{T} is applied next to the D_1 block. The teal shaded areas show where the $W_{(3,1)}X_{(3,1)}^{\mathrm{T}}$, $W_{(2,1)}X_{(2,1)}^{\mathrm{T}}$, and $W_{(1,1)}X_{(1,1)}^{\mathrm{T}}$ have been reduced. The purple shaded areas have not been affected by U_1^{T}.

Figure 4.4. An example of a block 8×8 matrix where the Householder transformations, U_1^{T} in the UBV decomposition, are applied from the left to the right off-diagonal blocks $W_{(3,1)}X_{(3,1)}^{\mathrm{T}}$, $W_{(2,1)}X_{(2,1)}^{\mathrm{T}}$, and $W_{(1,1)}X_{(1,1)}^{\mathrm{T}}$ associated with diagonal block D_i.

upper rows of the matrix,

$$U_i^{\mathrm{T}}\mathscr{B}_i = \begin{bmatrix} 0 \\ L \end{bmatrix}.$$

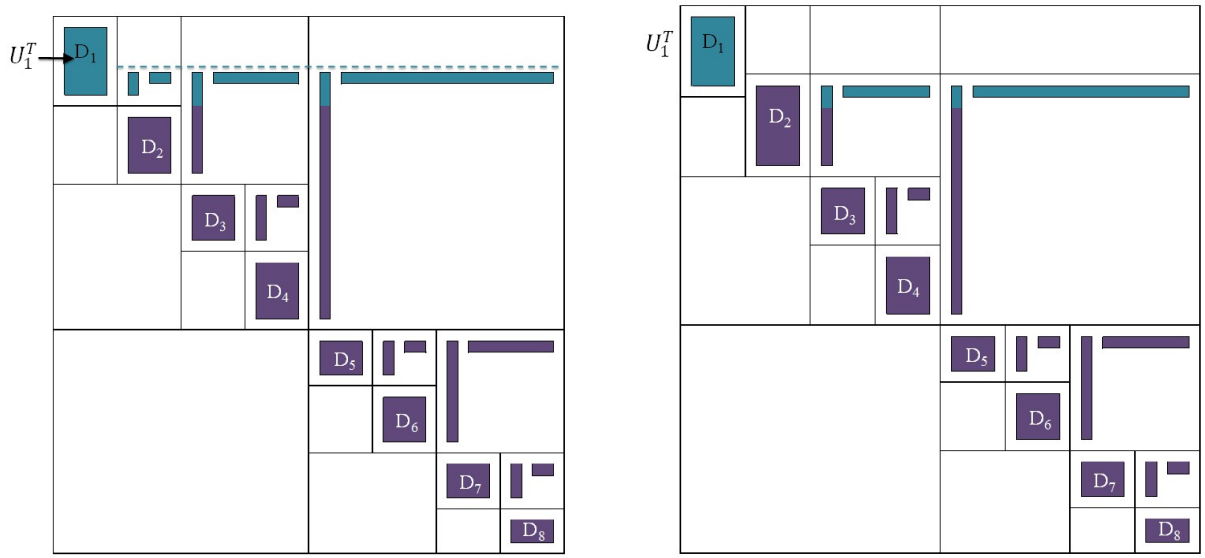
The computed Householder linked list from the smaller basis matrix $\mathscr{B}_i$ will compress the upper right off-diagonal blocks of matrix C corresponding to $Collection_{D_i}$. The permuted Householder transformations, $U_{i,1}U_{i,2}\ldots U_{i,k}$, are applied from the left to the $W_{l,b}$ factor from $Collection_{D_i}$, in order to introduce zeros into the top nonzero rows of the right off-diagonal blocks. An 8×8 block matrix example of this procedure is illustrated for diagonal block D_1 in Figure 4.4. The compression algorithm for $W_{(l,b)}$ and $X_{(l,b)}$ is essentially the same as the one outlined in Section 3.6, it just operates on a different set of factors. Once the off-diagonal

blocks have zeros in the top rows, U_i^{T} is then applied to diagonal block D_i, $\hat{D}_i = U_i^{\mathrm{T}} D_i$, completing the Householder transformation across matrix C, and the matrix is ready for repartitioning at diagonal block $\hat{D}_i$. The Householder linked lists, U_i^{T}, for each $Collection_{D_i}$, are stored in the comprehensive UBV data structure for access by the fast solver. The UBV complex data structure is a bidirectional linked list that houses data structures for linked lists of Householder vector sequences, collects index information for diagonal block $\hat{D}_i$, tracks the ranks of the all the off-diagonal blocks, and stores the data for $\hat{D}_i$.

As the compression algorithm operates on $Collection_{D_i}$, it also tags the rows of the factors needed for repartitioning with the blocks directly below $Collection_{D_i}$. The number of nonzero rows tagged for repartitioning is based on the rank, s, of the basis, $\mathscr{B}_i$, computed from the right off-diagonal blocks. Assume $\hat{D}_i$ has size $m_d \times n_d$ where $n_d \leq m_d$, then the elements in the top $m_d - s$ rows of $Collection_{D_i}$ contain zeros, and the bottom s nonzero rows are tagged for repartitioning. The repartitioning merge procedure is exactly the same as the one presented in Section 3.6.

4.3 Computing B and V Sequences of the UBV

Repartitioning is the next step in the decomposition of matrix C. Essentially, the process combines two pairs of off-diagonal factors into a single pair utilizing a merge method detailed in Section 3.6. The block and factor selection procedure is slightly different from the one in Section 3.6. The number of right off-diagonal blocks involved ranges from 1 to $d-1$ where d is the depth, and what determines the number involved is where it is located in the binary tree in addition to whether it is associated with an odd or even diagonal block. The rank of basis $\mathscr{B}_i, s$, and the bottom rows of each factor involved in repartitioning were set aside while computing the sequence of unitary transformations, U_i^{T}. A convenient pattern emerges and two routines are created for selecting off-diagonal blocks based on the current diagonal block, $\hat{D}_i$.



(a) The $\hat{D}_1$ block is teal shaded to denote where the sequence of unitary transformations U_1^{T} has been applied. The dashed line shows where the $W_{3,1}$ and $X_{3,1}$ factors will be merged into the D_2 block below, thus repartitioning D_2.

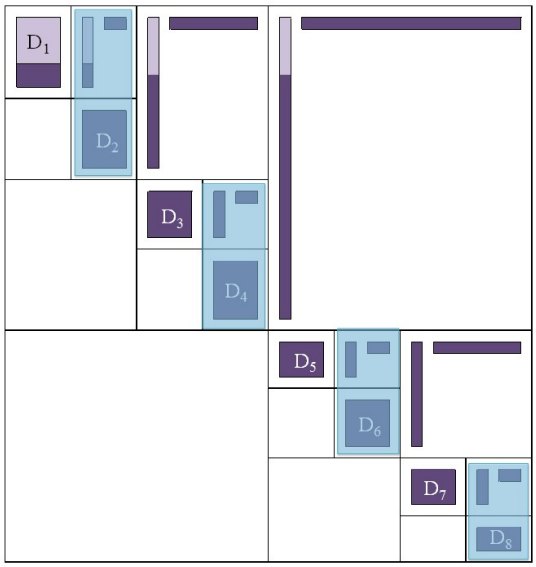
(b) The $\tilde{D}_2$ block has been repartitioned, and the right off-diagonal blocks are updated. Note that $W_{(2,1)}, W_{(1,1)}, X_{(2,1)}$ and $X_{(1,1)}$ are not merged, but not necessarily repartitioned since a block boundary has not been crossed.

Figure 4.5. Illustration of a block 8×8 example where U_1^{T} has been applied to diagonal block $\hat{D}_1$ and $Collection_{D_1}$ followed by the repartitioning of $W_{(3,1)}X_{(3,1)}^{\mathrm{T}}$ with diagonal block $\tilde{D}_2$ below. The top rows of off-diagonal blocks $W_{(2,1)}X_{(2,1)}^{\mathrm{T}}$, and $W_{(1,1)}X_{(1,1)}^{\mathrm{T}}$ contain zeros.

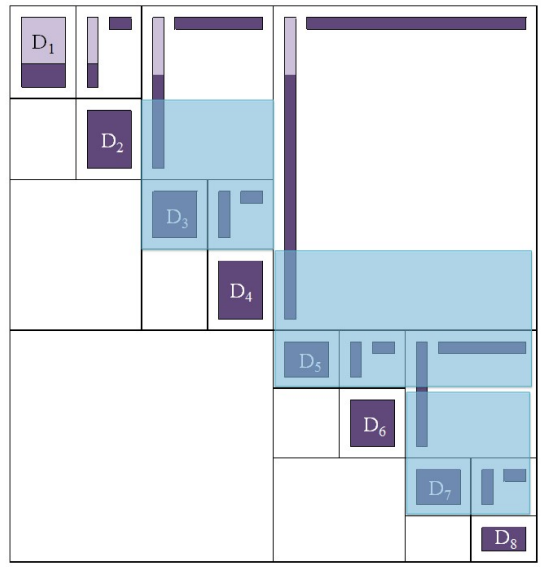
If i is odd, then only one block below $\hat{D}_i$ is involved in repartitioning and it is diagonal block D_{i+1}. Referring to Section 3.1, the depth of the binary tree is $d = \log_2(\frac{n}{m_b}) + 1$ where n is the size of matrix C and m_b is the original terminal block size. The right off-diagonal block at level $d-1$, associated with $\hat{D}_i$, is involved in repartitioning as shown in Figure 4.6(a). The tagged pair of factors, W_r and X_r, from this single off-diagonal block are merged with diagonal block D_{i+1} directly below. Thus, D_{i+1} becomes

$$\tilde{D}_{i+1} = \begin{bmatrix} W_r X_r^{\mathrm{T}} \\ D_{i+1} \end{bmatrix}.$$

An example of repartitioning for the first diagonal is seen in Figure 4.5.



(a) The odd diagonal blocks D_1, D_3, D_5, D_7 will always have level $= d-1$ off-diagonal blocks to the immediate right involved in repartitioning, and that repartitioning involves only the D_{i+1} block. Thus blocks $W_{(3,1)}X^{\mathrm{T}}_{(3,1)}, \ldots, W_{(3,4)}X^{\mathrm{T}}_{(3,4)}$ are repartitioned with D_2, D_4, D_6, D_8 respectively.



(b) The even diagonal blocks D_2, D_4, D_6 will always have the off-diagonal blocks repartition with multiple blocks below, including the D_{i+1} block. Block $W_{(l,b)}X^{\mathrm{T}}_{(l,b)}$ is repartitioned with all blocks below from level $= l+1, \ldots, d-1$, and D_{i+1}. Hence for D_4, $W_{(1,1)}X^{\mathrm{T}}_{(1,1)}$ is repartitioned with $D_5, W_{(3,3)}X^{\mathrm{T}}_{(3,3)}$, and $W_{(2,2)}X^{\mathrm{T}}_{(2,2)}$.

Figure 4.6. An 8×8 block diagram of the odd and even pattern for repartitioning. Only a single contiguous off-diagonal block to the right of D_i is involved in repartitioning. The left figure highlights the region to the right of each odd diagonal block that is involved in repartitioning. The right figure highlights the region to the right of each even diagonal block that is involved in repartitioning. A pattern emerges for efficient handling of the odd blocks.

If i is even, then multiple blocks below $\hat{D}_i$ are involved in repartitioning. The one right off-diagonal block, associated with $\hat{D}_i$, selected for repartitioning is located at the last left branch in the tree before reaching $\hat{D}_i$, and the level of this block is noted as k. This block will be referred to as $F_{(k,q)}$ where k represents the level and q is the block number. The tagged pair of factors, W_r and X_r, from this single off-diagonal block are merged with all right off-diagonal blocks associated with diagonal block D_{i+1} and at levels greater than k, and illustrated in Figure 4.6(b). The off-diagonal block $F_{(k,q)}$ spans multiple blocks directly below it, then factor X_r is partitioned to align with the columns in the blocks below. Prior

to merging, the blocks look like

$$\left[\begin{array}{c} W_r X_r^{\mathrm{T}} \\ D_{i+1} \left| W_{(d-1,b)} X_{(d-1,b)}^{\mathrm{T}} \right| \cdots \left| W_{(k+1,c)} X_{(k+1,c)}^{\mathrm{T}} \right. \end{array}\right]$$

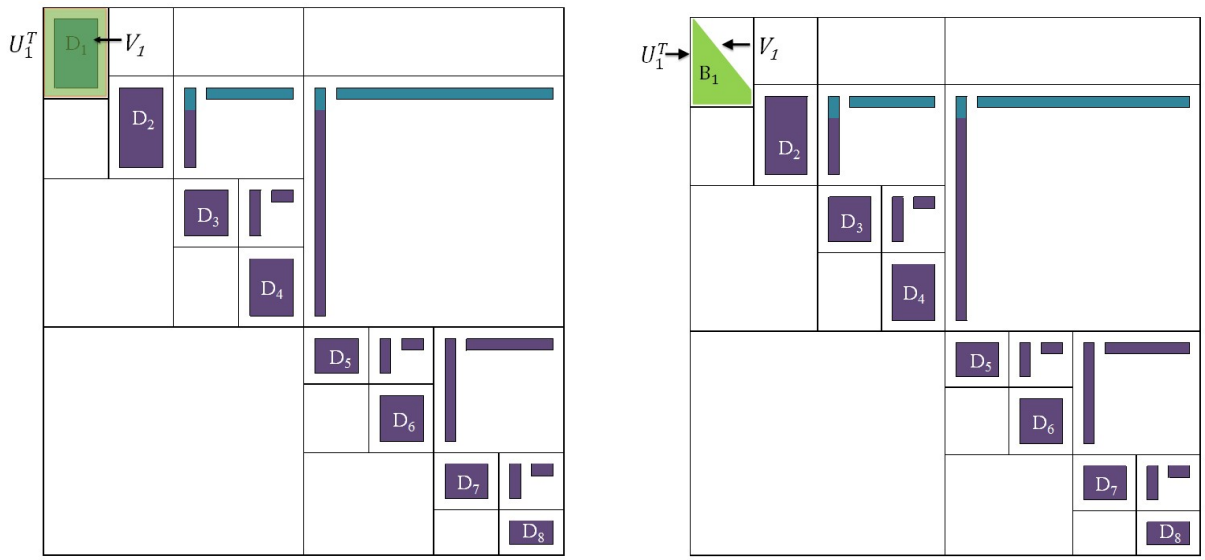
where the level of the off-diagonal blocks below range from $d-1$ to $k+1$.

At this juncture in the algorithm, all elements directly below and to the right of $\hat{D}_i$ are zero which is seen in Figure 4.5(b). An LQ factorization method is applied to $\hat{D}_i$ producing a lower triangular block and sequence of unitary transformations. The LQ factorization of $\hat{D}_i$ yields a reduced matrix,

$$\hat{D}_i = \left[\begin{array}{cc} B_i & 0 \end{array}\right] V_i$$

where V_i is the sequence of Householder vectors and B_i is the lower triangular block. An example of the BV decomposition is shown in Figure 4.7 for diagonal block $\hat{D}_1$. Both the V_i and B_i data structures are stored in the comprehensive UBV data structure.

Once the region associated with diagonal block D_i is decomposed and repartitioned, the control algorithm recursively moves to the next region below associated with $\tilde{D}_{i+1}$. Every subsequent decomposition is dependent on all the previous decompositions because of the repartitioning. When the algorithm reaches the last diagonal block, $\tilde{D}_{2^{d-1}}$, no sequence for U is computed because there is nothing to the right of the block. The last diagonal block is only decomposed into $B_{2^{d-1}}$ and $V_{2^{d-1}}$. The final UBV matrix decomposition is illustrated in Figure 4.8. The comprehensive UBV data structure is now ready to be input into the fast solver.



(a) The yellow highlights the area where an LQ factorization is computed on $\hat{D}_1$. The sequence of Householder transformations, V_1^{T}, are applied from the right to $\hat{D}_1$.

(b) The green highlights the reduction of $\hat{D}_1$ into a lower triangular block B_1. All white blocks have zeroes introduced.

Figure 4.7. An illustration for the nested product BV decomposition diagonal block, $\hat{D}_1$.

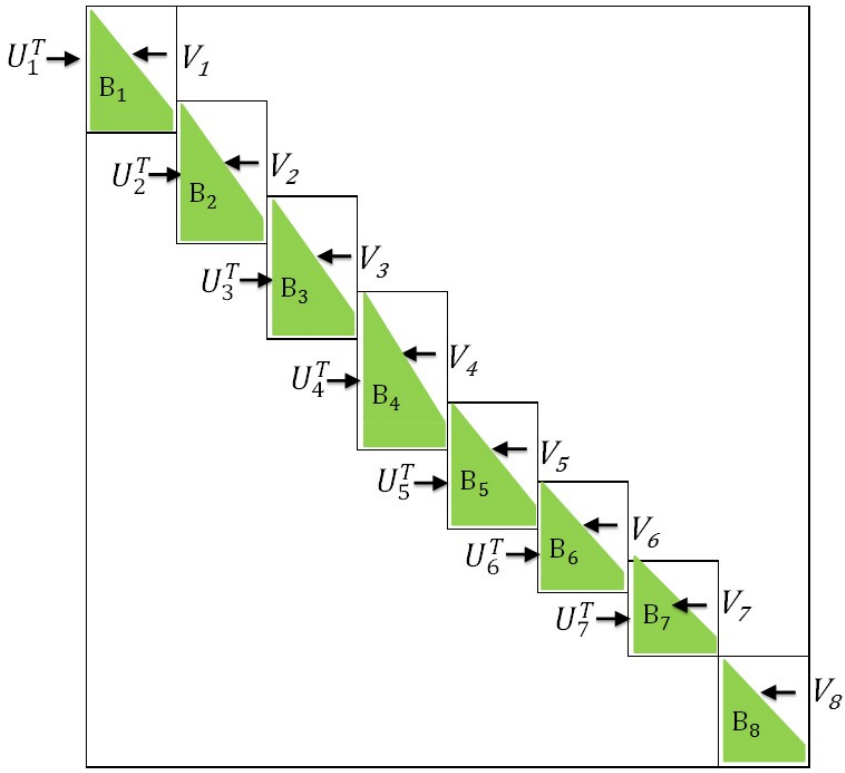


Figure 4.8. Diagram of a block 8×8 example displaying the hierarchical representation of a rank structured matrix after the row compression and nested product decomposition have been performed. The compression and decomposition algorithms have introduced zeros into all lower left and upper right off-diagonal blocks which are white. The green shaded areas represent the nonzero diagonal lower triangular blocks that are considered full rank and dense.

CHAPTER 5

FAST SOLVER AND APPLICATIONS

The fast nested product system solver in this research is the existing algorithm presented in [3], and will be extended to the hierarchical variant of the UBV decomposition. When the hierarchical representation of a quasiseparable matrix is compressed and decomposed into the UBV data structure, the fast, linear system, nested product solver can be used. The fast solver and matrix-vector product algorithms are important procedures operating on a data sparse matrix.

5.1 Matrix-Vector Multiplication

The nested UBV decomposition developed in Chapter 4, is used in the matrix-vector multiplication algorithm in [3]. The matrix-vector product algorithm operates directly on the nested product data structure. Given a quasiseparable matrix A and vector $\mathbf{x}$, compute $\mathbf{b} = A\mathbf{x}$. Matrix $A = QC$ where Q is the sequence of unitary transformations from the row compression in Chapter 3 and and C is the nested UBV decomposition $C = D_0 = R_0 V^{\mathrm{T}}$. From [3], we have

$$R_0 = U_{k_0}\left(B_{k_0} + U_{k_1}\left(B_{k_1} + \cdots + U_{k_{p-1}}\left(B_{k_{p-1}} + D_{k_p}\right)\right)\cdots\right), \quad \text{and} \quad V = V_{k_0}\cdots V_{k_{p-2}}V_{k_{p-1}}, \tag{5.1}$$

then the problem becomes $\mathbf{b} = QR_0V^{\mathrm{T}}\mathbf{x}$. A recurrence relation arises that can be used in the sequence of matrices R_j from the smaller nested products. Let $R_p = D_{k_p}$, and, for $j = 0, 1, \ldots, p-1$, let $R_j = U_{k_j}\left(B_{k_j} + R_{k_{j+1}}\right)$. The recurrence yields

$$R_j = U_{k_j}\left(B_{k_j} + U_{k_{j+1}}\left(B_{k_{j+1}} + \cdots + U_{k_{p-1}}\left(B_{k_{p-1}} + D_{k_p}\right)\cdots\right)\right). \tag{5.2}$$

The first step in the matrix-vector multiplication algorithm is to let

$$\mathbf{y} = V^{\mathrm{T}}\mathbf{x} = V_{k_{p-1}}^{\mathrm{T}} V_{k_{p-2}}^{\mathrm{T}} \cdots V_{k_0}^{\mathrm{T}}\mathbf{x}$$

then $\mathbf{b} = QR_0\mathbf{y}$. This leads directly into the second step $R_0\mathbf{y} = Q^{\mathrm{T}}\mathbf{b}$ where we let

$$\mathbf{z} = R_0\mathbf{y} = U_{k_0}\left(B_{k_0} + U_{k_1}\left(B_{k_1} + \cdots + U_{k_{p-1}}\left(B_{k_{p-1}} + D_{k_p}\right)\right)\cdots\right)\mathbf{y}. \tag{5.3}$$

Here, $\mathbf{y}_p = D_{k_p}\mathbf{y}$, $\mathbf{y}_{j-1} = U_{k_{j-1}}(\mathbf{y}_j + B_{k_{j-1}}\mathbf{y})$, and $\mathbf{y}_0 = R_0\mathbf{y} = \mathbf{z}$. The third and final step is $\mathbf{b} = Q\mathbf{z} = Q_1Q_2\ldots Q_r\mathbf{z}$. Due to the recurrence of the computation and the sparsity of the blocks, the complexity of the matrix-vector product multiplication is $O(n)$.

5.2 Nested Product Fast Solver

Given a quasiseparable matrix A, vector $\mathbf{b}$, and $A\mathbf{x} = \mathbf{b}$, then compute $\mathbf{x}$. The first step of the fast nested product solver is to take the system $R_0\mathbf{y} = Q^{\mathrm{T}}\mathbf{b}$ and form the system $\mathbf{b} = QR_0V^{\mathrm{T}}\mathbf{x}$. Let $\mathbf{y}$ and $\mathbf{c}$ be defined as follows

$$\mathbf{y} = V^{\mathrm{T}}\mathbf{x} = V_{k_{p-1}}^{\mathrm{T}} V_{k_{p-2}}^{\mathrm{T}} \cdots V_{k_0}^{\mathrm{T}}\mathbf{x}, \text{ and } \mathbf{c} = \mathbf{c}_0 = Q^{\mathrm{T}}\mathbf{b}$$

where $\mathbf{x}$ is efficiently computed from $\mathbf{y} = V\mathbf{x}$. Now the system $R_0\mathbf{y} = Q^{\mathrm{T}}\mathbf{b}$ looks like

$$U_{k_0}\left(B_{k_0} + U_{k_1}\left(B_{k_1} + \cdots + U_{k_{p-1}}\left(B_{k_{p-1}} + D_{k_p}\right)\right)\cdots\right)\mathbf{y} = Q^{\mathrm{T}}\mathbf{b} = \mathbf{c}_0, \tag{5.4}$$

and becomes $R_0\mathbf{y} = \mathbf{c}_0$.

The solver algorithm strips each stratum on the left-hand side of Equation 5.2, to focus on solving for a single $\mathbf{y}_j$ where $\mathbf{y}_j$ is $d_{j-1}\times 1$ for $j = 1, 2, \ldots, p$. Then the algorithm multiplies both sides of the equation by U_{j-1}. In the next step, the solver algorithm multiplies $\mathbf{c}_0 = R_0\mathbf{y}$

by $U_{k_0}^{\mathrm{T}}$ on both sides to solve the first layer of the nested product,

$$U_{k_1}\left(B_{k_1}+\cdots+U_{k_{p-1}}\left(B_{k_{p-1}}+D_{k_p}\right)\cdots\right)\mathbf{y}+B_{k_0}\mathbf{y}=U_{k_0}^{\mathrm{T}}\mathbf{c}_0=\begin{bmatrix}\mathbf{c}_{0,1}\\ \mathbf{c}_{0,2}\end{bmatrix},$$

yielding the first component of $\mathbf{y}$. Only the first column of B_{k_0} is nonzero, and the first row of R_1 is all zeros except for the $(1,1)$ element. Hence,

$$B_{k_0}\mathbf{y}=y_1B_{k_0}\mathbf{e}_1,\quad y_1=c_{0,1}/b_{0,1},\qquad\text{and}\qquad \mathbf{c}_1=U_0^{\mathrm{T}}\mathbf{c}_0-y_1B_{k_0}\mathbf{e}_1.$$

The first layer is peeled away, and $\mathbf{c}_1=R_1\mathbf{y}$ is the next system to solve for in the nested product.

The new system R_1 is bordered by a column and row of zeros. The smaller system is solved with the same process as R_0, and y_1 isn't involved. The remaining elements of $\mathbf{y}$ are found by repeating the procedure for the sequence of zero bordered systems. Recall from Section 5.1,

$$R_j=U_{k_j}\left(B_{k_j}+R_{k_{j+1}}\right)=U_{k_j}\left(B_{k_j}+U_{k_{j+1}}\left(B_{k_{j+1}}+\cdots+U_{k_{p-1}}\left(B_{k_{p-1}}+D_{k_p}\right)\right)\cdots\right).$$

In the subsequent systems, each $\mathbf{c}_j$ will be bordered by d_{j-1} rows and columns with

$$\mathbf{c}_j=R_j\mathbf{y}=U_{j-1}^{\mathrm{T}}\mathbf{c}_{j-1}-B_{j-1}\begin{bmatrix}\mathbf{y}_j\\ 0\end{bmatrix},$$

such that $\mathbf{y}_j$ is not required in any of the following systems in the repetitive procedure. The last system to solve is $D_p\mathbf{y}=\mathbf{c}_p$. The nested product solver easily extends to the matrix subblocks in the UBV decomposition from Chapter 4. Linear time systems with a hierarchical representation can now be stably solved. This opens up the opportunity of stably and efficiently deblurring an image using the nested product decomposition and solver.

CHAPTER 6

IMAGE RESTORATION APPLICATION

Image reconstruction can benefit from the development of the new hierarchical row compression and nested product conversion algorithms in Chapters 3 and 4 respectively, as well as, access to the fast stable solver. The application of the row compression and nested product decomposition is used in conjunction with deconvolution methods for image deblurring. A concise mathematical model represents the blurring process of the image.

6.1 Image Restoration Overview

The restoration process of an image begins with the input image $f(x, y)$ in the spatial domain. When a degradation function H and additive noise η are applied to the input image the result is a degraded image $g(x, y)$. If H is linear and invariant, then the degraded image is given by

$$g(x, y) = h(x, y) \bigstar f(x, y) + \eta(x, y) \tag{6.1}$$

where h is the spatial representation of the degradation function and $\bigstar$ indicates convolution. A diagram of the degradation process is shown in Figure 6.1. Convolution in the spatial domain of image processing is analogous to multiplication in the frequency domain. The frequency domain equivalent to Equation 6.1 is

$$G(u, v) = H(u, v) \cdot F(u, v) + N(u, v) \tag{6.2}$$

where G, H, N are the Fourier transforms of the corresponding terms in Equation 6.1 [64].

When there is insufficient knowledge about the degradation of an image, then the degradation is modeled and estimated. Some degradations can be represented by

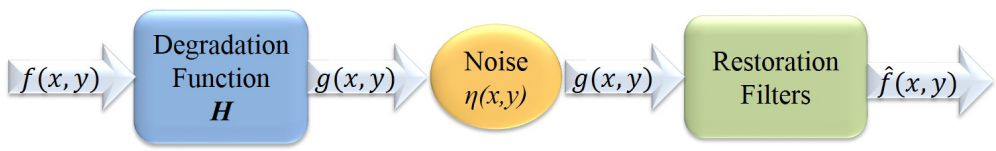


Figure 6.1. Illustration of the input image $f(x, y)$ that has the degradation function H and additive noise function η applied to f producing the degraded image $g(x, y)$ [64].

simple functions like relative constant speed movement, wrong lens focus and atmospheric turbulence [65]. The research in this dissertation looks at the atmospheric turbulence degradation mathematical model derived in [77]

$$H(u, v) = e^{-k(u^2+v^2)} \tag{6.3}$$

where k is a constant dependent on the nature of the turbulence. The atmospheric turbulence model is referred to as the *blur* model or operator in this research. There are various approaches to deblur or restore the image degraded by atmospheric turbulence such as inverse filtering.

6.2 Deblurring Methods

The first and most obvious choice in image restoration is *direct* inverse filtering based on the transform of the image [64, 65]. In this simple method, an estimate of the transform of the original image is computed,

$$\hat{F}(u, v) = \frac{1}{H(u, v)} \cdot G(u, v) + \frac{1}{H(u, v)} \cdot N(u, v).$$

Unfortunately, if the degradation function is known, this does not mean that the noise, $N(u, v)$, is known. Therefore, computing the exact input image is not possible. Combine the unknown noise $N(u, v)$ with $H(u, v)$ containing zeros or small values, and $\frac{1}{H(u,v)} \cdot N(u, v)$

will dominate the entire image restoration. Together, both of these issues, produce a poor result of full inverse filtering. In summation, direct inverse methods explicitly invert the blur function, and are extremely costly in terms of time and memory, and are sensitive to noise, since the blur function is often severely ill-conditioned.

When reconstructing a satellite image, noise is bound to be present. Filtering a solution, when restoring the image, diminishes the effects of the noise, and decompositions exist to filter the restoration process. The blur model is linear, and therefore admits a matrix formulation

$$\mathbf{g} = A\mathbf{f} + \boldsymbol{\eta}.$$

The SVD of the degradation function H is a direct method that filters H in reconstruction, and enforces regularity conditions on the solution [67]. However, direct algorithms for general matrices, such as the degradation function, require $O(n^3)$ work which is an exorbitant cost. Techniques for addressing this difficulty include regularization, and iterative methods such as conjugate gradient (CG) and generalized minimal residual (GMRES). Iterative solvers are based on Krylov subspaces, and when combined with good preconditioners these methods are alternatives to the classical FFT-based algorithms [31, 32]. Approximations to the blur matrix are exploited, efficiently computed, and efficiently inverted. This concept of a preconditioner increases the rate of convergence in the iteration process.

For this research on deblurring, the atmospheric turbulence blur model used is represented as a 2-D separable Gaussian function,

$$A(r, c) = e^{-k(c^2+r^2)} = e^{-k(c^2)} \cdot e^{-k(r^2)} = A_c \cdot A_r$$

where r is the row and c is the column. The ill-posed system to solve for is

$$A_c X A_r^T = B$$

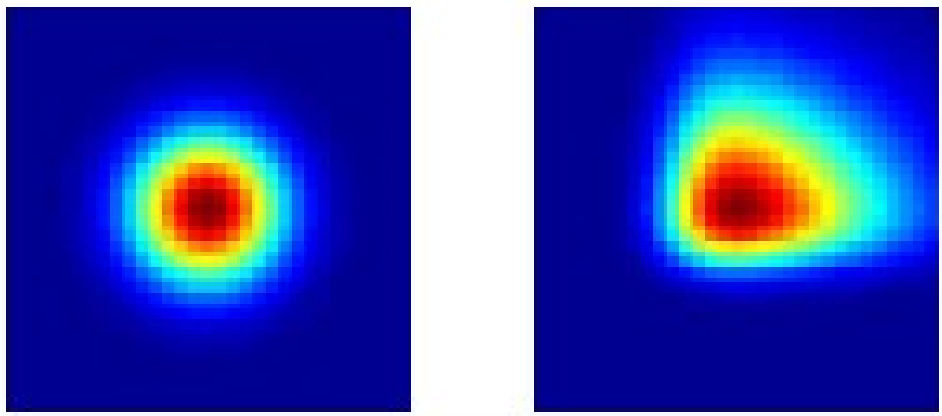


Figure 6.2. Symmetric blur model is on the left and nonsymmetric blur model is on the right [72].

where X is the restored image, B is the blurred noise-free image, and the two $n \times n$ Toeplitz matrices A_c and A_r represent blurring in the directions of columns and rows of the image respectively [67]. The equation to solve, in order to restore the image by direct inversion, is

$$X = A_c^{-1} B \left(A_r^T\right)^{-1}.$$

A zero boundary condition is assumed, as is a variant blur model. An illustration of the nonsymmetric, skew-normal blur function is seen in Figure 6.2 [72]. These zero boundary and spatially variant conditions allow the use of efficient methods such as the FFT and spectral decomposition [71].

The iterative restoration method our research initially explored uses the Kronecker product to form the PSF matrix, and GMRES to approximate the inverse to the preconditioner. The Kronecker GMRES iterative method is applied to the normal equations without explicitly forming the complex conjugate transpose of a matrix. The convergence of any inverse approximation iterative algorithm depends on the eigenvalues of the coefficient matrix A [66]. If the spectrum is clustered around one, then convergence will be rapid. So in

order for the Kronecker GMRES iterative method to be useful, preconditioning is typically applied to cluster the spectrum around one. Preconditioners and their inverses can be computed directly, but to do so is unnecessary. Often an approximate inverse preconditioner can be found by constructing a matrix composed of vectors which allows the algorithm to simply perform vector-matrix multiplication [69].

The preconditioners used for this particular implementation are based on the circulant Toeplitz system. The Toeplitz column and row blur operators are first transformed into Cauchy-like (CL), quasiseparable matrices [66]. The quasiseparable matrices are the basis for a preconditioned iterative method. The CL matrices have the property that each of the off-diagonal blocks have low rank. A fast divide and conquer algorithm extracts the rank structure, and compresses the approximations [78]. Generators are formed from the ranks and approximations, and preconditioners are constructed from the generators. A detailed presentation on the transformation of the Toeplitz blur operators to CL matrices is done in Section 6.3.

The new structured matrices are quasiseparable, and allow for an $O(n)$ solution to the approximate system for deblurring. The quasiseparable systems are ported into the superfast solver from [51]. The construction and solver both demonstrate stability. The cost of quasiseparable construction and the solver are both $O(np^2)$, where p is the maximum rank of the off-diagonal blocks in the CL matrices. Total cost is bounded by $O(n\log(n)) + O(np^2)$. The results produced artifacts in the image, and the lower the rank the worse the restoration appeared to be [66]. If direct methods with reasonable costs and stability can be developed, then deblurring of these systems would greatly benefit.

6.3 Deblurring Using Nested Product Algorithms

A direct method for the restoration of images degraded by atmospheric turbulence is examined. This work uses the class of approximations to blurring operators representing Gaussian blur as described in Section 6.2. When a satellite image is degraded by atmospheric

turbulence, regularized inverse methods to restore the image use a PSF to model the blur. In this dissertation, we consider $n \times n$ images, and PSFs that are spatially invariant and separable with a Toeplitz matrix to represent the PSF [72]. Iterative methods for solving Toeplitz systems may be fast, if they converge quickly, but they do not always converge quickly [67]. The system here is solved directly using a hierarchical representation of the matrix, and the row compression algorithm presented in Chapter 3 coupled with the new proven stable nested product decomposition and solver presented in [3]. The row compression algorithm refers to the compression of a linear system and is not to be confused with image compression.

The fast Fourier transform (FFT) is used to transform the Toeplitz blur matrix, T, into a Cauchy-like matrix with rank structure. We detail the Toeplitz matrices and their transformation here as was mentioned previously in Section 6.2. Given an $n \times n$ Toeplitz matrix, T, with constant diagonals of the form,

$$T = \begin{bmatrix} t_0 & t_{-1} & t_{-2} & \dots & t_{-(n-1)} \\ t_1 & t_0 & t_{-1} & \ddots & \vdots \\ t_2 & t_1 & \ddots & \ddots & t_{-2} \\ \vdots & \ddots & \ddots & t_0 & t_{-1} \\ t_{n-1} & \dots & t_2 & t_1 & t_0 \end{bmatrix}. \tag{6.4}$$

A common special case of Toeplitz matrices is when every row of the matrix is a right cyclic shift of the row above forming the circulant matrix, T_{cir}, [31]

$$T_{cir} = \begin{bmatrix} t_0 & t_{-1} & t_{-2} & \dots & t_{-(n-1)} \\ t_{-(n-1)} & t_0 & t_{-1} & \ddots & \vdots \\ t_{-(n-2)} & t_{-(n-1)} & \ddots & \ddots & t_{-2} \\ \vdots & \ddots & \ddots & t_0 & t_{-1} \\ t_{-1} & \dots & t_{-(n-2)} & t_{-(n-1)} & t_0 \end{bmatrix}. \tag{6.5}$$

The circulant matrix corresponds to periodic convolution which can be done efficiently $O(n\log(n))$ using the FFT algorithm.

For a general Toeplitz matrix, fast algorithms depend upon the fact that every Toeplitz matrix satisfies a displacement equation where two downshift matrices Z_1 and Z_{-1} are combined with the Toeplitz blur matrices to create a displacement matrix,

$$Z_1T - TZ_{-1} = \begin{bmatrix} t_{-(n-1)} - t_{-1} & t_{-(n-2)} - t_{-2} & \dots & t_1 - t_{-(n-1)} & 2t_0 \\ 0 & \dots & \dots & 0 & t_{-(n-1)} + t_1 \\ \vdots & & & \vdots & t_{-(n-2)} + t_2 \\ \vdots & & & \vdots & \vdots \\ 0 & \dots & \dots & 0 & t_{-1} + t_{-(n-1)} \end{bmatrix}. \tag{6.6}$$

The Toeplitz displacement equation also has the factorization

$$Z_1T - TZ_{-1} = XY^{\mathrm{T}} \tag{6.7}$$

where $X \in \mathbb{R}^{n\times\alpha}$, $Y^{\mathrm{T}} \in \mathbb{R}^{\alpha\times n}$, and $\alpha \leq n$ is the displacement rank$(XY^{\mathrm{T}}) = \alpha$ [51]. The factors X and Y are

$$X = \begin{bmatrix} 1 & t_0 \\ 0 & t_{-(n-1)} + t_1 \\ \vdots & t_{-(n-2)} + t_2 \\ \vdots & \vdots \\ 0 & t_{-1} + t_{-(n-1)} \end{bmatrix} \text{ and } Y = \begin{bmatrix} t_{-(n-1)} - t_1 & t_{-(n-2)} - t_2 & \dots & t_1 - t_{-(n-1)} & t_0 \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix}$$

respectively, and the displacement structure has rank 2.

The FFT is applied to both sides of Equation 6.3 resulting in

$$FZ_1F^HFT - FTZ_{-1} = FXY^{\mathrm{T}}.$$

The eigenvalues of the FFT of the displacement equation $\omega = e^{\frac{\pi i}{n}}$ are the base for the CL matrix, $\hat{C}$.

The CL matrix $\hat{C}$ is defined as

$$\hat{C} = FTD_0^{-1}F^H, \qquad c_{kj} = \frac{x_k^H y_j}{\omega^{2k-2} - \omega^{2j-1}} \tag{6.8}$$

where F is the normalized inverse of the discrete FFT matrix, D_0 is the diagonal matrix of ω raised to powers from 0 to $n-1$, and T is the Toeplitz blur matrix. Matrix $\hat{C}$ is uniquely determined by the generators from X and Y of the SVD. A second displacement equation is formed,

$$D_1\hat{C} - \hat{C}D_{-1} = (FX)\left(YD_0^{-1}F^H\right) \tag{6.9}$$

where D_1 and D_{-1} are diagonal matrices of ω raised to even and odd powers from 0 to $n-1$ [51]. Matrix $\hat{C}$ has the property that off-diagonal blocks can be approximated by matrices of low rank, and is classified as a quasiseparable matrix. All stages of the Toeplitz to Cauchy transformation algorithm are stable. Since matrix $\hat{C}$ is quasiseparable, it is a suitable target for the algorithms of this dissertation.

The hierarchical representation of a rank structured matrix partitions the CL matrix and represents the elements of the matrix using an expansion series in [51]. The resulting matrix is shown in Figure 3.2. The row compression algorithm from Chapter 3 begins with the hierarchical representation of the CL matrix, $\hat{C}$, and compresses the rows of the off-diagonal blocks by applying Householder transformations. The nested product decomposition from Chapter 4 operates on the compressed CL matrix and computes a sparse orthogonal nested product decomposition, UBV.

The nested product solver algorithm in [3] operates on the equation $\mathbf{y} = \hat{C}\mathbf{x}$. The solver begins with the outermost layer of the nested product, solves for that layer, strips the current layer away revealing the next layer, and repeats the process. The decomposition can be used to solve a system stably and with linear complexity. The separable model

with direct inversion is equivalent to $T_c X T_r^T = \tilde{B}$, where T_c and T_r are Toeplitz matrices. The hierarchical parameterization, row compression, nested product conversion, and linear system solver algorithms solve two regularized systems,

$$(T_c + \alpha I)X_0 = Y \text{ for } X_0, \qquad \text{and} \qquad X(T_r + \alpha I) = X_0 \text{ for } X$$

to obtain the deblurred image. This particular regularization only works naturally for positive definite matrices, which includes the matrix used to represent Gaussian blur. The method generalizes to Tikhonov regularization and more general blur functions. The direct deblurring methods presented here are proven stable.

CHAPTER 7

IMAGE COMPRESSION APPLICATION

Image compression minimizes volume of data in an image without loss of image information. Algorithms for image compression remove redundancies that appear in the data [65]. It is possible to further reduce redundancies in multiresolution wavelet compressions with the hierarchical row compression algorithm in this dissertation.

7.1 Image Compression Fundamentals

Data compression is the process of reducing the amount of data to represent an image. The goal is to remove redundant data and still retain quality information to represent the image. There three types of data redundancies: coding redundancy, spatial and temporal redundancy, and irrelevant information. The first redundancy deals with the binary coded bits that represent the intensities in the image. The second redundancy examines the array spatially for replicated correlated pixels. The third and simplest compression is removal of extraneous information that is not used. Irrelevant information compression is the direction of the research [64].

Human visual perception of the image information does not involve a quantitative analysis of the pixels. The pixel values can be modified, within given parameters, without any subjective degradation to the resulting image. For removal of irrelevant information, the digital image has its intensity values examined. Where there are clusters in the intensity values, then an averaging is done to have a single value represent that intensity. There is irreversible loss of information, but the loss is not perceived in the quality of the reconstructed image. This loss of quantitative information is referred to as quantization. There are three techniques for removing an image's irrelevant information: block transform coding predictive coding, and wavelet coding.

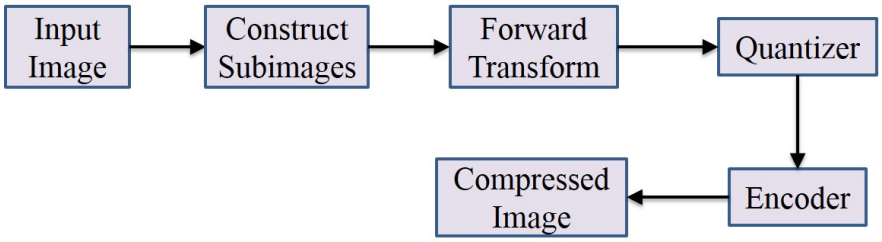


Figure 7.1. Diagram of a block transform encoding process.

In block transform coding, an image is divided into equal size subblocks that do not overlap. Each subblock is processed independently using a 2-D transform, such as the FFT. Then the transform coefficients are quantized. A diagram for the block transform coding process is found in Figure 7.1 [64]. In predictive coding, the pixels of the image are inspected and eliminating any redundancies in closely spaced pixels. The value of a closely spaced pixel is predicted, and the difference between the prediction and the actual value is stored. The differences are quantized and encoded. A diagram for the predictive coding process is found in Figure 7.2 [64].

The wavelet transform maps the spatial domain of an image to a frequency domain, then excessive redundancies in the image are exploited and removed [73, 74]. Wavelet transformations make it easier to compress, transmit and analyze images. The transform coding process is done in four major steps: apply the wavelet transform, detect the

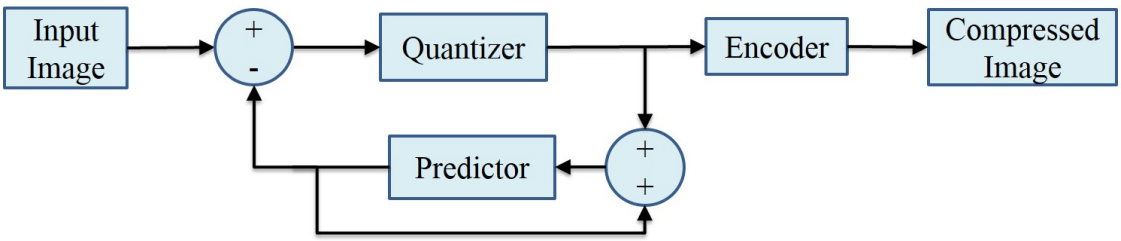


Figure 7.2. Diagram of a predictive encoding process.

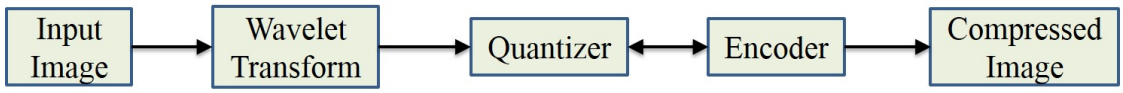


Figure 7.3. Diagram of the typical wavelet transform encoding and decoding process.

threshold, entropy code the quantized transform coefficients, and apply an inverse transform as shown in Figure 7.3 [75]. The difference between the wavelet transform coding and block transform coding methods is that the wavelet transform process does not include the subblock preprocessing stage in block transform coding.

7.2 Wavelets and Row Compression

Mathematical wavelet transforms are used extensively in image compression. Reducing the huge volume of data in a direct image spatial domain is important for transmission or storage. This method of lossy compression of the image is acceptable since the reconstruction of the image need not be exact [74]. The research here focuses on the discrete wavelet transform (DWT) which computes the series expansion coefficients for a function that is comprised of a wavelet function, $\psi(x)$, and a scaling function, $\varphi(x)$ [65]. In the one-dimensional case, the DWT of an image computes the approximation (low frequency) coefficients and the detail (high frequency) coefficients.

A single-level, separable, 2-D orthogonal Daubechies wavelet decomposition of an image forms a 2×2 block partitioned matrix W where each of the four subblocks contain detail or approximation coefficients. In the 2-D DWT, four separable functions are required:

$$\begin{aligned} \psi(x,y) &= \psi(x)\psi(y), && \text{scaling;} \\ \varphi^H(x,y) &= \varphi(x)\psi(y), && \text{horizontal edges;} \\ \varphi^V(x,y) &= \psi(x)\varphi(y), && \text{vertical edges;} \\ \varphi^D(x,y) &= \varphi(x)\varphi(y), && \text{diagonal edges.} \end{aligned} \tag{7.1}$$

Figure 7.4 depicts a three-stage 2-D DWT decomposition of an image with $A, H, V,$ and D as the low, horizontal, vertical and diagonal bands respectively [64]. Once an image is decomposed by the multiresolution wavelet transform, a hierarchical structure is evident in the resulting decomposed matrix W, where the lower left off-diagonal blocks represent the vertical bands and the upper right off-diagonal blocks represent the horizontal bands of the image. This hierarchical structure can be seen in Figure 7.4. This research exploits common rank structure between the lower off-diagonal blocks to introduce additional zeros. Similarly, this exploitation also applies to the common horizontal band structure in the upper off-diagonal blocks of the wavelet.

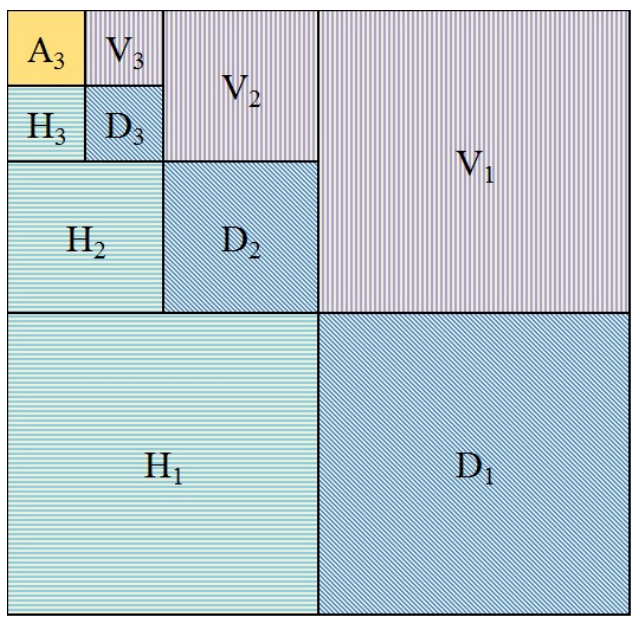


Figure 7.4. Wavelet mulitresolution image decomposition where L_i is a low band, V_i is a vertical band, H_i is a horizontal band, and D_i is a diagonal band generated from a three-stage wavelet transformation.

The vertical and horizontal edge off-diagonal blocks in the wavelet decomposition have similarities to the hierarchical matrices. The connection between the wavelet structure and the hierarchical representation is apparent. The row compression algorithm, developed in Chapter 3, applies a sequence of unitary transformations to the the low rank factorization to

introduce zeros into the off-diagonal blocks of the matrix. The row compression algorithm can be amended to operate on the Daubechies DWT common edge blocks, exploiting both the sparsity and rank structure of the wavelet transform. A permutation and subtraction of the vertical and horizontal edges aligns the common edge space for compression. Orthogonal transformations are applied to the common edge space and zeros are introduced. Thus the information about the edges captured by the wavelet transform is compressed.

CHAPTER 8

CONCLUSION AND COMPARISONS

The conclusion to this dissertation contains four parts. The comparisons of this research and algorithms to existing algorithms for hierarchical representations and other representations for rank structured matrices. Using double precision, IEEE machine epsilon as a benchmark, the relative backward error results on a numerically sensitive parameterization, for the algorithms in this research, are computed. Additionally, a specific structured matrix, that is unstable in other algorithms, with poor backward error, is used in our algorithms and shows good numerical stability. Next a list of the contributions made by this research is presented. Lastly, future work is discussed.

8.1 Complexity Comparisons

The algorithms investigated during this research operated on different types and/or representations of rank structured matrices. In order to present an "apples to apples" comparison, knowing the representation and type of rank structured matrix that each algorithm began with, is necessary for comparison. Since this research is on the hierarchical representation of a quasiseparable matrix, the first comparative table examines the different complexity aspects of algorithms that operate on the hierarchical representation of a structured matrix. There are three different types of matrices involved: $\mathscr{H}-$matrices, semiseparable matrices, and quasiseparable matrices. Table 8.1 summarizes the complexities of the algorithms that begin with a hierarchical representation of structured matrix. The $\mathscr{H}-$matrix is already in a hierarchical form which imposes no additional cost. However, algorithms for solving a system that involved an $\mathscr{H}-$matrix preconditioner either computed approximate inverses [8, 79] or performed an $\mathscr{H}-$LU factorization [80, 81]. Both approaches require $O(n\log(n))$ operations. The semiseparable [17–20] and quasiseparable [3] matrices

compute the hierarchical representation, which result in a cost of $O(n\log(n))$ operations; however, sparse systems and other possible structures can lower parameterization costs. The cost of the system solvers involving $\mathscr{H}$and $\mathscr{H}^2$ matrices, semiseparable, and quasiseparable matrices is linear which is a marked improvement over the $\mathscr{H}-$matrix system solvers. Storage involving semiseparable and quasiseparable matrices begins at $O(n\log(n))$, which is comparable to that of the $\mathscr{H}-$matrices. However, as the solver progresses only $O(n)$ storage is necessary. Thus storage and solver costs for semiseparable and quasiseparable matrices are comparable or lower than those for $\mathscr{H}-$matrices. Of these system solvers, the one presented in [3] is the only proven stable solver.

Table 8.1. Hierarchical Representation and Associated Complexity Costs (Initial parameterization storage is $O(n\log(n))$ for all algorithms.)

Matrix Type & Solver Algorithm	Compute Parameterization	Final Parameter Storage	Solver Operations
$\mathscr{H}-$matrix, Inversion	NA	$O(n\log(n))$	$O(n\log(n))$
$\mathscr{H}-$matrix, $\mathscr{H}-$LU	NA	$O(n\log(n))$	$O(n\log(n))$
$\mathscr{H}^2-$matrix, $\mathscr{H}-$LU	$O(n\log(n))$	$O(n)$	$O(n)$
Quasiseparable, UBV	$O(n\log(n))$	$O(n)$	$O(n)$

We now compare algorithms that begin with a general $n \times n$ rank structured matrix. There are four different types of matrices involved: $\mathscr{H}-$matrices, semiseparable matrices, hierarchical semiseparable and quasiseparable matrices. In this comparison, all algorithms begin with the general matrix form and must compute the hierarchical parameterization. In Table 8.2, the cost to compute the parameterization is the same across all algorithms. Storage involving semiseparable matrices, hierarchical semiseparable and quasiseparable matrices begins at $O(n\log(n))$, which is comparable to that of the $\mathscr{H}-$matrices. However, as the solvers progress only $O(n)$ storage is necessary. The ULV and UBV algorithms outperform the $\mathscr{H}-$matrix algorithm. Of these listed, only the UBV algorithm is proven stable. Thus the UBV algorithm is the subject of our research. It is clear that the nested UBV algorithms

have at least comparable, and, in some cases, improved complexity over other algorithms in this area. In Section 8.2, the stability of the row compression and nested UBV decomposition is discussed.

Table 8.2. General $n \times n$ Matrix and Associated Complexity Costs
(Initial parameterization storage is $O(n^2)$ for all algorithms.)

Matrix Type & Solver Algorithm	Compute Parameterization	Final Parameter Storage	Solver Operations
$\mathscr{H}-$matrix, Inversion	$O(n^2)$	$O(n\log(n))$	$O(n\log(n))$
$\mathscr{H}^2-$matrix, $\mathscr{H}-$LU	$O(n^2)$	$O(n)$	$O(n)$
Hierarchical Semiseparable, ULV	$O(n^2)$	$O(n\log(n))$	$O(n)$
Sequentially Semiseparable, ULV	$O(n^2)$	$O(n)$	$O(n)$
Quasiseparable, UBV	$O(n^2)$	$O(n)$	$O(n)$

The cost to compute the row compression of a hierarchical representation of a quasiseparable matrix is $O(n\log(n))$ operations. Converting the hierarchical row compression to a nested product requires $O(n\log(n))$ operations. In contrast, the row compression and nested product decomposition of a general matrix, presented in [3], use $O(n^2)$ operations. Thus, the algorithms developed in this research have reduced the complexity from $O(n^2)$ to $O(n\log(n))$ for matrices with a hierarchical representation. The storage for a hierarchical representation of a quasiseparable matrix is $O(n\log(n))$, and the storage for the final nested product decomposition is $O(n)$.

8.2 Stability

The row compression and nested product decomposition algorithms developed in this research focus on stability with comparable or improved complexity to other methods that currently exist. The essential numerical linear algebraic computations in the row compression and nested product decomposition algorithms are Householder transformations and the QR factorization. The strength of Householder transformations is their unconditional numerical

stability. The Householder QR factorization is well known to be normwise backward stable. The algorithms for the row compression and conversion to the nested product decomposition in this dissertation were specifically designed to take advantage of the numerical stability of Householder transformations and the QR factorization.

The backward error expression used to compute the row compression algorithm in this dissertation is from [3] and was discussed in Section 1.2. Given an $n \times n$ quasiseparable matrix A of rank 2, and the following information $P, Q, Y, Z \in \mathbb{R}^{n\times 2}$, $d_k \in \mathbb{R}$, and $n = 1024$. Matrix A is defined to be

$$\text{triu}(A) = \text{triu}(YZ^{\text{T}}), \quad \text{tril}(A) = \text{tril}(PQ^{\text{T}}), \quad \text{diag}(A) = D = \text{diag}(d_1, \ldots, d_n),$$

where triu, tril, and diag denote the upper triangular part, lower triangular part and diagonal of matrix A respectively.

The computed relative backward error for the hierarchical row compression algorithm applied to A, in double precision, is

$$\frac{\|\tilde{Q}\tilde{C} - A\|_2}{\|A\|_2} \approx 5.4985 \times 10^{-14}.$$

The relative backward error for the nested product decomposition begins with the definition of matrix C presented in Section 5.1 with R_0 and V^{T} defined in Equation 5.1. Given an $n \times n$ row compressed matrix C where $n = 1024$. The computed relative backward error for the nested product decomposition algorithm applied to C, in double precision, is

$$\frac{\|\tilde{R}_0\tilde{V}^{\text{T}} - C\|_2}{\|C\|_2} \approx 1.1150 \times 10^{-14}.$$

A full backward error analysis for the nested product decomposition and solver has been proven in [3].

The generator representation can be used in many standard algorithms for structured matrices. Regrettably, some generator representations display instability when algorithms are applied to these generators. It is sufficient to illustrate the instability of these generators in a specific example where the unstructured matrix-vector multiplication is applied to a quasiseparable matrix L. This example is presented in [3]. Given a quasiseparable matrix L that has the generator representation from Equation 2.1.1. The strictly lower triangular part is represented by the generators

$$\mathbf{p}_k^{\mathrm{T}} = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad A_k = \frac{1}{4}\begin{bmatrix} 11 & 4 \\ -4 & 1 \end{bmatrix}, \qquad \text{and } \mathbf{q}_k = \begin{bmatrix} 1+2^{-45} \\ -2 \end{bmatrix}$$

which do not depend on k, and $L = L + L^{\mathrm{T}}$.

The matrix-vector multiplier, $L\mathbf{x} = \mathbf{y}$, for a quasiseparable matrix and vector, $\mathbf{x} = \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}^{\mathrm{T}}$, can be computed by

$$\mathbf{z}_k = A_{k-1}\mathbf{z}_{k-1} + \mathbf{q}_{k-1}x_{k-1}, \quad \text{and} \quad y_k = \mathbf{p}_k^{\mathrm{T}}\mathbf{z}_k$$

where $k = 2, 3, \dots, n$, $y_1 = 0$ and $\mathbf{z}_1 = 0$. For $n = 32$, the double precision computations of the matrix-vector multiplier $L\mathbf{x} = \mathbf{y}$ formed the approximation $\hat{\mathbf{y}} = \mathrm{fl}(L\mathbf{x})$. An additional approximation was computed using unstructured matrix-vector multiplication which formed $\tilde{\mathbf{y}} = \mathrm{fl}(L\mathbf{x})$. The computed backward error was

$$\|L\|_2 \approx 3.82, \quad \frac{\|\tilde{\mathbf{y}} - \hat{\mathbf{y}}\|_2}{\|L\|_2\|\mathbf{x}\|_2} \approx 8.22 \times 10^{-8}.$$

These results are not consistent with reasonable normwise backward error bounds on matrix L. Therefore, the existing algorithms working with such generators cannot be expected to be normwise backward stable [3]. The issue is intrinsic to the generators and is not an issue with the algorithm. The nested product approach averts this problem by avoiding a generator representation.

Nonetheless, when the row compression and nested product algorithms are applied to this same matrix L, the backward error is consistent with reasonable normwise backward error bounds. For $n = 32$, the double precision computations yielded

$$\frac{\|\tilde{Q}\tilde{C} - L\|_2}{\|L\|_2} \approx 1.0803 \times 10^{-14} \qquad \text{and} \qquad \frac{\|\tilde{R}_0\tilde{V}^{\mathrm{T}} - L\|_2}{\|L\|_2} \approx 1.6213 \times 10^{-15}.$$

This is a major improvement over the generator based algorithms from the example in [3]. The row compression and nested product backward errors with respect to L demonstrates the stability of both algorithms in the dissertation.

The approach for computing the hierarchical representation of a quasiseparable matrix is directly related to the works in [3, 17–19], and has been adapted for this research. The hierarchical parameterization algorithm is not considered part of the original research in this dissertation. The number of operations to compute the hierarchical parameterization depends upon whether updating techniques are used, if additional structure can be further exploited, and what type of matrix is involved. The extraction of parameters for the hierarchical representation of a quasiseparable matrix is stable. Other parameterizations, such as generator representation and rotation representation, can have stability issues for some matrices.

The total operation complexity for computation of the row compression, nested product decomposition, matrix-vector multiplication and nested product solver of a hierarchical representation of a quasiseparable matrix is

$$O(n\log(n)) + O(n\log(n)) + O(n) + O(n) = O(n\log(n)).$$

The row compression and nested product decomposition in this dissertation are stable, and the overall cost is an improvement over the work in [3] for matrices for which a hierarchical representation is readily available or easily computed.

8.3 Conclusion

Quasiseparable matrices, and hierarchical representations of them, are playing an ever increasing role in applied mathematics, engineering and computer science with the versatility of their use in many applications. Stable algorithms for the nested product representation of a quasiseparable matrix, matrix-vector multiplication and fast solver, have been developed in [3]. These algorithms are the only proven stable algorithms for this class of matrices. When they are coupled with existing gaps in algorithms for conversion of hierarchical representations to other parameterizations and potential applications in image deblurring and wavelet compression, it is clear there is room for major advancements in this area of numerical linear algebra and associated applications to image processing. The research in this dissertation on row compression and nested product decomposition is a discernible contribution towards research involving quasiseparable matrices.

This research began by looking at hierarchical matrices, the nested product solver for quasiseparable matrices in [3], stable algorithms, and applications in image processing. The following results have been achieved:

1. A new row compression algorithm for the hierarchical representation of quasiseparable matrices has been introduced that utilizes the stability of Householder transformations to compress the matrix. The row compression algorithm exploits the data sparse characteristic of the hierarchical representation, and computes the compression in $O(n\log(n))$ operations, which is an improvement over the row compression in [3]. The main goal in this research was stability, and this has been accomplished. For a 1024×1024 quasiseparable matrix, the relative backward error is $\approx 5.4985 \times 10^{-14}$ for the row compression algorithm.

2. The row compression algorithm is recursive and operates on partitions of the matrix. A novel approach, within the algorithm, is the repartitioning of the compressed blocks

prior to the next compression. The repartitioning procedure in the row compression is a very difficult procedure to implement, and is not being done in any related papers.

3. The nested UBV decomposition algorithm is directly connected to the work by Bella, Olshevsky, and Stewart. The conversion to a nested product representation required adapting the nested UBV decomposition to work with blocks. The nested UBV decomposition of a row compressed hierarchical representation of a quasiseparable matrix has $O(n\log(n))$ operations which is an improvement over the nested product decomposition in [3]. For a 1024×1024 quasiseparable matrix, the relative backward error is $\approx 1.1150 \times 10^{-14}$ for the nested product decomposition. The nested UBV decomposition algorithm is stable and a full backward error analysis of the algorithm is done in [3].

4. The primary reason for this research was to allow the stable procedures from [3] to be applied to hierarchical representations of a quasiseparable matrix. Now a systems with a hierarchical representation of a quasiseparable matrix, with a strong guarantee of stability, can be solved in $O(n\log(n))$ total operations.

5. A stable linear time algorithm for the solution of a rank structured system is applied to a point spread function (PSF) represented by a Toeplitz matrix. The algorithm is used to restore an image degraded by a spatially invariant and separable blur. The PSF is transformed into a rank structured Cauchy-like matrix. The resulting matrix has a hierarchical representation that is compressed into a decomposition that yields a linear time system solver.

6. This research exploits a stable row compression algorithm for decomposing a hierarchically or sequentially structured matrix to compress an $n \times n$ image represented by a wavelet transform. The multiresolution discrete wavelet transform is used to decompose an image. The row compression algorithm builds up a low rank approximation of the

wavelet transform by applying orthogonal transformations and updating techniques. The cost is $O(n^2)$ operations.

8.4 Future Work

The row compression and nested product conversion algorithms presented in this dissertation have been designed and implemented in MATLAB. The focus of this research was designing stable algorithms with lower complexity. However, in order to optimize the algorithms, in practice they must be translated into an efficiently compiled language to increase efficiency and speed, and manage memory. Thus one aspect of future work is the implementation of the algorithms in Fortran. The complex data structures in the algorithms were designed and created in MATLAB, and require a great deal of finesse when implementing them in Fortran. The Fortran translation also requires incorporating LAPACK numerical linear algebra routines for systems of equations. The Fortran implementation has commenced. The performance of the application of the algorithms in image deblurring and wavelet compression will be examined upon completion of the Fortran conversion. The hierarchical representation conversion opens the door to extend our work to include other rank structured matrices with a hierarchical structure and the problems they model. Some such matrices are boundary, edge, or dyadic clustering concentrations in the partitioning of hierarchical matrices in [10], and they would benefit from access to the proven stable solver in [3].

Parallelization of the hierarchical parameterization is an area to explore. As the algorithms were being developed in this research, there was discussion on how to do part of the computations in parallel. The nested product conversion algorithm has matrix blocks that are interdependent on each other within the computations of the decomposition, and thus would not benefit from parallelization at this point in time. However, the hierarchical partitioning of the quasiseparable matrix performs computations on matrix blocks that are independent of each other, and we would like to do parallel partitions of the hierarchical

parameterization in our future research. Additionally when designing and implementing the row compression algorithm, an idea emerged about parallelizing the row compression into two sets of compressions, and then merging the results together. Future work would investigate parallel row compressions on the hierarchical representation of a quasiseparable matrix.

REFERENCES

[1] D. A. Bini, V. Mehrmann, V. Olshevsky, E. E. Tyrtyshnikov, and M. van Barel, Eds., *Numerical methods for structured matrices and applications*, ser. Operator Theory: Advances and Applications. Basel, CH: Birkhäuser Verlag, 2010, vol. 199, the Georg Heinig memorial volume.

[2] V. Olshevsky, Ed., *Structured matrices in mathematics, computer science, and engineering. I*, ser. Contemporary Mathematics, vol. 280. Providence, RI: American Mathematical Society, 2001.

[3] T. Bella, V. Olshevsky, and M. Stewart, "A nested product decompostion of a Quasiseparable matrix," *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 4, pp. 1520–1555, 2013.

[4] P. Dewilde and A.-J. van der Veen, *Time-varying systems and computations.* Boston, MA: Kluwer Academic Publishers, 1998.

[5] ——, "Inner-outer factorization and the inversion of locally finite systems of equations," *Linear Algebra Appl.*, vol. 313, no. 1-3, pp. 53–100, 2000.

[6] Y. Eidelman and I. Gohberg, "A modification of the Dewilde-van der Veen method for inversion of finite structured matrices," *Linear Algebra Appl.*, vol. 343/344, pp. 419–450, 2002, Special issue on structured and infinite systems of linear equations.

[7] E. Alijagic and P. Dewilde, "Minimal quasi-separable realizations for the inverse of a quasi-separable operator," *Linear Algebra Appl.*, vol. 414, no. 2-3, pp. 445–463, 2006.

[8] W. Hackbusch, "A sparse matrix arithmetic based on $\mathscr{H}$-matrices. I. Introduction to $\mathscr{H}$-matrices," *Computing*, vol. 62, no. 2, pp. 89–108, 1999.

[9] W. Hackbusch, L. Grasedyck, and S. Börm, "An introduction to hierarchical matrices," in *Proceedings of EQUADIFF, 10 (Prague, 2001)*, vol. 127, no. 2, 2002, pp. 229–241.

[10] L. Grasedyck and W. Hackbusch, "Construction and arithmetics of $\mathscr{H}$-matrices," *Computing*, vol. 70, no. 4, pp. 295–334, 2003.

[11] S. Börm and W. Hackbusch, "A short overview of $\mathscr{H}^2$-matrices," *PAMM*, vol. 2, no. 1, pp. 33–36, 2003. [Online]. Available: http://dx.doi.org/10.1002/pamm.200310009

[12] S. Börm, "$\mathscr{H}^2$-matrices multilevel methods for the approximation of integral operators," *Computing and Visualization in Science*, vol. 7, no. 3-4, pp. 173–181, 2004. [Online]. Available: http://dx.doi.org/10.1007/s00791-004-0135-2

[13] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, and A. J. v. d. Veen, "Fast stable solver for sequentially semi-separable linear systems of equations," in *Proceedings of the 9th International Conference on High Performance Computing*, ser. HiPC '02. London, UK: Springer-Verlag, 2002, pp. 545–554.

[14] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A.-J. van der Veen, and D. White, "Some fast algorithms for sequentially semiseparable representations," *SIAM J. Matrix Anal. Appl.*, vol. 27, no. 2, pp. 341–364, 2005.

[15] R. Vandebril, M. Van Barel, and N. Mastronardi, "A note on the representation and definition of semiseparable matrices," *Numer. Linear Algebra Appl.*, vol. 12, no. 8, pp. 839–858, 2005. [Online]. Available: http://dx.doi.org/10.1002/nla.455

[16] ——, *Matrix computations and semiseparable matrices. Vol. I.* Baltimore, MD: Johns Hopkins University Press, 2008.

[17] S. Chandrasekaran, M. Gu, and W. Lyons, "A fast adaptive solver for hierarchically semiseparable representations," *Calcolo*, vol. 42, no. 3-4, pp. 171–185, 2005.

[18] S. Chandrasekaran, M. Gu, and T. Pals, "A fast ULV decomposition solver for hierarchically semiseparable representations," *SIAM J. Matrix Anal. Appl.*, vol. 28, no. 3, pp. 603–622, 2006.

[19] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals, "A fast solver for HSS representations via sparse matrices," *SIAM J. Matrix Anal. Appl.*, vol. 29, no. 1, pp. 67–81, 2006/07.

[20] Z. Sheng, P. Dewilde, and S. Chandrasekaran, "Algorithms to solve hierarchically semi-separable systems," in *System theory, the Schur algorithm and multidimensional analysis*, ser. Oper. Theory Adv. Appl. Basel, CH: Birkhauser, 2007, vol. 176, pp. 255–294.

[21] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, "Fast algorithms for hierarchically semiseparable matrices," *Numer. Linear Algebra Appl.*, vol. 17, no. 6, pp. 953–976, 2010.

[22] W. Hackbusch and B. Khoromskij, "A sparse h-matrix arithmetic: general complexity estimates," *Journal of Computational and Applied Mathematics*, vol. 125, no. 12, pp. 479 – 501, 2000, numerical Analysis 2000. Vol. VI: Ordinary Differential Equations and Integral Equations. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377042700004866

[23] W. Hackbusch and B. N. Khoromskij, "A sparse $\mathscr{H}$-matrix arithmetic. II. Application to multi-dimensional problems," *Computing*, vol. 64, no. 1, pp. 21–47, 2000.

[24] S. Le Borne, "Multilevel hierarchical matrices," *SIAM J. Matrix Anal. Appl.*, vol. 28, no. 3, pp. 871–889 (electronic), 2006.

[25] S. Delvaux, "Rank structured matrices," Dissertation, University of Leuven, 2007.

[26] S. Dianat and E. Saber, *Advanced Linear Algebra for Engineers With Matlab.* Taylor & Francis, 2009.

[27] T. Kailath and A. H. Sayed, Eds., *Fast reliable algorithms for matrices with structure.* Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 1999.

[28] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. Comput. Phys.*, vol. 73, no. 2, pp. 325–348, 1987. [Online]. Available: http://dx.doi.org/10.1016/0021-9991(87)90140-9

[29] R. Beatson and L. Greengard, "A short course on fast multipole methods," in *Wavelets, multilevel methods and elliptic PDEs (Leicester, 1996)*, ser. Numer. Math. Sci. Comput. New York, NY: Oxford Univ. Press, 1997, pp. 1–37.

[30] D. A. Bini, L. Gemignani, and J. R. Winkler, "Structured matrix methods for CAGD: an application to computing the resultant of polynomials in the Bernstein basis," *Numer. Linear Algebra Appl.*, vol. 12, no. 8, pp. 685–698, 2005.

[31] J. W. Demmel, *Applied numerical linear algebra.* Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 1997.

[32] L. N. Trefethen and D. Bau, III, *Numerical linear algebra.* Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 1997.

[33] R. A. Liebler, *Basic matrix algebra with algorithms and applications*, ser. Chapman & Hall/CRC Mathematics. Chapman & Hall/CRC, Boca Raton, FL, 2003.

[34] G. Strang, *Linear algebra and its applications*, 4th ed. Belmont, CA: Thomson, Brooks/Cole, 2006.

[35] G. W. Stewart, *Matrix algorithms. Vol. I.* Philadelphia, PA: Society for Industrial and Applied Mathematics, 1998, Basic Decompositions.

[36] G. H. Golub and C. F. Van Loan, *Matrix computations*, 3rd ed., ser. Johns Hopkins Studies in the Mathematical Sciences. Baltimore, MD: Johns Hopkins University Press, 1996.

[37] N. J. Higham, *Accuracy and stability of numerical algorithms*, 2nd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2002.

[38] R. Vandebril, M. Van Barel, and N. Mastronardi, *Matrix computations and semiseparable matrices. Vol. II.* Baltimore, MD: Johns Hopkins University Press, 2008.

[39] D. A. Bini, Y. Eidelman, L. Gemignani, and I. Gohberg, "Fast QR eigenvalue algorithms for Hessenberg matrices which are rank-one perturbations of unitary matrices," *SIAM J. Matrix Anal. Appl.*, vol. 29, no. 2, pp. 566–585, 2007.

[40] S. Chandrasekaran and M. Gu, "Fast and stable eigendecomposition of symmetric banded plus semi-separable matrices," *Linear Algebra Appl.*, vol. 313, no. 1-3, pp. 107–114, 2000. [Online]. Available: http://dx.doi.org/10.1016/S0024-3795(00)00106-3

[41] D. Fasino, N. Mastronardi, and M. Van Barel, "Fast and stable algorithms for reducing diagonal plus semiseparable matrices to tridiagonal and bidiagonal form," in *Fast algorithms for structured matrices: theory and applications (South Hadley, MA, 2001)*, ser. Contemp. Math. Providence, RI: Amer. Math. Soc., 2003, vol. 323, pp. 105–118. [Online]. Available: http://dx.doi.org/10.1090/conm/323/05699

[42] T. Bella, "Topics in numerical linear algebra related to quasiseparable and other structured matrices," Dissertation, University of Conneticut, 2008.

[43] T. Bella, Y. Eidelman, I. Gohberg, and V. Olshevsky, "Computations with quasiseparable polynomials and matrices," *Theoret. Comput. Sci.*, vol. 409, no. 2, pp. 158–179, 2008.

[44] G. H. Golub and C. F. Van Loan, *Matrix computations*, 4th ed., ser. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 2013.

[45] J. Xia, S. Chandrasekaran, M. Gu, and X. Li, "Superfast multifrontal method for large structured linear systems of equations," *SIAM J. Matrix Anal. Appl.*, vol. 31, no. 3, pp. 1382–1411, 2009.

[46] J. Xia, "On the complexity of some hierarchical structured matrix algorithms," January 2012, submitted.

[47] P. Zhlobich, "Quasiseparable Matrices and Polynomials," Dissertation, University of Connecticut, 2010.

[48] J. Carrier, L. Greengard, and V. Rokhlin, "A fast adaptive multipole algorithm for particle simulations," *SIAM J. Sci. Statist. Comput.*, vol. 9, no. 4, pp. 669–686, 1988.

[49] Y. Eidelman and I. Gohberg, "Linear complexity inversion algorithms for a class of structured matrices," *Integral Equations Operator Theory*, vol. 35, no. 1, pp. 28–52, 1999.

[50] S. Delvaux and M. Van Barel, "A QR-based solver for rank structured matrices," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 2, pp. 464–490, 2008.

[51] S. Chandrasekaran, M. Gu, X. Sun, J. Xia, and J. Zhu, "A superfast algorithm for Toeplitz systems of linear equations," *SIAM J. Matrix Anal. Appl.*, vol. 29, no. 4, pp. 1247–1266, 2007.

[52] S. Delvaux and M. Van Barel, "A Givens-weight representation for rank structured matrices," *SIAM J. Matrix Anal. Appl.*, vol. 29, no. 4, pp. 1147–1170, 2007.

[53] Y. Eidelman and I. Gohberg, "On generators of quasiseparable finite block matrices," *Calcolo*, vol. 42, no. 3-4, pp. 187–214, 2005.

[54] T. F. Chan and J. Shen, *Image processing and analysis : variational, PDE, wavelet, and stochastic methods.* Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2005.

[55] M. Van Barel, R. Vandebril, N. Mastronardi, S. Delvaux, and Y. Vanberghen, "Rank structured matrix operations," in *Workshop on State-of-the-Art in Scientific and Parallel Computing, PARA06*, Umea, SE, June 2006.

[56] Y. Eidelman, I. Gohberg, and V. Olshevsky, "The QR iteration method for Hermitian quasiseparable matrices of an arbitrary order," *Linear Algebra Appl.*, vol. 404, pp. 305–324, 2005.

[57] D. A. Bini, L. Gemignani, and V. Y. Pan, "Fast and stable QR eigenvalue algorithms for generalized companion matrices and secular equations," *Numer. Math.*, vol. 100, no. 3, pp. 373–408, 2005. [Online]. Available: http://dx.doi.org/10.1007/s00211-005-0595-4

[58] W. Hackbusch and S. Börm, "$\mathscr{H}^2$-matrix approximation of integral operators by interpolation," *Appl. Numer. Math.*, vol. 43, no. 1-2, pp. 129–143, 2002.

[59] T. Bella, V. Olshevsky, and P. Zhlobich, "Classifications of recurrence relations via subclasses of (h, m)-quasiseparable matrices," in *Numerical Linear Algebra in Signals, Systems and Control*, ser. Lecture Notes in Electrical Engineering, P. Van Dooren, S. P. Bhattacharyya, R. H. Chan, V. Olshevsky, and A. Routray, Eds. Springer Netherlands, 2011, vol. 80, pp. 23–53.

[60] S. Chandrasekaran and M. Gu, "Fast and stable algorithms for banded plus semiseparable systems of linear equations," *SIAM J. Matrix Anal. Appl.*, vol. 25, no. 2, pp. 373–384, 2003. [Online]. Available: http://dx.doi.org/10.1137/S0895479899353373

[61] P. Benner and T. Mach, "On the qr decomposition of $\mathscr{H}-$matrices," *Computing*, vol. 88, no. 3-4, pp. 111–129, 2010.

[62] S. Delvaux, K. Frederix, and M. Van Barel, "Transforming a hierarchical into a unitary-weight representation," *Electron. Trans. Numer. Anal.*, vol. 33, pp. 163–188, 2008/09.

[63] S. Börm and J. Gördes, "An exact solver for simple $\mathscr{H}$-matrix systems."

[64] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2008.

[65] M. Sonka, V. Hlavác, and R. Boyle, *Image Processing, Analysis, and Machine Vision*, 3rd ed. Toronto, Canada: Thomson-Engineering, 2008.

[66] M. Hudachek-Buswell, C. Matos, and M. Stewart, "Deblurring with rank-structured inverse approximations," in *SIGGRAPH '09: Posters*, ser. SIGGRAPH '09. New York, NY: ACM, 2009, pp. 34:1–34:1.

[67] P. C. Hansen, J. G. Nagy, and D. P. OLeary, *Deblurring images : matrices, spectra, and filtering*. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2006.

[68] M. K. Ng and B. Plemmons, "Blind deconvolution and structured matrix computations with applications to array imaging, blind deconvolution: Theory and applications," 2007.

[69] J. Nagy, R. Plemmons, and T. Torgersen, "Iterative image restoration using approximate inverse preconditioning," *Image Processing, IEEE Transactions on*, vol. 5, pp. 1151–1162, 1996.

[70] M. Benzi and M. Ng, "Preconditioned iterative methods for weighted Toeplitz least squares problems," *SIAM J. Matrix Anal. Appl.*, vol. 27, no. 4, pp. 1106–1124 (electronic), 2006.

[71] J. G. Nagy, K. Palmer, and L. Perrone, "Iterative methods for image deblurring: a matlab object-oriented approach," *Numerical Algorithms*, vol. 36, pp. 73–93, 2003.

[72] P. C. Hansen and T. K. Jensen, "Noise propagation in regularizing iterations for image deblurring," *Electron. Trans. Numer. Anal.*, vol. 31, pp. 204–220, 2008.

[73] R. DeVore, B. Jawerth, and B. Lucier, "Image compression through wavelet transform coding," *Information Theory, IEEE Transactions on*, vol. 38, no. 2, pp. 719–746, March 1992.

[74] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *Image Processing, IEEE Transactions on*, vol. 1, no. 2, pp. 205–220, Apr 1992.

[75] X. Wu, "Compression of wavelet transform coefficients," in *The Transform and Data Compression Handbook*, ser. Electrical Engineering and Applied Signal Processing Series. Boca Raton, FL: CRC Press, 2001, pp. 347–378. [Online]. Available: http://dx.doi.org/10.1201/9781420037388.ch8

[76] G. Strang, "Groups of banded matrices with banded inverses," *Proc. Amer. Math. Soc.*, vol. 139, no. 12, pp. 4255–4264, 2011. [Online]. Available: http://dx.doi.org/10.1090/S0002-9939-2011-10959-6

[77] R. E. Hufnagel and N. R. Stanley, "Modulation transfer function associated with image transmission through turbulent media," *J. Opt. Soc. Am.*, vol. 54, no. 1, pp. 52–60, Jan 1964. [Online]. Available: http://www.opticsinfobase.org/abstract.cfm?URI=josa-54-1-52

[78] R. H.-F. Chan and X.-Q. Jin, *An introduction to iterative Toeplitz solvers*, ser. Fundamentals of Algorithms. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2007, vol. 5.

[79] G.-L. Börm, Steffen and W. Hackbusch, "Introduction to hierarchical matrices with applications," *Engineering Analysis with Boundary Elements*, vol. 27, no. 5, pp. 405–422, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0955799702001522

[80] L. Grasedyck, "Adaptive recompression of $-$matrices for bem," *Computing*, vol. 74, no. 3, pp. 205–223, 2005. [Online]. Available: http://dx.doi.org/10.1007/s00607-004-0103-1

[81] L. Grasedyck, R. Kriemann, and S. LeBorne, "Domain decomposition based $-$lu preconditioning," *Numerische Mathematik*, vol. 112, no. 4, pp. 565–600, 2009. [Online]. Available: http://dx.doi.org/10.1007/s00211-009-0218-6