DETERMINING THE COMPLETE SPECTRUM FOR SPARSE SYMMETRIC MATRICES

By

Ian A. Cavers

B.Sc. University of British Columbia

M.Sc. (Computer Science) University of British Columbia

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF COMPUTER SCIENCE

We accept this thesis as conforming

to the required standard

.....

.....

THE UNIVERSITY OF BRITISH COLUMBIA

September 1994

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science
The University of British Columbia
Vancouver, Canada

Date: Sept. 27, 1994

Abstract

This dissertation studies a restricted form of the fundamental algebraic eigenvalue problem. From the broad spectrum of eigenvalue problems and solution methods, it focuses upon sequential direct methods for determining moderately large subsets of eigenvalues or the complete spectrum of large sparse symmetric matrices. The thesis uses a combination of theoretical analysis and experimentation with symbolic and numeric implementations to develop generally applicable, reliable, efficient and accurate algorithms that are easily applied by novice and expert practitioners alike. This dissertation's approach is to reexamine eigenvalue methods based on the similarity reduction of matrices to tridiagonal form, developing algorithms that more fully exploit matrix sparsity.

Using specially developed sparse reduction tools, the thesis identifies the deficiencies and limitations of existing direct tridiagonalization methods, providing an improved understanding of the underlying fill characteristics of sparse reductions. The best previously published approach combines a bandwidth reducing preordering with Rutishauser and Schwarz's $O(bn^2)$ band-preserving tridiagonalization algorithm. This approach places complete reliance upon the preordering to exploit sparsity, but it typically leaves the band of the matrix relatively sparse prior to reduction. The thesis presents several novel sparse reduction algorithms, including the hybrid tridiagonalization methods HYBBC and HYBSBC, that rearrange the elimination of nonzero entries to improve band sparsity utilization.

HYBBC combines Bandwidth Contraction, a diagonally-oriented sparse reduction, with Rutishauser and Schwarz's column-oriented tridiagonalization. For a wide range of 70 practical sparse problems the new algorithm reduces CPU requirements by an average of 31%, with reductions as high as 63%. HYBSBC improves upon HYBBC's successful techniques by substituting the novel Split Bandwidth Contraction algorithm for Bandwidth Contraction. The Split Bandwidth Contraction algorithm takes additional advantage of band sparsity to significantly improve the efficiency of partial bandwidth contractions. In addition, HYBSBC employs the Δ-*transition strategy* to precisely regulate the transition between its two reduction stages, permitting tridiagonalization in as little as 1/5 the time of Rutishauser and Schwarz. Finally, to demonstrate the relative efficiency of sparse tridiagonalization based eigenvalue methods, the thesis compares variants of the Lanczos algorithm to HYBSBC using theoretical analysis and experimentation with leading Lanczos codes.

Contents

List of Tables

List of Figures

Acknowledgements

First and foremost, I would like to thank my supervisor, Jim Varah, for believing in and encouraging a young general sciences student so many years ago. Without your guidance, patience, friendship, and gentle prodding my graduate school years would not have been so memorable or as successful. Many thanks Jim.

I would also like to thank the other members of my supervisory committee: Uri Acher, Maria Klawe, and Manfred Trummer for their guidance and support. Thanks as well go to Robert Schreiber, David Kirkpatrick, Brian Wetton and Larry Roberts for their participation on the examination committee.

Many thanks to Roger Grimes and John Lewis from Boeing Computer Services for providing me access to the BCS shift and invert Lanczos code. I would also like to express my appreciation to John for his interest in my research and his many valuable suggestions.

The many friends I have made during graduate school have helped to turn the endless days of hard work into a wonderful experience. To Murr, Don, Terry, Barry, Rick, Felix, George(s), Mike(s), Peter(s), Andrew, Andy, Pierre, John, Swamy, Dave(s), Alex, the CS and CICSR office staff, Karen, Frank, Ming, Glen, Carlin, Moyra, Luping, and their spouses – and the many, many others whom I've enjoyed getting to know – thanks for your friendship and help. Long live "Bald Guy"!

Outside the university I have a large extended family whose unfailing faith, love, and support has meant everything to me during this endeavor. A special thanks to my Mum (Gerry) for proofreading every single word of this dissertation. Thanks also go to Aunt Hope, to my brother and sisters (Drum, Joan and Shelagh), their spouses (Pat, Al and Mike), my in-laws (George, Zora, Irene and Ken), and my wonderful gaggle of nieces and nephews (Lindsay, Graham, Andy, Lauren, Gordon, Robert and Matthew). And to the other members of my extended family: the Harrisons, the Cavers, the Smith Clan, the Shearer-Mealings (beat ya Ayden), the Sperlings, the Andrews, Jim, Jon, ..., many thanks.

Finally, I have saved the most important thank-you for last. I would like to thank my wife Diana for her endless support, patience, understanding and love. Start the e.b.o. Jenna your muč is coming home.

This dissertation is dedicated to the memory of
Dr. Stuart Donald Cavers,
professor, teacher, friend and Dad.

Chapter 1

Introduction

This thesis studies a restricted form of the fundamental algebraic eigenvalue problem. Given an $n \times n$ square matrix A values of the scalar λ, called *eigenvalues*, are sought for which the following set of homogeneous linear equations has a nontrivial solution.

$$Ax = \lambda x \tag{1.1}$$

The nontrivial solution, x, corresponding to a particular λ is referred to as its *eigenvector*.

Eigenvalue problems arise in many different areas of science and engineering. Applications range from more traditional problems found in structural analysis, quantum physics and chemistry, to problems in molecular dynamics, the analysis of electrical power systems, and the modeling of tides and currents in oceanographic research. Numerical methods for these eigenvalue problems are desired which are reliable, appropriately accurate, exhibit efficient execution and require low levels of storage. Despite the simple formulation of the basic eigenvalue problem, no single algorithm can provide an optimal balance of these goals for all applications. Instead, there is a wide range of different numerical eigenvalue methods. Each approach tries to exploit particular matrix properties, differing solution requirements or special computing environment characteristics.

Important matrix properties influencing algorithm design include symmetry, real or complex entries, the fraction of entries which are zero, and the presence of special sparsity structures. Among the many solution requirements affecting algorithm design and selection, answers to the following questions are fundamental.

- Are all eigenvalues required?
- Are a few of the largest or smallest eigenvalues needed?
- Is the identification of eigenvalue multiplicity required?
- Are all eigenvalues in a specified region required?
- Are eigenvectors needed?

Many characteristics of the computing environment also affect algorithm design and selection. Is the algorithm targeted for a sequential or parallel architecture? Do the processing units have scalar or vector capabilities? Finally, algorithm design is also affected by the alternatives of a shared or distributed memory architecture, and cache memory implementations.

From this broad spectrum of eigenvalue problems and solution methods, this thesis focuses upon sequential direct methods for determining moderately large subsets of eigenvalues or the complete spectrum, including eigenvalue multiplicities, of large sparse symmetric matrices. Other than symmetry, no special assumptions are made of a matrix's sparsity pattern. Although sequential eigenvalue calculations are the primary objective of this thesis, when appropriate we comment upon parallel implementation and eigenvector computations.

A precise definition of the term *sparse matrix* is difficult to formulate. For the purposes of this thesis, however, a matrix is considered sparse if few of its entries are nonzero. Acceptable ranges of nonzero density could be proposed, but J. H. Wilkinson [GMS92] suggested that a matrix is sparse if "it has enough zeros that it pays to take advantage of them". Similarly, classifying the size of a sparse matrix problem is also difficult. Typically, the size of a *large* sparse problem is taken to be of order 1000 or greater, but matrices of several hundred rows and columns may be included if the density of nonzero entries is sufficiently high. Although the thesis only considers matrices with real valued entries, our sparse matrix techniques are easily generalized to complex hermitian matrices.

1.1 A General Introduction to Eigensolvers

As previously outlined, there are a wide range of eigenvalue methods. Each algorithm is designed to exploit particular matrix characteristics, solution requirements, or properties of the computing environment. In this section we provide a general overview of eigenvalue methods for symmetric matrices, identifying the existing algorithms best suited to the solution of our particular sparse eigenvalue problem.

In general, eigenvalue methods for dense symmetric matrices can be grouped into two broad classes of algorithms. Members of the first group are applied directly to the original matrix, while algorithms in the second group work with an *equivalent* intermediate matrix constructed from the original problem.

An example of the first algorithm class is the Jacobi algorithm [Wil65, GV89]. Historically, the Jacobi algorithm was the method of choice for determining all the eigenvalues of dense matrices, until superseded by more modern algorithms. (Recently, there has been renewed interest in variants of the algorithm that are inherently parallel.) The Jacobi algorithm applies sweeps of orthogonal similarity transformations, constructed from plane rotations, that systematically reduce the norm of a matrix's off-diagonal entries. Typically, the Jacobi algorithm requires $O(n^3)^\dagger$ flops‡ to diagonalize a dense matrix.

When large subsets or all the eigenvalues of a dense symmetric matrix are desired the dominant method is the QR (or QL) algorithm. (See [GV89] or [Par80] for a list of references.) Although the QR algorithm could be applied directly to the original dense matrix, a costly QR factorization is required by each iteration of the algorithm. Consequently, the QR algorithm is in the second class of eigenvalue methods that first reduce a matrix to a simplier canonical form with the same eigenvalues. Typically, this reduction is effected by Householder or Givens reductions, which use sequences of orthogonal similarity transformations to reduce a matrix to tridiagonal form in $O(n^3)$ flops. The QR algorithm requires $O(n^2)$ flops to determine the complete spectrum of a tridiagonal matrix.

$^\dagger g(n) = O(f(n))$ means that $g(n) \leq \text{Const.} f(n)$ for sufficiently large n.

‡Following [GV89] a flop is defined to be any floating point arithmetic operation.

Many other eigenvalue methods also use a two stage approach to avoid working with the original matrix directly. For example, the Bisection and Multisectioning algorithms [Wil65, GV89] isolate eigenvalues of a symmetric tridiagonal matrix using Sturm sequencing. Sequentially, the Bisection algorithm is typically used to find a few eigenvalues in a specified region, but it can isolate all eigenvalues of a tridiagonal matrix in $O(n^2)$ flops. In addition, successful variants of both algorithms have been implemented on distributed memory multiprocessors. (See [Jes89] for example.)

Another eigenvalue method for tridiagonal matrices is Cuppen's [Cup81] divide and conquer algorithm. This approach recursively splits a tridiagonal matrix into smaller and smaller tridiagonal subproblems using rank-1 tearings. The recursive tearing process stops when subproblems are on the order of a 2×2 matrix. The eigensystem of the original problem is reconstructed by gluing the eigensystems of the lowest level, and intermediate subproblems, back together using rank-1 modification techniques [Gol73, BNS78]. This divide and conquer approach requires $O(n^2)$ flops to sequentially isolate all eigenvalues. Several researchers have developed parallel versions of Cuppen's algorithm. For example, Dongarra and Sorensen [DS87] successfully implemented Cuppen's algorithm on a shared memory multiprocessor.

The four eigenvalue methods mentioned previously permit the efficient determination of a dense symmetric matrix's complete spectrum. Unfortunately, these approaches are unable to meaningfully exploit matrix sparsity. If the Jacobi algorithm or Givens or Householder reductions are applied to a sparse matrix, typically the orthogonal transformations used by these algorithms produce overwhelming numbers of *fill* entries (new nonzeros) that quickly destroy sparsity. In fact, for most sparse problems almost all zero entries experience fill during the course of the diagonalization or tridiagonalization. As as result, these approaches require $O(n^3)$ flops and $O(n^2)$ storage and sparse matrices might as well be treated as dense.

For several decades the Lanczos algorithm [Par80, GV89] has attracted researchers interested in large sparse eigenvalue problems. This iterative algorithm performs an implicit partial tridiagonalization of a sparse matrix using a three term recurrence relation to di-

rectly compute entries of the tridiagonal matrix. Lanczos leaves the original sparse matrix intact and for the simple Lanczos iteration the core of the computation is a matrix-vector product, which permits exploitation of matrix sparsity. When the iteration terminates, the Ritz values of the partial tridiagonal matrix approximate a subset of the original matrix's eigenvalues. Unfortunately, the loss of orthogonality amongst Lanczos vectors complicates the algorithm's application. Although variants of the Lanczos algorithm have been considered that attempt the computation of the complete spectrum of a sparse symmetric matrix (See [ELT79],[CW79], and [PR81] for example.), in practice Lanczos is best suited to finding small subsets of a sparse matrix's spectrum. As shown in Chapter 6, it appears that none of the current Lanczos variants can reliably and economically find the complete spectrum of most large symmetric sparse matrices, or even moderately large subsets of their eigenvalues.

We complete our brief overview of symmetric eigenvalue methods with direct methods designed to take advantage of sparse matrices in banded form. The first algorithm [MRW71] is a variant of the QR algorithm intended for direct application to symmetric banded matrices. This method is implemented as routine BQR of EISPACK [GBDM77]. Because the QR iteration is band-preserving, this approach avoids the extreme fill difficulties of previous methods. Although BQR is intended for finding only a few eigenvalues, all eigenvalues of a bandwidth[§] b matrix can be isolated using $O(b^2n^2)$ flops. When b is in the range $2 \leq b < \sqrt{n}$, the banded QR approach is theoretically faster than using an $O(n^3)$ flop direct tridiagonalization and an $O(n^2)$ tridiagonal eigenvalue method. The next algorithm we consider, however, dramatically improves the tridiagonalization of banded matrices.

As previously mentioned, Givens or Householder reductions create many new nonzeros, quickly filling the entries outside the band of the unreduced portion of the matrix. Alternatively, the band-preserving algorithm of Rutishauser [Rut63] and Schwarz [Sch71] tridiagonalizes a symmetric banded matrix with $O(bn^2)$ flops and $O(bn)$ storage. The algorithm uses Givens transformations to zero band entries column by column. Between the elimination of each band nonzero, fill entries created outside the band are chased off

[§]Bandwidth (or semi-bandwidth) is defined as $b = \max_{i,j \in \{1 \ldots n\}, i \neq j} |i - j|$ such that $A_{ij} \neq 0$.

the end of the matrix with *bulge chasing* transformations. The Rutishauser-Schwarz algorithm is implemented as EISPACK's BANDR routine [GBDM77]. LAPACK's [ABB+92] version of this routine, SSBTRD, is based on Kaufman's [Kau84] vectorized version of the Rutishauser-Schwarz algorithm. Recent work by Lang [Lan92] and Bischof and Sun [BS92] presents parallel algorithms, closely related to Rutishauser-Schwarz, that investigate the use of multiple elimination and delayed bulge chasing techniques.

These algorithms are primarily designed for the tridiagonalization of densely banded matrices. There are many well-established heuristic algorithms [Cut72, GPS76a, CCDG82, Lew82], however, for identifying bandwidth reducing preorderings of general symmetric sparse matrices. (These preorderings are formulated as similarity transformations, avoiding changes to a matrix's eigenvalues.) Combining such a preordering scheme with the Rutishauser-Schwarz tridiagonalization and an $O(n^2)$ tridiagonal eigenvalue method, is the best published sequential direct method for finding the complete spectrum of a symmetric sparse matrix. The popular numerical software system MATLAB [Mat92] includes this approach for the solution of sparse eigenvalue problems. Although this scheme takes some advantage of zero entries, for many problems sparsity exploitation is limited. This method is almost completely dependent upon the bandwidth reduction algorithm to take maximum advantage of sparsity, but typically the selected preordering leaves the band of the permuted matrix relatively sparse. Unfortunately, the Rutishauser-Schwarz algorithm rapidly fills the sparse band and further opportunity to exploit band sparsity is lost. If matrix sparsity could be more fully exploited, the potential exists for dramatic improvements in the efficiency of sparse tridiagonalization techniques.

1.2 Thesis Statement

Existing algorithms for determining the complete spectrum of sparse matrices, or moderately large subsets of their eigenvalues, are well suited for densely banded problems, but are sub-optimal for symmetric matrices with more irregular sparsity structures. This dissertation develops sequential algorithms that more fully exploit the sparsity of symmetric

matrices while determining their eigenvalues. The approach selected by this research is to reexamine the reduction of symmetric matrices to tridiagonal form, developing novel sparse tridiagonalization algorithms. The class of algorithms studied have the following generic form,

$$
\begin{aligned}
&A_0 := A \\
&\text{FOR } i := 1, 2, \ldots, k \\
&\qquad A_i := Q_i^T A_{i-1} Q_i
\end{aligned}
$$

in which A is systematically reduced to tridiagonal form ($A_k = T$) using a sequence of k carefully selected orthogonal similarity transformations $Q_i^T A_{i-1} Q_i$.

The following points outline this dissertation's approach to the design and development of new sparse tridiagonalization techniques.

- Representative sparse model problems conducive to formal analysis are not available for many practical sparse problems. Consequently, we support algorithm development using a combination of experimental and theoretical complexity analysis.
- We develop symbolic tridiagonalization tools to facilitate the use of experimental evidence as a research tool.
- We characterize the underlying fill properties of sparse tridiagonalization and identify the difficulties and limitations associated with existing direct tridiagonalization methods extended for use with sparse matrices.
- Using formal complexity analysis, experimental results and insight gained from the analysis of existing algorithms, we design and develop novel tridiagonalization algorithms which take better advantage of matrix sparsity.

The novel algorithms developed in this dissertation are assessed using both formal analysis and experimental comparison to existing direct methods. Given the shortage of representative model problems, however, we emphasize experimentation with symbolic and

numeric implementations of the new algorithms. An important goal of this work is to develop sparse tridiagonalization techniques that are generally applicable. Consequently, all experiments are conducted using a large test suite of sparse problems drawn from a broad range of applications.

To benchmark the success of our sparse tridiagonalization schemes relative to other eigenvalue approaches, we investigate the ability of Lanczos type algorithms to solve our sparse eigenvalue problem. Once again, comparison with our sparse tridiagonalization techniques employ both theoretical and experimental analysis.

1.3 Thesis Contributions

The primary contributions of this thesis's research are:

- Development of a framework for the complexity analysis of algorithms employing sequences of Givens transformations.
- An improved understanding of the fill properties associated with the reduction of sparse symmetric matrices to tridiagonal form.
- An enhanced formal and experimental analysis of existing algorithms extended for the tridiagonalization of general sparse symmetric matrices.
- Development of the symbolic sparse reduction tools `Xmatrix` and `Trisymb` and a graph-theoretic model for symmetric sparse tridiagonalization and partial bandwidth contraction.
- Development of the Bandwidth Contraction algorithm, demonstrating that sparsity exploitation can be improved by rearranging the elimination sequence of nonzeros.
- Development of the Hybrid Bandwidth Contraction algorithm, which combines sparsely and densely banded tridiagonalization algorithms to improve efficiency.
- Development of the Split Bandwidth Contraction algorithm to reduce bulge chasing transformation costs and increase the efficiency of partial bandwidth contractions.

- Development of the transition strategies based on formal analysis leading to the Hybrid Split Bandwidth Contraction algorithm.
- Extensive experimental analysis demonstrating the efficiency of the new sparse tridiagonalizations algorithms relative to existing direct methods.
- A theoretical and experimental comparison of sparse tridiagonalization based eigensolvers to variants of the Lanczos algorithm, demonstrating the efficiency of the novel sparse tridiagonalization algorithms.

1.4 Thesis Overview

The remainder of the thesis is organized in the following manner.

Chapter 2 provides an overview of relevant background material and introduces a number of novel sparse tridiagonalization tools created to support the development and evaluation of sparse reduction algorithms. The chapter begins with general definitions and notations, followed by a brief description of a large test suite of sparse symmetric problems, and a review of Givens and Householder transformations and their fill properties. It then introduces a general analysis framework for algorithms employing sequences of Givens similarity transformations. The following two sections of the chapter describe the newly developed symbolic reduction tools `Xmatrix` and `Trisymb` that manipulate sparsity structures to predict the computational requirements of sparse reductions. Finally, we introduce a graph-theoretic model for the application of orthogonal transformations to sparse symmetric matrices.

Using these sparse reduction tools, Chapter 3 explores existing direct tridiagonalization approaches, analyzing their limitations and their potential extension to improve sparsity exploitation. Employing two model problems and practical sparse matrices, we first characterize the fill properties of both standard and customized Givens Reduction algorithms. As alternatives to these algorithms, we investigate the Rutishauser-Schwarz and Lang band-preserving tridiagonalization methods for banded matrices, which restrict the accumulation

of fill entries to improve reduction efficiency. Chapter 3 concludes with the development of algorithms generalizing these band-preserving techniques for the tridiagonalization of general sparse symmetric matrices, and analysis of the relative merits of the resulting sparse reduction methods.

In response to the limitations of the previous sparse reduction algorithms, Chapter 4 develops alternative approaches to sparse tridiagonalization that also use bandwidth reducing preorderings and band-preserving reduction techniques, but rearrange the elimination of nonzero entries to more fully exploit internal band sparsity. Chapter 4 first describes the development of the Bandwidth Contraction algorithm, or BC, whose diagonally-oriented reduction allows it to significantly reduce the tridiagonalization costs of many practical sparse problems. Employing theoretical analyses of the Rutishauser-Schwarz and Bandwidth Contraction algorithms, we then guide the development of the effective hybrid tridiagonalization algorithm HYBBC, which exploits the best characteristics of both algorithms to form a versatile and efficient two stage sparse reduction process. Following a discussion of key aspects of the numerical implementation of BC and HYBBC, extensive experimentation with a large test suite of practical sparse problems is used to evaluate the success of both sparse tridiagonalization algorithms. The chapter's final section investigates the relationship between sparsity structures, preorderings and the performance of the Bandwidth Contraction algorithm using additional experiments with practical sparse problems.

The algorithms developed in Chapter 4 demonstrate the ability of diagonally-oriented reductions to take advantage of band sparsity. Chapter 5 expands upon these successful techniques, developing second generation sparse algorithms for bandwidth contraction and tridiagonalization. The chapter begins with the development of the Split Bandwidth Contraction algorithm, or SBC, which employs bidirectional elimination techniques to enhance band sparsity exploitation and potentially halve the computational requirements of each diagonal's reduction. To evaluate the efficiency of performing partial bandwidth contractions with SBC, we provide an extensive experimental comparison of BC and SBC using both symbolic and numeric implementations.

Building on the success of SBC, the novel Hybrid Split Bandwidth Contraction algorithm, or HYBSBC, incorporates many aspects of HYBBC's approach, but replaces the BC stage with the Split Bandwidth Contraction algorithm. The new hybrid algorithm lends itself to formal analysis, permitting the development of the Δ-transition strategy for precise regulation of the transition between the algorithm's two stages. Once again, a numerical implementation of HYBSBC is described before using a wide variety of experiments with practical sparse problems to evaluate the performance of HYBSBC and the optimality of the Δ-transition strategy.

To investigate the relative efficiency of sparse tridiagonalization based eigenvalue methods, Chapter 6 compares the resource requirements of variants of the Lanczos algorithm with the Hybrid Split Bandwidth Contraction algorithm. The chapter starts with a brief overview of the mathematics underlying the basic Lanczos iteration and then surveys different techniques employed by practical Lanczos implementations. The presentation of each Lanczos approach includes a theoretical discussion of its ability to efficiently compute the complete spectrum of a sparse symmetric problem. To support this analysis experiments are conducted with two well regarded Lanczos codes and practical sparse problems. The resource requirements of these codes to compute subsets of a sparse problem's eigenvalues, ranging in size from moderate fractions of its eigenvalues to its complete spectrum, are compared to the requirements of HYBSBC and the TQLRAT routine from EISPACK.

Chapter 7 concludes the thesis by providing a summary of its major results and conclusions, and identifying directions of possible future research.

Chapter 2

Background Material and Sparse Tridiagonalization Tools

This chapter provides an overview of relevant background material and introduces a number of sparse tridiagonalization tools developed to support the research presented in the remainder of the dissertation. We begin the chapter with general definitions and notation. The following section briefly describes the large test suite of sparse symmetric problems used in the extensive experiments of later chapters. We then provide a review of Givens and Householder transformations, emphasizing their relationship to sparsity structures and fill production. Section 2.4 introduces a general analysis framework for algorithms using sequences of Givens similarity transformations. The following two sections introduce `Xmatrix` and `Trisymb`: symbolic reduction tools for small sparse and sparsely banded symmetric matrices respectively. Both tools manipulate sparsity structures to predict the computational requirements of sparse reductions. Finally, Section 2.7 introduces a graph-theoretic model for the application of orthogonal transformations to symmetric sparse matrices.

2.1 Notation and Definitions

This section outlines some general definitions and notation used in the remainder of the dissertation.

Symbol Conventions: Unless mentioned otherwise, an uppercase Roman letter represents a matrix, while a lowercase Roman letter represents a vector or scalar value.

$A_{i,j}$: The entry in row i and column j of matrix A is denoted by $A_{i,j}$.

Matrix Bandwidth: The bandwidth (sometimes called semi-bandwidth) of A is defined as $b = \max_{i,j \in \{1...n\}, i \neq j} |i - j|$ such that $A_{i,j} \neq 0$.

flop: Following [GV89] a flop is defined to be any floating point arithmetic operation.

2.2 Sparse Symmetric Test Problems

As outlined in Section 1.2, experimentation with practical sparse problems plays a key role in the development and evaluation of sparse tridiagonalization algorithms. Table 2.1 lists the identifier and size of a suite of 115 test problems selected from the Harwell–Boeing sparse matrix collection [DGLP82, GLD92]. The originating applications for Table 2.1 sparse problems include: air traffic control, structural engineering, oceanic modeling and power system networks. A description of each matrix's characteristics is provided in [GLD92]. When the Harwell–Boeing collection only specifies a matrix's sparsity pattern, we assign a random value in the range $(0.0, 1.0]$ to each nonzero entry.

By no means all problems in Table 2.1 originated as eigenvalue problems. In addition, Table 2.1 also contains many small matrices, even though a primary theme of this dissertation is the tridiagonalization of large sparse problems. The test suite includes these additional problems because we seek general reduction techniques that are suitable for a broad range of sparsity patterns.

2.3 Orthogonal Transformations and Sparse Tridiagonalization

As discussed in Section 1.2, this dissertation studies the class of sparse tridiagonalization and partial reduction algorithms that use a sequence of orthogonal similarity transformations to eliminate unwanted nonzero entries. These algorithms could construct a similarity

Problem	n	Problem	n	Problem	n	Problem	n
1138 BUS	1138	BCSSTK21	3600	DWT 59	59	DWT 2680	2680
494 BUS	494	BCSSTK22	138	DWT 66	66	ERIS1176	1176
662 BUS	662	BCSSTK23	3134	DWT 72	72	GR 30 30	900
685 BUS	685	BCSSTK24	3562	DWT 87	87	LSHP 265	265
ASH292	292	BCSSTK26	1922	DWT 162	162	LSHP 406	406
ASH85	85	BCSSTK27	1224	DWT 193	193	LSHP 577	577
BCSPWR01	39	BCSSTK28	4410	DWT 198	198	LSHP 778	778
BCSPWR02	49	BCSSTM07	420	DWT 209	209	LSHP1009	1009
BCSPWR03	118	BCSSTM10	1086	DWT 221	221	LSHP1270	1270
BCSPWR04	274	BCSSTM27	1224	DWT 234	234	LSHP1561	1561
BCSPWR05	443	BLCKHOLE	2132	DWT 245	245	LSHP1882	1882
BCSPWR06	1454	CAN 24	24	DWT 307	307	LSHP2233	2233
BCSPWR07	1612	CAN 61	61	DWT 310	310	LSHP2614	2614
BCSPWR08	1624	CAN 62	62	DWT 346	346	LSHP3025	3025
BCSPWR09	1723	CAN 73	73	DWT 361	361	LSHP3466	3466
BCSSTK01	48	CAN 96	96	DWT 419	419	LUND A	147
BCSSTK02	66	CAN 144	144	DWT 492	492	LUND B	147
BCSSTK03	112	CAN 161	161	DWT 503	503	NOS1	100
BCSSTK04	132	CAN 187	187	DWT 512	512	NOS2	237
BCSSTK05	153	CAN 229	229	DWT 592	592	NOS3	957
BCSSTK06	420	CAN 256	256	DWT 607	607	NOS4	960
BCSSTK07	420	CAN 268	268	DWT 758	758	NOS5	468
BCSSTK08	1074	CAN 292	292	DWT 869	869	NOS6	675
BCSSTK09	1083	CAN 445	445	DWT 878	878	NOS7	729
BCSSTK10	1086	CAN 634	634	DWT 918	918	PLAT1919	1919
BCSSTK11	1473	CAN 715	715	DWT 992	992	PLAT362	362
BCSSTK12	1473	CAN 838	838	DWT 1005	1005	SSTMODEL	3345
BCSSTK19	817	CAN 1054	1054	DWT 1007	1007	ZENIOS	2873
BCSSTK20	485	CAN 1072	1072	DWT 1242	1242	—	—

Table 2.1: A Test Suite of Sparse Symmetric Matrices

transformation using one of many different orthogonal transformations. Givens rotations, $G(i,j,\theta)$, are often chosen for sparse matrix applications because they permit fine grained control over the elimination of nonzeros and the introduction of fill entries. In fact rotations can be constructed to zero any entry of a matrix using *any* other entry in its row or column as the *zeroing entry.* (In Figure 2.1, x_1 is the zeroing entry.) The trivial example

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 & x & & x & x & \\ x_2 & & x & x & & x \end{bmatrix} \longrightarrow \begin{bmatrix} \hat{x}_1 & x & x & x & x & x \\ 0 & x & x & x & x & x \end{bmatrix}$$

Figure 2.1: Sparsity Structure Unioning by a Givens Rotation

in Figure 2.1 illustrates an important property of rotations which impacts matrix sparsity. If cancellation is ignored and x_1 is nonzero, the sparsity structure of both modified rows is the union of the structure of the two rows prior to the rotation's application. In the special cases of x_2 or x_1 zero, the rotation is either the identity or swaps the sparsity structure of the two rows, avoiding the introduction of fill entries. It is advantageous to have a zero *zeroing entry.* In addition to fill avoidance, swapping matrix rows to eliminate nonzero x_2 requires no floating point computations.

Tridiagonalization algorithms can use Givens rotations to construct orthogonal similarity transformations of the form $G(i,j,\theta)^T A\, G(i,j,\theta)$, which modify both rows and columns i and j of A. By exploiting the symmetry of sparse problems and of similarity transformations, algorithms need only consider transformation modifications to either the upper or lower triangular portion of each matrix. Without loss of generality we work with a matrix's lower triangular portion. From the wide selection of Givens rotations that can be constructed to eliminate a particular nonzero entry, many algorithms presented in later chapters use *adjacent rotations* exclusively. A rotation $G(i,j,\theta)^T$ is considered adjacent if $|i-j| = 1$.

To assess the stability of Givens similarity transformations, suppose k transformations of the form $G_i^T B G_i$ are applied to matrix B. If the updated matrix $\widehat{B}$ is calculated using finite precision floating point operations, Wilkinson [Wil65] shows that in the absence of

roundoff

$$\widehat{B} = G_k^T \ldots G_1^T (B + E) G_1 \ldots G_k, \tag{2.1}$$

where $\| E \|_2 \leq c\mu \| B \|_2$, constant c is dependent on n and k, and μ is the machine precision. In other words, $\widehat{B}$ is exactly similar to a matrix close to B.

Another commonly used orthogonal transformation is a Householder transformation; an $n \times n$ orthogonal matrix of the form

$$H = I - 2vv^T / v^T v,$$

where $v \in \mathbb{R}^n$. In general, a Householder transformation can be constructed to reflect any given vector onto any other vector of the same dimension and equal l_2-norm. Consequently, we can construct a Householder transformation that simultaneously annihilates multiple entries from a matrix's column, when applied from the left.

Compared to Givens rotations, applying Householder transformations to sparse matrices generally creates higher levels of fill. A Givens rotation applied to the left of sparse matrix A to eliminate a single nonzero, unions the sparsity structure of two rows. In contrast, for a Householder transformation simultaneously eliminating k entries of a matrix's column, the final structure of each modified row is the union of the sparsity structures of all modified rows prior to the transformation's application. Duff and Reid [DR75] show there exists a sequence of k Givens rotations, $G_k, \ldots, G_1$, able to eliminate the same group of nonzeros such that

$$sparsity\_structure(G_k \ldots G_2 G_1 A) \subseteq sparsity\_structure(HA).$$

2.4 A Complexity Analysis Framework for Sparse Tridiagonalization Algorithms and Symbolic Reduction Tools

This section introduces a general framework for analyzing the complexity of algorithms applying sequences of Givens similarity transformations to sparse symmetric matrices. This framework guides the formal analyses of many partial reduction and tridiagonalization algorithms presented in subsequent chapters. Examples of detailed analyses performed

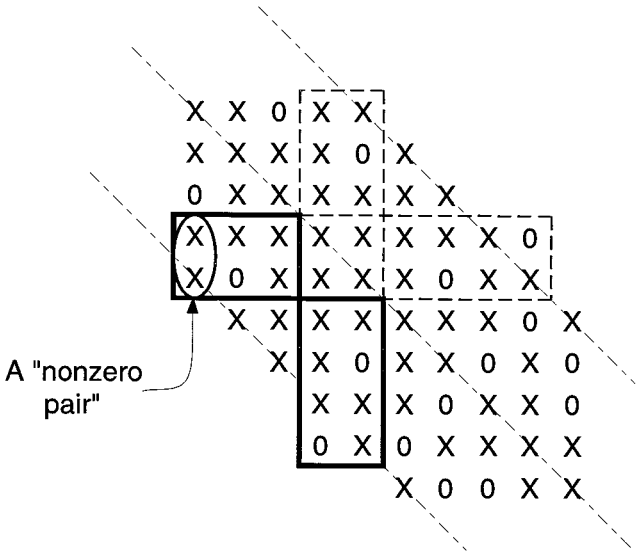


Figure 2.2: A Transformation Length Example.

using this framework are provided in [Cav93]. The general framework is also used by the symbolic reduction tools `Xmatrix` and `Trisymb` (See Sections 2.5 and 2.6.) to coordinate the gathering of flop count statistics.

To simplify the resolution of each algorithm's computational requirements, the framework directs each analysis to be split into two sub-tasks.

Task 1: Calculate the number of nontrivial transformations, T_{total} , employed by the reduction.

Task 2: Calculate the total number of off-diagonal, lower triangular pairs of nonzero entries modified by the reduction's nontrivial transformations. We refer to this value as the total *transformation length* or L_{total} .

The first sub-task is self-evident but the second requires additional clarification. The length of a single transformation is the number of pairs of lower triangular nonzero entries it modifies, excluding those entries updated by both rotations constituting the transformation. We consider a pair of modified entries *nonzero* if one or both entries are nonzero. As an example, the length of the transformation modifying the highlighted entries of Figure 2.2's matrix is 7. (Section 2.4.3 considers a specialized variant of the analysis framework, for densely banded matrices, that exploits the sparsity of a pair of entries creating a *bulge*.) We

note that a transformation's length is equal to the total number of pairs of nonzero entries on both sides of the main diagonal affected by the application of $G(i,j,\theta)^T$. As a result, it is often easier to consider the number of pairs of nonzero entries modified by a single rotation when symmetry is ignored, rather than apply the strict definition of transformation length.

Once an individual analysis has found T_{total} and L_{total}, we use the following general formula to calculate the algorithm's flop requirements.

$$\text{Total\_flops} = (F_{\text{trans}})(T_{\text{total}}) + (F_{\text{pair}})(L_{\text{total}}) + \text{OTC} \tag{2.2}$$

F_{trans} represents the number of flops required to construct a transformation and apply it to the entries modified by both the transformation's rotations. F_{pair} represents the number of flops required to apply a rotation to a single pair of nonzero entries. OTC represents one time costs that are not spread over individual transformations. The total flop count does not include the cost of square roots, which are separately accounted for by the analysis framework.

Section 2.3 introduced the standard Givens transformation. To reduce computational requirements many algorithms actually employ a "root free" variant of the transformation, with identical fill properties, often referred to as a *fast Givens transformation* [Gen73]. The specific values of F_{trans}, F_{pair}, and OTC are dependent upon the type of Givens transformation used by an algorithm. The following two subsections refine Equation 2.2 for each transformation type. Finally, Section 2.4.3 provides a specialized analysis framework for densely banded matrices reduced by band-preserving algorithms. (Band-preserving reductions are introduced in Chapter 3.)

2.4.1 Standard Givens Transformations

F_{pair}

A standard 2×2 Givens rotation has the generic form $\begin{bmatrix} c & -s \\ s & c \end{bmatrix}$. Applying this rotation

to a typical pair of entries $\begin{bmatrix} c & -s \\ s & c \end{bmatrix}^T \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ requires

$$F_{\text{pair}} = 6 \text{ flops.} \tag{2.3}$$

F_{trans}

The calculation of c and s requires 5 flops and one square root [GV89]. The cost of updating the 3 lower triangular entries modified by both rotations making up the transformation requires more detailed consideration. By using the following scheme, we save 3 flops over the most obvious approach.

$$\begin{aligned}
\begin{bmatrix} \hat{a}_{ii} & \hat{a}_{ij} \\ \hat{a}_{ji} & \hat{a}_{jj} \end{bmatrix} &= \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix} \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \\
&= \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} ca_{ii} + sa_{ij} & -sa_{ii} + ca_{ij} \\ ca_{ji} + sa_{jj} & -sa_{ji} + ca_{jj} \end{bmatrix} \\
&= \begin{bmatrix} c^2a_{ii} + csa_{ij} + csa_{ji} + s^2a_{jj} & -csa_{ii} + c^2a_{ij} - s^2a_{ji} + csa_{jj} \\ -csa_{ii} - s^2a_{ij} + c^2a_{ji} + csa_{jj} & s^2a_{ii} - csa_{ij} - csa_{ji} + c^2a_{jj} \end{bmatrix} \\
&\quad \text{but } a_{ji} = a_{ij} \\
&= \begin{bmatrix} c^2a_{ii} + 2csa_{ji} + s^2a_{jj} & (c^2 - s^2)a_{ji} + cs(a_{jj} - a_{ii}) \\ (c^2 - s^2)a_{ji} + cs(a_{jj} - a_{ii}) & s^2a_{ii} - 2csa_{ji} + c^2a_{jj} \end{bmatrix}
\end{aligned} \tag{2.4}$$

The total number of flops required to compute the final value of the twice modified entries $\hat{a}_{ii}$, $\hat{a}_{jj}$, and $\hat{a}_{ji}$, is summarized in the following table. Each calculation is free to use those values appearing to the left of it in the table.

Calculation	c^2	cs	s^2	$2csa_{ji}$	$\hat{a}_{ii}$	$\hat{a}_{jj}$	$\hat{a}_{ji}$	Total
Flops	1	1	1	2	4	4	5	18

Finally, it is not necessary to calculate the updated value of the eliminated entry, saving 3 flops for each transformation. Thus for standard Givens transformations

$$F_{\text{trans}} = 5 + 18 - 3 = 20 \text{ flops.} \tag{2.5}$$

OTC

There are no one time costs associated with tridiagonalization algorithms using standard Givens transformations.

Standard Givens Flop Formula

For standard Givens transformations Equation 2.2 becomes

$$\text{Total\_flops\_SG} \quad = \quad 20(T_{\text{total}}) + 6(L_{\text{total}}) \tag{2.6}$$

In addition to this flop count, the reduction requires T_{total} square roots.

2.4.2 Fast Givens Transformations

In this section we assume that the reader is familiar with the fast Givens transformation presentation of [GV89]. Suppose that a series of fast Givens transformations are accumulated in a single similarity transformation $Q^T AQ$. In this case Q is equivalent to the product of a series of standard Givens rotations. The novel idea behind the fast Givens approach is to represent Q as the product of two matrices $MD^{-1/2}$. D is a diagonal matrix that is initially set to the identity. As the reduction proceeds the effects of each transformation are accumulated in D

$$D_{\text{new}} = M^T DM \tag{2.7}$$

and this portion of the transformation is finally applied to the tridiagonal matrix at the end of the reduction.

$$T_{\text{final}} = D^{-1/2} T D^{-1/2} \tag{2.8}$$

On the other hand, each M is applied to A immediately to effect the elimination of nonzero entries. Following the presentation of [GV89], and using a 2×2 example for simplicity, M can take one of two forms. We assume that M^T is applied to $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ to zero x_2.

$$M_1 = \begin{bmatrix} \beta_1 & 1 \\ 1 & \alpha_1 \end{bmatrix} \qquad M_2 = \begin{bmatrix} 1 & \alpha_2 \\ \beta_2 & 1 \end{bmatrix}$$
$$\text{where } \alpha_1 = \tfrac{-x_1}{x_2} \quad \beta_1 = -\alpha_1(\tfrac{d_2}{d_1}) \qquad \text{where } \alpha_2 = \tfrac{-x_2}{x_1} \quad \beta_2 = -\alpha_2(\tfrac{d_1}{d_2})$$

F_{pair}

Applying M_1 or M_2 to a typical pair of entries

$$\begin{bmatrix} \beta_1 & 1 \\ 1 & \alpha_1 \end{bmatrix}^T \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 1 & \alpha_2 \\ \beta_2 & 1 \end{bmatrix}^T \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \tag{2.9}$$

requires

$$F_{\text{pair}} = 4 \text{ flops.} \tag{2.10}$$

F_{trans}

We consider the cost of updating the 3 lower triangular entries modified by both M^T and M in detail. The cost of updating these entries using transformations constructed from either M_1 or M_2 is identical. Without loss of generality the following analysis considers M_1.

$$\begin{aligned}
\begin{bmatrix} \hat{a}_{ii} & \hat{a}_{ij} \\ \hat{a}_{ji} & \hat{a}_{jj} \end{bmatrix} &= \begin{bmatrix} \beta_1 & 1 \\ 1 & \alpha_1 \end{bmatrix} \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix} \begin{bmatrix} \beta_1 & 1 \\ 1 & \alpha_1 \end{bmatrix} \\
&= \begin{bmatrix} \beta_1 & 1 \\ 1 & \alpha_1 \end{bmatrix} \begin{bmatrix} \beta_1 a_{ii} + a_{ij} & a_{ii} + \alpha_1 a_{ij} \\ \beta_1 a_{ji} + a_{jj} & a_{ji} + \alpha_1 a_{jj} \end{bmatrix} \\
&= \begin{bmatrix} \beta_1^2 a_{ii} + \beta_1 a_{ij} + \beta_1 a_{ji} + a_{jj} & \beta_1 a_{ii} + \beta_1 \alpha_1 a_{ij} + a_{ji} + \alpha_1 a_{jj} \\ \beta_1 a_{ii} + a_{ij} + \beta_1 \alpha_1 a_{ji} + \alpha_1 a_{jj} & a_{ii} + \alpha_1 a_{ij} + \alpha_1 a_{ji} + \alpha_1^2 a_{jj} \end{bmatrix}
\end{aligned}$$

but $a_{ji} = a_{ij}$

$$= \begin{bmatrix} \beta_1^2 a_{ii} + 2\beta_1 a_{ji} + a_{jj} & \beta_1 a_{ii} + \beta_1 \alpha_1 a_{ji} + a_{ji} + \alpha_1 a_{jj} \\ \beta_1 a_{ii} + a_{ji} + \beta_1 \alpha_1 a_{ji} + \alpha_1 a_{jj} & a_{ii} + 2\alpha_1 a_{ji} + \alpha_1^2 a_{jj} \end{bmatrix} \tag{2.11}$$

The total number of flops required to compute the final value of the twice modified entries $\hat{a}_{ii}$, $\hat{a}_{jj}$, and $\hat{a}_{ji}$, is summarized in the following table. Each calculation is free to use those values appearing to the left of it in the table.

Calculation	$\beta_1 a_{ii}$	$\beta_1 a_{ji}$	$2(\beta_1 a_{ji})$	$\beta_1(\beta_1 a_{ii})$	$\alpha_1(\beta_1 a_{ji})$	$\alpha_1 a_{jj}$	...
Flops	1	1	1	1	1	1	

...	$2\alpha_1 a_{ji}$	$\alpha_1(\alpha_1 a_{jj})$	$\hat{a}_{ii}$	$\hat{a}_{jj}$	$\hat{a}_{ji}$	Total
	2	1	2	2	3	16

The next component of F_{trans} is the cost of updating the diagonal matrix D. For the moment we assume the first fast Givens transformation type has been selected.

$$\begin{aligned}
\begin{bmatrix} \hat{d}_{ii} & 0 \\ 0 & \hat{d}_{jj} \end{bmatrix} &= M_1^T D M_1 \\
&= \begin{bmatrix} d_{jj}(1 - \alpha_1 \beta_1) & 0 \\ 0 & d_{ii}(1 - \alpha_1 \beta_1) \end{bmatrix}
\end{aligned} \tag{2.12}$$

In this case, the calculation of $\hat{d}_{ii}$ and $\hat{d}_{jj}$ requires a total of 4 flops.

Determining the cost of constructing a fast Givens transformation is complicated by the required choice between two transformation types. The normal procedure is to first calculate α_1 and β_1 using 3 flops. To check the stability of this first transformation, the magnitude of $(1 - \alpha_1\beta_1)$ is evaluated. (The cost of computing $(1 - \alpha_1\beta_1)$ is included in the cost of updating D.) If $(1 - \alpha_1\beta_1)$ is too large, the second fast Givens transformation type must be used and computing α_2 and β_2 requires 3 additional flops. Assuming the value of $\alpha_1\beta_1$ is saved, the new scaling factor $(1 - \alpha_2\beta_2)$ can be computed from $-(1 - \alpha_1\beta_1)/\alpha_1\beta_1$ using one additional flop. If we assume that 1/2 of the transformations employed are type 2, constructing the average fast Givens transformation requires

$$\frac{1}{2}(3+3+1) + \frac{1}{2}(3) = 5\text{flops}. \tag{2.13}$$

Finally, it is not necessary to calculate the updated value of the eliminated entry, saving 2 flops for each transformation. Thus for fast Givens transformations

$$F_{\text{trans}} = 16 + 4 + 5 - 2 = 23 \text{ flops}. \tag{2.14}$$

OTC

When A has been reduced to tridiagonal form, the fast Givens process is completed as shown by equation 2.8. The calculation of $D^{1/2}$ requires N square roots. The following equation illustrates the modifications made to the tridiagonal matrix by entry $d_i^{-1/2}$.

$$\begin{bmatrix} \ddots & & \\ & d_i^{-1/2} & \\ & & \ddots \end{bmatrix} \begin{bmatrix} \ddots & b_i & \\ b_i & a_i & b_{i+1} \\ & b_{i+1} & \ddots \end{bmatrix} \begin{bmatrix} \ddots & & \\ & d_i^{-1/2} & \\ & & \ddots \end{bmatrix}$$

$$= \begin{bmatrix} & b_i/d_i^{-1/2} & \\ b_i/d_i^{-1/2} & a_i/d_i & b_{i+1}/d_i^{-1/2} \\ & b_{i+1}/d_i^{-1/2} & \end{bmatrix} \tag{2.15}$$

By exploiting symmetry this update requires 3 flops. Generalizing this result to the cost of the entire update

$$\text{OTC} = 3n \text{ flops}. \tag{2.16}$$

Fast Givens Flop Formula

For fast Givens transformations Equation 2.2 becomes

$$\text{Total\_flops\_FG} = 23(T_{\text{total}}) + 4(L_{\text{total}}) + 3n. \tag{2.17}$$

In addition to this flop count, n square roots are required by the reduction.

As discussed in [GV89], fast Givens transformations require periodic rescaling to avoid overflow problems. Rescaling costs are difficult to predict and are not included in the analysis leading to Equation 2.17. Fortunately, in Section 4.3.4 we demonstrate that rescaling costs are typically insignificant when sparse tridiagonalization algorithms are applied to large problems. Recently, Anda and Park [AP94] presented new fast plane rotations that do not require periodic rescaling. To facilitate direct comparison with existing code, however, the implementations developed for this thesis employ standard fast Givens transformations.

2.4.3 An Enhanced Framework for Densely Banded Matrices

In the general framework described above we increment transformation length if one or both entries in a modified pair are nonzero. For densely banded matrices, those transformations creating a *bulge* modify a single entry pair with only one nonzero. (As described in Chapter 3, a bulge is a fill entry created outside the current band.) The zero entry in this pair is filled by the bulge. If the sparsity of this modified pair is exploited, each fast Givens transformation creating a bulge saves 3 flops, while a standard Givens transformation save 4 flops. If CR is the total number of nontrivial bulge chasing transformations used by a band-preserving reduction, then the enhanced flop formulas are given by the following equations.

$$\text{Total\_flops\_SG}' = 20(T_{\text{total}}) + 6(L_{\text{total}}) - 4CR \tag{2.18}$$

$$\text{Total\_flops\_FG}' = 23(T_{\text{total}}) + 4(L_{\text{total}}) + 3n - 3CR \tag{2.19}$$

2.5 Xmatrix

`Xmatrix` is an interactive symbolic sparse matrix reduction tool. It assumes exact numerical cancellation does not occur and manipulates the sparsity structures of symmetric matrices to model the application of sequences of Givens similarity transformations.

An example of `Xmatrix`'s mouse driven graphical interface, written for the X Window System using the Athena widget set, is illustrated in Figure 2.3. The dominant feature of the interface is the matrix situated in the top left corner of the `Xmatrix` window. The size of this matrix is specified by the user at start-up. Due to the physical constraints of the interface, however, the order of the largest matrix that can be easily manipulated is 50. Each entry of the matrix is a button that can be selected by different mouse buttons with varying effects. The menu to the right of the matrix provides the user access to `Xmatrix`'s different features. For example, the `Load`, `Dump`, `Permute` and `Movie` buttons activate pop-up windows, which are also displayed in Figure 2.3. Finally, the area below the matrix records statistics for the current reduction.

`Xmatrix` assumes that all sparse matrices are symmetric and that their main diagonal is nonzero. Otherwise, a user is completely free to construct an arbitrary sparsity pattern with one of two mechanisms. First, a user can interactively construct a sparse matrix by toggling the nonzero status of entries selected with the middle mouse button. Alternatively, existing sparsity structures can be loaded from files in Harwell–Boeing or `Xmatrix` format using the `Load` pop-up window. Generally, Xmatrix marks nonzero entries with an "`X`", but in keeping with the band-preserving reduction algorithms of Chapters 3, 4 and 5, a nonzero entry lying outside the current band is marked by a "`B`".

Once the desired sparsity structure has been established, a user can model the effects of applying a sequence of standard or fast Givens similarity transformations. Each transformation is interactively specified with the mouse. First, the nonzero entry to be eliminated is selected from the lower or upper triangular portion of the matrix. Using the mouse to select the zeroing entry (see Section 2.3) completes the similarity transformation's definition, causing `Xmatrix` to automatically update the sparsity of the modified pair of rows and

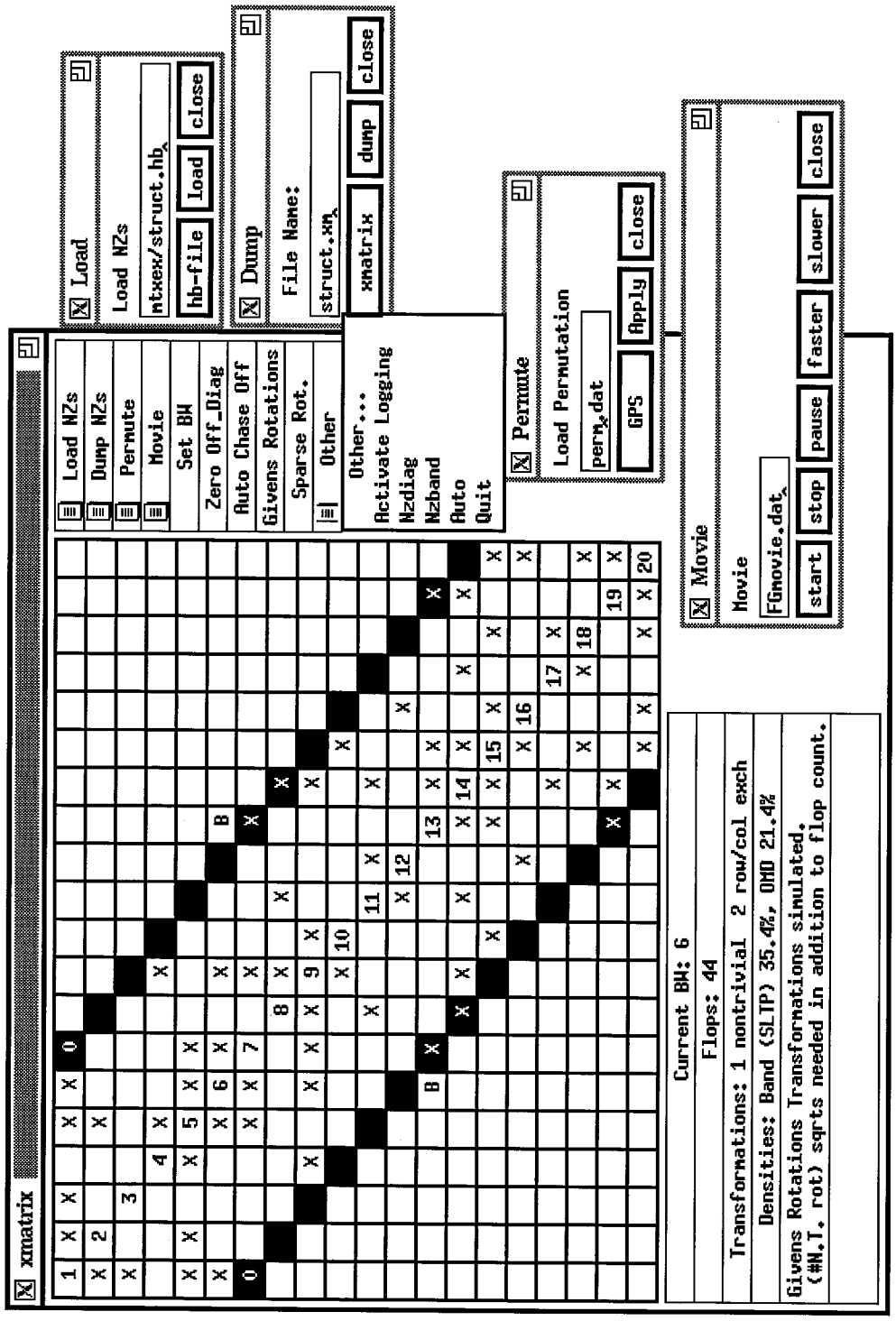


Figure 2.3: The Xmatrix Interface

columns, and zero the target entry. On completion of these updates another transformation may be selected.

The statistics area of the `Xmatrix` window displays the matrix's current bandwidth and the nonzero density of the outermost diagonal and band. In addition, `Xmatrix` summarizes trivial and nontrivial transformation totals, and estimates the flop requirements associated with the applied sequence of Givens transformations using the general analysis framework of Section 2.4.1 or 2.4.2.

Additional features of Xmatrix include:

- `Xmatrix` permits modeling of both sparse and dense band transformations. `Xmatrix` assumes a dense band when computing the lengths of dense transformations.
- The user may select automatic *bulge chasing* up or down the band. (See Chapters 3, 4 and 5.)
- The current sparse matrix can be permuted using a user specified permutation vector or by GPS, RCM, GK, MDA, ND or reverse reordering algorithms.
- The current sparsity structure can be saved to a file in Harwell–Boeing, `Xmatrix` or `mtxps` format. Using the `mtxps` filter on the latter type of file creates a postscript image of the saved sparse matrix.
- The "movie" feature permits the recording and playback of a series of Givens transformations.

As shown in this brief introduction, `Xmatrix` is a versatile tool, useful for the design and evaluation of sparse algorithms employing sequences of Givens transformations. `Xmatrix` was used extensively during the development of algorithms presented in Chapters 4 and 5.

2.6 Trisymb

`Trisymb` is a symbolic sparse matrix reduction tool for sparsely (or densely) banded symmetric matrices. Assuming exact cancellation does not occur, `Trisymb` manipulates sparsity structures to model the application of a sequence of Givens similarity transformations to sparsely banded symmetric matrices.

`Trisymb` provides an efficient internal data structure for a banded matrix's sparsity structure and a set of basic routines for its manipulation. For example, `Trisymb` provides routines for applying row and column exchanges or nontrivial transformations to the stored sparse matrix. Using this basic platform, sparse reduction algorithms are easily implemented in `Trisymb`. Currently, Trisymb provides the following five algorithms: sparse R–S, BC, SBC, HYBBC and HYBSBC. (The many unfamiliar terms and acronyms used in this section will be fully explained in later chapters.)

A user must supply `Trisymb` with a sparse symmetric matrix in Harwell–Boeing format and a permutation vector, which typically specifies a bandwidth reducing preordering of the sparse problem. Unlike `Xmatrix`, `Trisymb` is suitable for the symbolic reduction of both small and large symmetric sparse matrices. The availability of resources on a particular machine places the only restrictions on problem size. Once appropriate data structures have been initialized with a sparse problem, `Trisymb` executes the requested symbolic reduction by calling routines to manipulate the stored sparsity structure.

Throughout each reduction `Trisymb` gathers a wide range of statistics, producing a general reduction report and separate reports tailored to highlight special features of individual algorithms. The general report summarizes the extent of the executed reduction, and provides separate transformation and flop counts for each stage of the algorithm. `Trisymb` estimates flop requirements using the general analysis framework of Section 2.4.1 or 2.4.2.

To permit the application of different variants of the five implemented algorithms, a user may select from several command line arguments specifying the following modeling options.

- `Trisymb` can model either standard or fast Givens transformations.
- `Trisymb` permits modeling of both sparse and dense band transformations. When computing the lengths of dense transformations `Trisymb` assumes a dense band, but does not modify the underlying manipulation of sparsity structures.
- For appropriate algorithms, users may request either a partial bandwidth contraction or a complete tridiagonalization.

- `Trisymb` provides fixed, Δ, *nosplit* and density thresholded transition and termination strategies.

`Trisymb` predicts the computational requirements and sparsity characteristics of reductions without the aid of a matrix's nonzero entries. Despite working solely with sparsity structures, `Trisymb`'s no cancellation assumption provides relatively accurate analyses. `Trisymb` is especially useful when comparing a problem's reduction using several different algorithms or when assessing the effectiveness of a single algorithm applied to many different problems. The development and experimental evaluation of algorithms in subsequent chapters uses `Trisymb` extensively.

2.7 A Bipartite Graph Model for Sparse Tridiagonalization

This section introduces a graph-theoretic model for the tridiagonalization or partial reduction of symmetric sparse matrices. We begin with basic graph theory notation for bipartite graphs and establish their connection to symmetric sparse matrices. Using bipartite graphs, we then model the application of orthogonal transformations used in sparse reductions. This section's discussion assumes familiarity with basic graph-theoretic terminology. (A general graph theory reference is [Har69]).

2.7.1 Bipartite Graphs and Sparse Matrices

Bipartite graphs can be used to describe the nonzero structure of sparse matrices. The bipartite graph B_A of matrix A's sparsity structure consists of two sets of vertices or nodes V' and V, and a set of *edges* E. For an $n \times n$ matrix both vertex sets consist of the first n integers, with instances and variables of V' differentiated from those of V using primes. To model a sparse matrix we assume node $i' \in V'$ corresponds to row index i of A, while the members of V correspond to A's column indexes. Each edge in E consists of an ordered pair of vertices $< v', v >$, satisfying $v' \in V'$ and $v \in V$. Sparse matrix A's bipartite graph B_A has an edge $< i', j >$ for each entry $A_{i,j} \neq 0$.

Figure 2.4 illustrates a small symmetric sparse matrix and its corresponding bipartite

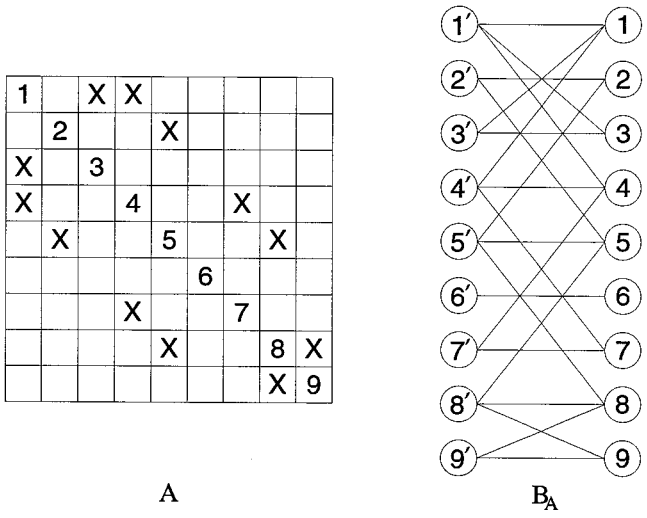


Figure 2.4: A Sparse Matrix and its Bipartite Graph

graph. Assuming the matrix's main diagonal is nonzero, there is a one-to-one correspondence between the edges of B_A and the nonzero entries of A. If the labels of vertices in sets V' and V of B_A are arranged in sequence, and v' and v are horizontally aligned, off-diagonal nonzero entries further from A's main diagonal correspond to steeper edges in the bipartite graph. In graphical terms, the ultimate goal of a tridiagonalization algorithm is to reduce B_A to a bipartite graph whose edges $< i', j >$ $(i' \in V', j \in V)$ satisfy $|i - j| \leq 1$.

One of the most efficient and convenient schemes for mathematically representing a bipartite graph uses *adjacency sets*. Two nodes $x' \in V'$ and $y \in V$ are adjacent if $< x', y > \in E$. The adjacency set of node $x' \in V'$ is defined by

$$Adj_{B_A}(x') = \{z \in V | < x', z > \in E\},$$

and similarly the adjacency set of node $y \in V$ is

$$Adj_{B_A}(y) = \{z' \in V' | < z', y > \in E\}.$$

As examples, the adjacency sets of nodes $8'$ and 4 in Figure 2.4 are $\{5, 8, 9\}$ and $\{1', 4', 7'\}$ respectively. The adjacency sets for all nodes in either V or V' completely defines E. For a symmetric matrix A,

$$\forall v \in V, \quad Adj_{B_A}(v) = (Adj_{B_A}(v'))'.$$

(We interpret the "priming" of a set as applying prime to each of its members.)

2.7.2 Modeling Givens Transformations

Using bipartite graphs we can model the changes to matrix sparsity structures resulting from the application of orthogonal similarity transformations. Suppose the Givens transformation $G(i,j,\theta)^T AG(i,j,\theta)$ eliminates nonzero entries $A_{j,k}$ and $A_{k,j}$ $(i \neq k)$, or equivalently eliminates edges $< j', k >$ and $< k', j >$ from B_A. We will investigate the application of the two halves of this transformation independently.

First, consider the application of rotation $G(i,j,\theta)^T$ to the left of A, eliminating nonzero $A_{j,k}$ or edge $< j', k >$. As discussed in Section 2.3, the precise implementation of the Givens rotation depends on the nonzero status of the *zeroing entry* $A_{i,k}$. When $A_{i,k}$ is nonzero (or $< i', k > \in E$), a nontrivial Givens rotation is applied. Except for the eliminated entry $A_{j,k}$, the sparsity structures of rows i and j both become the union of their sparsity structures just prior to modification. The corresponding modification of B_A adds the minimal set of edges to E that make $Adj_{B_A}(i') = Adj_{B_A}(j')$, and then removes edge $< j', k >$ from E. Figure 2.5 more formally outlines the modifications to B_A's adjacency sets resulting from $G(i,j,\theta)^T$ application.

1. (a) $U := Adj_{B_A}(i') \cup Adj_{B_A}(j')$
   (b) $Adj_{B_A}(i') := U$
   (c) $Adj_{B_A}(j') := U$
2. FOR EACH $z \in U$
   (a) $Adj_{B_A}(z) := Adj_{B_A}(z) \cup \{i', j'\}$
3. (a) $Adj_{B_A}(j') := Adj_{B_A}(j') - \{k\}$
   (b) $Adj_{B_A}(k) := Adj_{B_A}(k) - \{j'\}$

Figure 2.5: Updating B_A to Reflect $G(i,j,\theta)^T A$.

As an example, Figure 2.6 illustrates the effects of eliminating entry $A_{4,1}$ (or edge

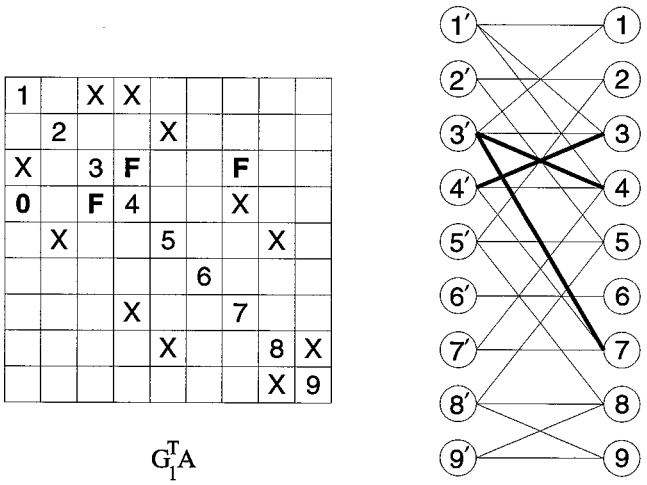


Figure 2.6: $A_{4,1}$ is Eliminated by $G(3,4,\theta_1)^T$.

$< 4', 1 >$) from Figure 2.4's example, using the adjacent rotation $G(3,4,\theta_1)^T$ (or G_1^T) applied to the left of A. The zeroing entry is $A_{3,1}$. Each fill entry is marked by an F and each fill edge in B_A is highlighted.

To complete the similarity transformation, $G(i,j,\theta_1)$ is applied to the right of A, eliminating nonzero $A_{k,j}$ and unioning the sparsity structures of columns i and j. The corresponding modification of B_A adds the minimal set of edges to E that make $Adj_{B_A}(i) = Adj_{B_A}(j)$, and then removes edge $< k', j >$ from E. Exchanging V' and V variables for their V and V' counterparts in Figure 2.5 creates a more formal specification of $B'_A s$ modifications. Figure 2.7 illustrates the effects of completing the Givens similarity transformation begun in Figure 2.6, by eliminating entry $A_{1,4}$ and edge $< 1', 4 >$ with adjacent rotation $G(3,4,\theta_1)$. In this case the zeroing entry is $A_{1,3}$.

Once again, suppose a Givens transformation $G(i,j,\theta)^T A G(i,j,\theta)$ is constructed to eliminate nonzero entries $A_{j,k}$ and $A_{k,j}$ $(i \neq k)$ from the symmetric sparse matrix A, but now assume $A_{i,k}=A_{k,i}=0$. As discussed in Section 2.3, in the special case that the *zeroing entries* $A_{i,k}$ and $A_{k,i}$ are zero, or edges $< i', k >, < k', i > \notin E$, the transformation simply swaps rows and columns i and j. In graphical terms, exchanging a pair of rows or columns corresponds to relabeling a pair of nodes in V' or V. In this case nodes i' and j' switch

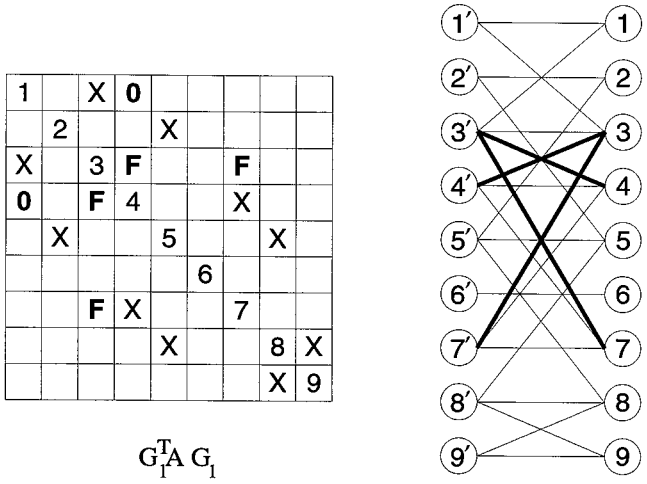


Figure 2.7: Completing the Nontrivial Transformation $G(3,4,\theta_1)^T AG(3,4,\theta_1)$.

labels, as do nodes i and j. More formally, the row and column swaps precipitated by the transformation require modifications to B_A's adjacency sets as outlined in Figure 2.8.

Continuing our example of Figures 2.4, 2.6 and 2.7, Figure 2.9 completes the reduction of column 1 by eliminating nonzero entries $A_{3,1}$ and $A_{1,3}$, using an adjacent transformation swapping rows and columns 2 and 3. In terms of the bipartite model, we simply exchange the labels of nodes $3'$ and $4'$, and nodes 3 and 4. Finally, we envisage moving the relabeled nodes, with their edges attached, to reestablish sequential order of V' and V. The affected nonzero entries and edges are highlighted in the figure.

2.7.3 Modeling Householder Transformations

As mentioned in Section 2.3, another commonly used orthogonal transformation is a Householder transformation. Unlike a Givens rotation, a Householder transformation can simultaneously eliminate multiple nonzero entries from a single row or column. Suppose the Householder transformation H is applied to the left of A as the first half of a similarity transformation. Further assume it modifies rows $i, i+1, \ldots, i+l$, eliminating entries $A_{i+1,k}, A_{i+2,k}, \ldots, A_{i+l,k}$. Except for these entries, the sparsity structure of each modified row becomes the union of the sparsity structures of all modified rows just prior to the

1. FOR EACH $z \in Adj_{B_A}(i') - Adj_{B_A}(j')$
   (a) $Adj_{B_A}(z) := (Adj_{B_A}(z) - \{i'\}) \cup \{j'\}$
   (b) $Adj_{B_A}(z') := (Adj_{B_A}(z') - \{i\}) \cup \{j\}$
2. FOR EACH $z \in Adj_{B_A}(j') - Adj_{B_A}(i')$
   (a) $Adj_{B_A}(z) := (Adj_{B_A}(z) - \{j'\}) \cup \{i'\}$
   (b) $Adj_{B_A}(z') := (Adj_{B_A}(z') - \{j\}) \cup \{i\}$
3. /* Switch adjacency sets.*/
   (a) $S := Adj_{B_A}(i')$
   (b) $Adj_{B_A}(i') := Adj_{B_A}(j')$
   (c) $Adj_{B_A}(j') := S$
   (e) $S := Adj_{B_A}(i)$
   (f) $Adj_{B_A}(i) := Adj_{B_A}(j)$
   (g) $Adj_{B_A}(j) := S$

Figure 2.8: Updating B_A to Reflect a Trivial Transformation Exchanging Rows and Columns i and j.

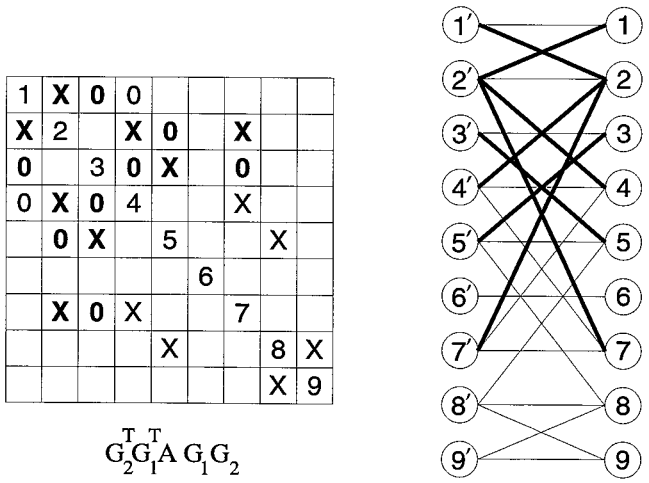


Figure 2.9: Completing the Reduction of Column and Row 1 to Tridiagonal Form with a Row and Column Exchange.

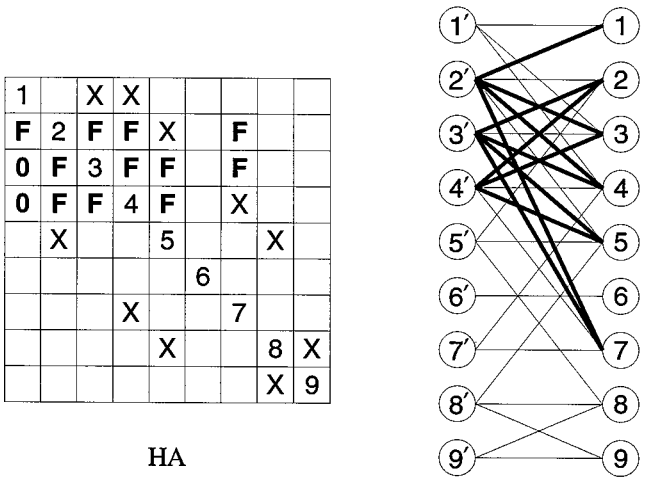


Figure 2.10: Reducing Column 1 to Tridiagonal Form With a Householder Transformation.

eliminations. The corresponding modification of B_A adds the minimal set of edges to E so that

$$Adj_{B_A}(i') = Adj_{B_A}((i+1)') = \ldots = Adj_{B_A}((i+l)'),$$

and removes edges $< (i+1)', k >, \ldots, < (i+l)', k >$ from E. Completing the similarity transformation by applying H to the right of A, results in similar sets of nodes being added and removed from the adjacency sets of nodes $i, i+1, \ldots, i+l$.

In two stages Figures 2.10 and 2.11 illustrate the application of a Householder similarity transformation to the sparse example in Figure 2.4. The first figure illustrates the effect of applying a Householder transformation to the left of the sparse matrix to eliminate entries $A_{4,1}$ and $A_{3,1}$ or edges $< 4', 1 >$ and $< 3', 1 >$. The second figure illustrates the modifications made to both the matrix and its associated bipartite graph by the complete similarity transformation. The application of this transformation and the application of the two Givens transformations leading to Figure 2.9 both result in the elimination of two nonzeros from both the first row and column. We note that the use of Givens transformations has resulted in 12 fewer fill entries than for the Householder transformation.

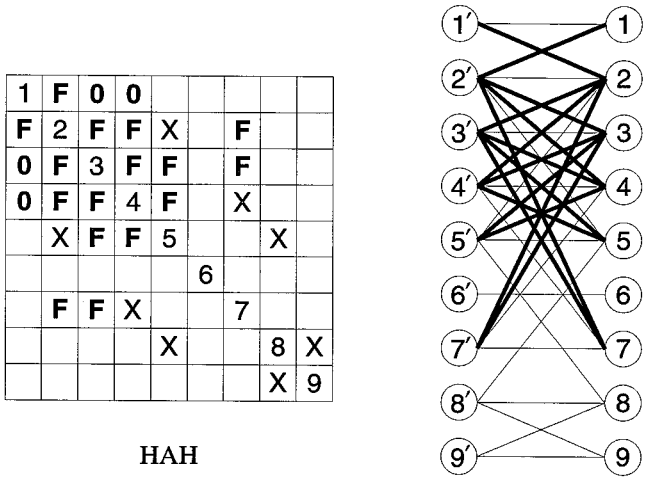


Figure 2.11: Completing the Reduction of Column and Row 1 with a Householder Similarity Transformation.

2.7.4 Concluding Remarks

The bipartite graph model presented in this section is intended for describing, understanding and providing insight into the sparse reduction algorithms described in subsequent chapters. Often symmetric sparse matrices are modeled using undirected graphs with a single node for each entry of the main diagonal and a single edge for each symmetric pair of off-diagonal nonzero entries. Using undirected graphs, however, it is more difficult to model the nonzero status of main diagonal entries and the fill they may create. Undirected graphs could be augmented with self-loop edges. Alternatively, bipartite graphs represent a diagonal entry as an edge between two distinct nodes, providing a more easily understood and natural modeling of diagonal entries. In addition, bipartite graphs can separately model the effects of applying each half of an orthogonal similarity transformation. The bipartite graph model is also easily extended to unsymmetric matrices or non-congruent transformations. Finally, bipartite graph models are conducive to describing sparse reduction algorithms using sequences of orthogonal similarity transformations. For example, the unioning of row and column sparsity structures corresponds to the unioning of adjacency sets. As we will see in the following chapter, the bipartite graph model provides a natural graphical interpretation

of band-preserving reduction algorithms artifacts such as *bulge chasing sequences.*

Chapter 3

Existing Algorithms and Their Extension for Sparse Tridiagonalization

This chapter explores existing direct tridiagonalization approaches, analyzing their limitations and their potential extension to improve sparsity exploitation. For evaluating the relative success of presented algorithms, we first introduce two simple sparse model problems that lend themselves to formal analysis. Employing these model problems, as well as other sparse matrices, we then characterize the fill properties of Givens Reduction, identifying the inefficiencies associated with this direct dense matrix approach. If we attempt to improve sparsity exploitation by changing the elimination order of each column and the planes of zeroing rotations, the next section shows even these customized Givens Reductions experience overwhelming levels of fill. In the following section we explore alternative band-preserving tridiagonalization approaches for banded matrices, which restrict the accumulation of fill entries to improve reduction efficiency. The chapter's final section considers the extension of these algorithms for the tridiagonalization of general sparse symmetric matrices, and demonstrates the relative efficiency of the *sparse* Rutishauser-Schwarz algorithm.

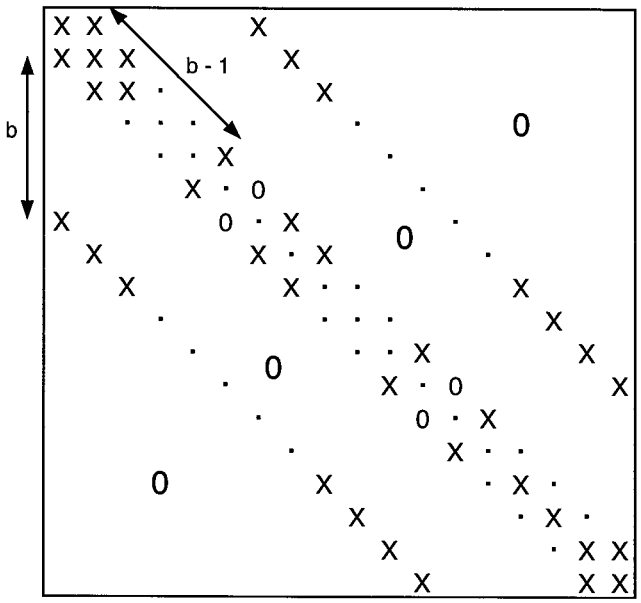


Figure 3.1: The 5-Point Model Problem

3.1 Sparse Model Problems

In practice the density and nonzero structure of sparse symmetric eigenvalue problems vary widely. To provide a common basis for the formal comparison of reduction algorithms, this section presents two families of sparse model problems that lend themselves to formal analysis.

The first family of model problems consists of densely banded symmetric matrices of bandwidth b and order n. We define the bandwidth b (or semi-bandwidth) of matrix A as $\max_{i,j \in \{1 \ldots n\}, i \neq j} |i - j|$ such that $A_{i,j} \neq 0$.

The second family of model problems is the 5-point problems. The symmetric sparse matrices associated with 5-point problems arise from discretizing partial differential equations on a square uniform grid of dimension $b \times b$, using Dirichlet boundary conditions and a 5-point differencing molecule. The resulting matrices are of order $n = b^2$ and using a standard lexicographic ordering, consist of five nonzero diagonals, as illustrated in Figure 3.1.

These families of model problems were selected because they are scalable and exhibit important features also found in many practical sparse symmetric problems. In addition, the simple and highly regular nature of their sparsity structures makes formal algorithm analysis feasible. Ultimately, however, we are interested in the tridiagonalization of general sparse symmetric matrices. Unfortunately, our sparse model problems do not show all the significant characteristics exhibited by sparse matrices. For example, many sparse matrices are much more irregular than our model problems. Consequently, we often augment formal analysis results for our model problems with experimentation using sparse symmetric problems from Section 2.2's test suite.

3.2 A Naive Approach to Sparse Tridiagonalization

One approach to sparse tridiagonalization is to simply enhance a successful dense matrix algorithm, such as Givens reduction, with basic sparse matrix operations. Using $O(n^3)$ flops and $O(n^2)$ storage, the standard Givens reduction tridiagonalizes a dense symmetric matrix column by column, as shown in Figure 3.2. Within each column nonzero entries are eliminated from the bottom up using the column's subdiagonal entry as the zeroing

```
FOR col:= 1 TO n-2 DO
    FOR row:= n DOWNTO col+2 DO
        A := G(col + 1, row, θ)^T A G(col + 1, row, θ)
```

Figure 3.2: Givens Reduction

entry. We suggest two modifications to this basic algorithm in an attempt to exploit matrix sparsity. First, we can obviously avoid constructing and applying transformations to eliminate entries that are already zero. In addition, we need to apply a transformation to only those lower triangular entries in the unioned sparsity structure of the two modified rows and columns.

To evaluate the potential of this modified form of Givens reduction, we first consider its

application to densely banded model problems. Unfortunately, for nontrivial bandwidths Givens reduction quickly overwhelms the banded nature of these sparse matrices with fill entries. In fact, after the reduction of the first

$$r = \left\lceil \frac{n-1}{b} - 1 \right\rceil = \frac{n-1}{b} - \frac{Mod(n-1,b)^*}{b}$$

columns the remaining $(n-r)\times(n-r)$ submatrix is completely filled in. Table 3.1 provides estimates of the computational requirements for this densely banded reduction computed using the analysis framework of Section 2.4. The analysis assumes the reduction employs standard Givens transformations, and $F^{\mathrm{G}}_{\mathrm{DBGR}}$ and T_{DBGR} refer to flop and transformation counts respectively. In addition to $F^{\mathrm{G}}_{\mathrm{DBGR}}$, the construction of each transformation requires

$F^{\mathrm{G}}_{\mathrm{DBGR}}$	T_{DBGR}
$(2-\frac{4}{b}+\frac{2}{b^2})n^3+O(n^2)$	$(1-\frac{1}{b})(\frac{n^2}{2}-n+\frac{1}{2})$ $-(\frac{Mod(n-1,b)}{2})(1-\frac{Mod(n-1,b)}{b})$

Table 3.1: Tridiagonalization Costs of Givens Reduction for a Densely Banded Matrix

one square root. For comparison, Table 3.2 provides formula for the tridiagonalization costs of reducing a dense symmetric matrix with Givens reduction. $F^{\mathrm{G}}_{\mathrm{DBGR}}$ and T_{DBGR} are

$F^{\mathrm{G}}_{\mathrm{GR}}$	T_{GR}
$2n^3+n^2-17n+14$	$\frac{n^2}{2}-\frac{3n}{2}+1$

Table 3.2: Tridiagonalization Costs of Givens Reduction for a Dense Matrix

very close to their dense matrix counterparts and the densely banded reduction takes little advantage of matrix sparsity. In addition, due to overwhelming levels of fill, the sparse reduction also requires $O(n^2)$ storage.

The overwhelming levels of fill produced by Givens reduction are not restricted to densely banded model problems. Despite the additional sparsity of the 5-pt problems, their tridiagonalization by Givens reduction also suffers from high levels of fill. After

*$Mod(x,y)$ is the remainder from the division of integer x by integer y.

Problem	n	Initial Off-Diagonal Nonzero Density	# of columns (rows) eliminated before remaining matrix is dense.
Dense Band	n	$\frac{b(2n-b-1)}{n^2-n}$	$\frac{n-1}{b} - \frac{Mod(n-1,b)}{b}$
5-Pt Problems	b^2	$\sim 4/b^2$	$2b-3$
PLAT1919	1919	0.83%	36
NOS3	960	1.62%	23
BCSSTK08	1074	1.03%	7

Table 3.3: Fill Characteristics of Givens Reduction

$r = 2b - 3$ columns of a 5-pt problem have been reduced to tridiagonal form the remaining $(n-r) \times (n-r)$ submatrix is dense. As for the densely banded model problems, the tridiagonalization of this remaining submatrix dominates reduction costs and sparsity exploitation is limited.

The speed with which Givens reduction fills our model problems is typical of most sparse symmetric problems. Table 3.3 summarizes the fill characteristics for both families of model problems and also provides data for three additional sparse symmetric problems from Section 2.2's test suite. To determine the fill characteristics of each Harwell–Boeing problem, we used the basic sparse reduction platform of `Trisymb` to implement the sparse Givens reduction algorithm. The resulting symbolic reduction code was applied to each sparse problem. In each case, Givens reduction produces overwhelming numbers of new nonzeros, quickly filling the unreduced portion of each matrix despite extremely low initial nonzero densities.

This section's results clearly demonstrate that a naive approach to tridiagonalization using Givens reduction is unable to significantly exploit matrix sparsity. The fill created by these reductions so quickly overwhelms matrix sparsity that the original matrix might as well have been treated as dense. There is no evidence to suggest that other dense matrix tridiagonalization algorithms, for example Householder's reduction, offer any significant advantage. In fact, as shown in Section 2.3, Householder transformations generally create higher levels of fill.

3.3 Customized Sparse Givens Reductions

The unsuccessful sparse tridiagonalization algorithm of Section 3.2 did not change the basic form of Givens reduction. It eliminates each column's nonzeros in the same order as the standard Givens reduction, using the same subdiagonal zeroing entry. The sparse algorithm places complete reliance on exploiting matrix sparsity at the transformation level, without regard for the position and number of fill entries produced. Moreover, no attempt is made to utilize the fine grained control over the elimination of nonzeros, and the introduction of fill entries, provided by Givens transformations.

The flexibility of Givens rotations permits both the order in which a column is zeroed, and the plane of each zeroing rotation, to be modified. (See Section 2.3.) These variable elements of the basic algorithm can be used to construct customized sparse Givens reduction algorithms which attempt to further improve sparsity exploitation. Unfortunately, the experimentation of Duff and Reid [DR75] shows that, independent of rotation plane selection and the elimination order of each column's nonzeros, adaptations of Givens column by column reduction for large sparse matrices generally experience overwhelming levels of fill, and matrix sparsity is quickly destroyed. In fact Duff and Reid conclude that if a Givens reduction approach is taken, typically little advantage is made of sparsity and it is preferable to treat the matrix as dense.

3.4 The Tridiagonalization of Banded Matrices

If tridiagonalization algorithms are to effectively utilize matrix sparsity it appears essential to restrict the accumulation of fill entries to some maintainable substructure of the matrix. Suppose that a symmetric matrix A of bandwidth b, is to be reduced to tridiagonal form. In addition, for the remainder of this subsection assume that the band is dense. We have shown that applying a column by column Givens reduction to this model problem leads to overwhelming levels of fill outside the band, quickly destroying matrix sparsity. Alternatively, the algorithm of Rutishauser [Rut63] and Schwarz [Sch71] (subsequently referred to

as the Rutishauser-Schwarz, or simply R-S, algorithm) controls the encumbering effects of fill by actively preserving a matrix's banded structure throughout tridiagonalization. After exploring this algorithm in detail, we briefly describe a closely related algorithm due to Lang [Lan92]. This algorithm is primarily intended for parallel implementation and sequentially is less efficient than the Rutishauser-Schwarz approach.

3.4.1 The Rutishauser-Schwarz Algorithm

The pseudocode in Figure 3.3 outlines the Rutishauser-Schwarz algorithm. Once again, Givens transformations are used by this band-preserving tridiagonalization algorithm to provide fine grained control over the elimination process. Globally the effect is that the

```
FOR col:= 1 TO n-2 DO
    FOR diag:= min(b,n-col) DOWNTO 2 DO
        /*Zero A_{col+diag,col}.*/
        A := G(col + diag, col + diag - 1, θ)^T  A  G(col + diag, col + diag - 1, θ)
        IF bandwidth(A) > b THEN
            Annihilate bulges with additional adjacent Givens transformations.
```

Figure 3.3: The Rutishauser-Schwarz Algorithm

banded matrix is reduced column by column. Indeed, within each column adjacent transformations eliminate the nonzeros from the outside in. The key difference from Givens reduction is that R-S immediately removes fill created outside the band of the original matrix. The symmetric elimination of a band nonzero produces a pair of fill entries, or *bulges*, outside the band as illustrated by the `B1` entries in Figure 3.4. Before eliminating the next band nonzero R-S chases the bulges off the end of the matrix with an additional sequence of adjacent transformations. (See Figure 3.4.) In this fashion the algorithm maintains the banded structure of the unreduced portion. During the reduction of a typical column k, the elimination of band entry $A_{i,k}$ requires $\lceil \frac{n-b-i+1}{b} \rceil$ bulge chasing transformations. Adjacent transformations are used exclusively by the algorithm because nonadjacent rotations create triangular bulges, consisting of several nonzero entries, which require a larger number of

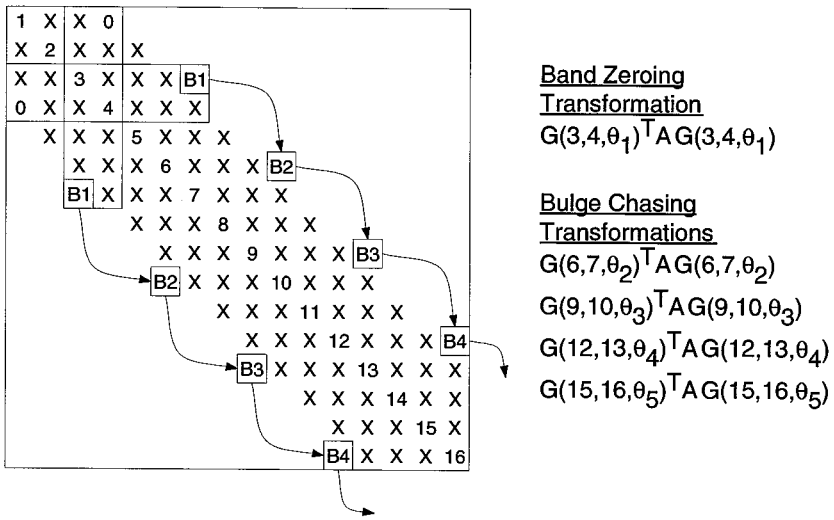


Figure 3.4: Bulge Chasing to Preserve Bandwidth

bulge chasing transformations.

The graph-theoretic model of Section 2.7 provides an alternative view of the bulge chasing process. In terms of our bipartite graph model, the steepest edge in the graph indicates the bandwidth of the corresponding sparse matrix.

$$b = \max_{<i',j>\in E} |i - j|$$

Consequently, if a reduction algorithm creates a bulge lying outside the original matrix's band, the corresponding *bulge edge* $< x', y >$ satisfies $|x - y| > b$.

As a working example, B_{A_1} of Figure 3.5 is the bipartite graph corresponding to the densely banded example of Figure 3.4. Without impacting subsequent discussion, we omit many edges corresponding to interior band nonzeros to avoid cluttering the illustrations. B_{A_1} highlights two paths of edges connecting nodes $(4', 7, 10', 13, 16')$ and $(4, 7', 10, 13', 16)$. Each pair of neighboring nodes in these lists are connected by an edge $< i', j >$ satisfying $b = |i - j|$. We refer to these edge sequences as *bulge chasing paths* for edges $< 4', 1 >$ and $< 1', 4 >$. The constituent edges of each path are ancestors of the bulge edges created by bulge chasing sequences accompanying R-S's elimination of nonzeros $A_{4,1}$ and $A_{1,4}$.

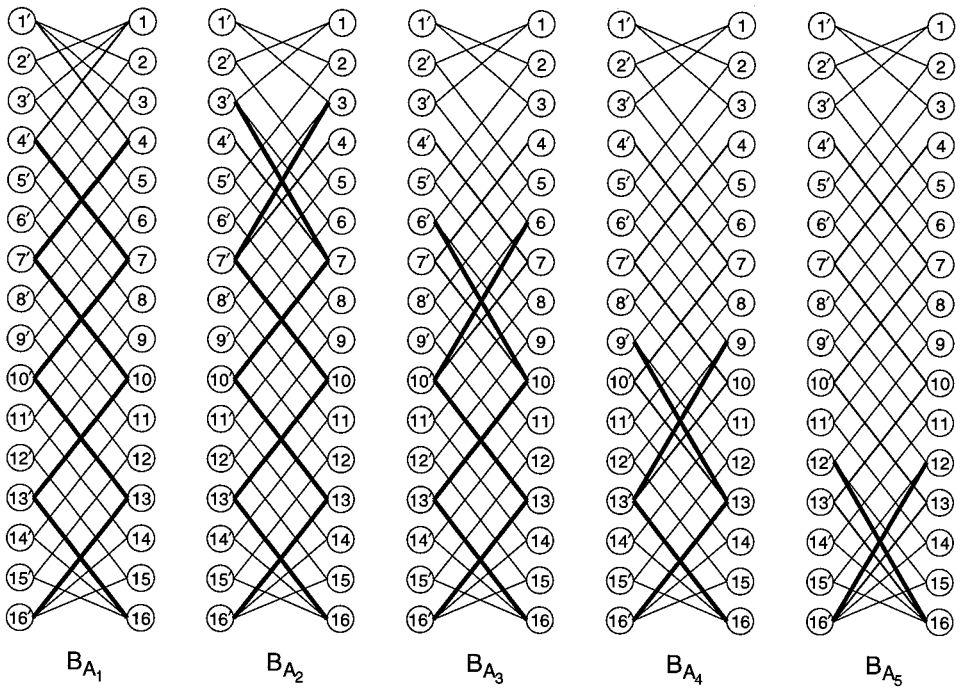


Figure 3.5: The Bulge Chasing Path

Prior to the elimination of a band nonzero, identifying its associated bulge chasing path in the bipartite graph is straightforward. More generally, suppose R-S will eliminate edge $< i', j >$ using an adjacent rotation that unions adjacency sets $Adj_{B_{A_1}}(i')$ and $Adj_{B_{A_1}}((i-1)')$. The first edge in the associated bulge chasing path connects node i' with node $k = (i+b) \in V$. Similarly, the next edge in the bulge chasing path is $< l', k > \in E$, where $l = k + b$. The construction of the bulge chasing path continues in this fashion until the an edge in this sequence is absent from E. For a densely banded reduction, the node, x or x', terminating each bulge chasing path satisfies $x + b > n$.

Continuing our example in Figure 3.5, consider the effects of R-S's elimination of edges $< 4', 1 >$ and $< 1', 4 >$ from B_{A_1}. Using the modeling techniques of Section 2.7.2, the elimination of these edges unions the adjacency sets of nodes $4'$ and $3'$, and 4 and 3, creating the two bulge edges $< 3', 7 >$ and $< 7', 3 >$ shown in bipartite graph B_{A_2}. To preserve the bandwidth an adjacent transformation eliminates these bulge edges, creating two new bulge edges $< 6', 10 >$ and $< 10', 6 >$ in B_{A_3}. The bulge chasing process continues until the

last bulge edges are eliminated from B_{A_5}. The net effect of the bulge chasing sequence is to duplicate all edges in the original bulge chasing paths and drag the endpoints of a copy of each edge up one node in the graph. We duplicate bulge chasing path edges, because R-S uses nontrivial transformations for densely banded reductions.

Table 3.4 provides formula for the tridiagonalization costs of the Rutishauser-Schwarz algorithm, computed using the analysis framework of Section 2.4. A comprehensive description of this analysis is presented in [Cav93]. The analysis assumes $2 < b \leq (n/2 - 1)$ and C_{R-S} is the nonanalytic term $Mod(n-1,b)(Mod(n-1,b)-b)$, which typically can be safely ignored without incurring large errors. $F^{\mathrm{G}}_{\text{R-S}}$ and $T_{\text{R-S}}$ refer to flop and transformation counts respectively. In addition to $F^{\mathrm{G}}_{\text{R-S}}$, the construction of each transformation requires

$F^{\mathrm{G}}_{\text{R-S}}$	$T_{\text{R-S}}$
$(6b - \frac{8}{b} + 2)n^2 - (6b^2 + 5b - \frac{16}{b} + 5)n$ $+2b^3 + 4b^2 - b - \frac{8}{b} + 3 + (\frac{8}{b} + 3)C_{R-S}$	$\frac{(b-1)(n-1)^2 + C_{R-S}}{2b}$

Table 3.4: Tridiagonalization Costs of the Rutishauser-Schwarz Algorithm for a Densely Banded Matrix

one square root. Although $T_{\text{R-S}}$ is comparable to the number of transformations used by Givens reduction on the same densely banded problem, in general the band-preserving approach modifies fewer nonzero entries with each transformation. As a result, for matrices of moderate bandwidth $F^{\mathrm{G}}_{\text{R-S}}$ is smaller than the floating point requirements of Givens reduction given by $F^{\mathrm{G}}_{\text{DBGR}}$.

EISPACK's [GBDM77] implementation of the R-S algorithm, BANDR, does not use standard Givens transformations. Instead it employs a "root free" variant of the transformation, with identical fill properties, often referred to as a *fast Givens transformation* [Gen73]. (See Sections 2.3 and 2.4.2.) In this case the computational requirements of the band-preserving tridiagonalization are reduced to

$$\begin{aligned} F^{\mathrm{FG}}_{\text{R-S}} &= (4b - \frac{10}{b} + 6)n^2 - (4b^2 + 3b - \frac{20}{b} + 10)n \\ &\quad + \frac{4b^3}{3} + \frac{5b^2}{2} - \frac{5b}{6} - \frac{10}{b} + 7 + (\frac{10}{b} + 2)C_{R-S} \end{aligned} \tag{3.1}$$

and n square roots. The analysis leading to $F_{\text{R-S}}^{\text{FG}}$ ignores the cost of periodic rescaling and assumes that a reduction uses the two types of fast Givens transformations in equal proportion. For either variant of the band-preserving R-S algorithm $O(bn)$ storage is required.

We note that the Rutishauser-Schwarz algorithm will not be faster for all bandwidths than the best dense matrix algorithm, Householder's reduction. Assuming that the Householder reduction requires $\frac{4}{3}n^3$ flops [GV89], while R-S requires $4bn^2$, the dense matrix algorithm requires fewer floating point operations for bandwidths larger than $\frac{n}{3}$. The selection of a tridiagonalization algorithm, however, is often strongly influenced by the lower storage requirements of the Rutishauser-Schwarz algorithm.

3.4.2 Lang's Algorithm

Lang's column-oriented tridiagonalization algorithm [Lan92] for symmetric banded matrices is closely related to the Rutishauser-Schwarz approach. Like the R-S algorithm, Lang's algorithm restricts the accumulation of fill to a maintainable substructure of the original matrix using bulge chasing. Unlike R-S, however, Lang's algorithm uses Householder similarity transformations to simultaneously eliminate multiple nonzero entries. Implementations of Lang's algorithm exploit the symmetry of sparse problems and similarity transformations, only considering transformation modifications to the lower triangular portion of each matrix.

Lang's algorithm begins by reducing the first column of our densely banded model problem to tridiagonal form, using a single Householder similarity transformation. This elimination creates a pair of $(b-1)\times(b-1)$ triangular bulges, as shown for the small example in Figure 3.6. Rutishauser [Rut63] points out that the cost of chasing the entire triangular bulge off the end of the matrix is unacceptably high. Alternatively, Lang eliminates only the first column of the lower triangular bulge, using an additional Householder similarity transformation. As shown by Figure 3.7, this elimination creates a second set of triangular bulges. Lang's algorithm continues the reduction by eliminating the first column of the second bulge, creating a third pair of bulges if n/b is sufficiently large. When the first column

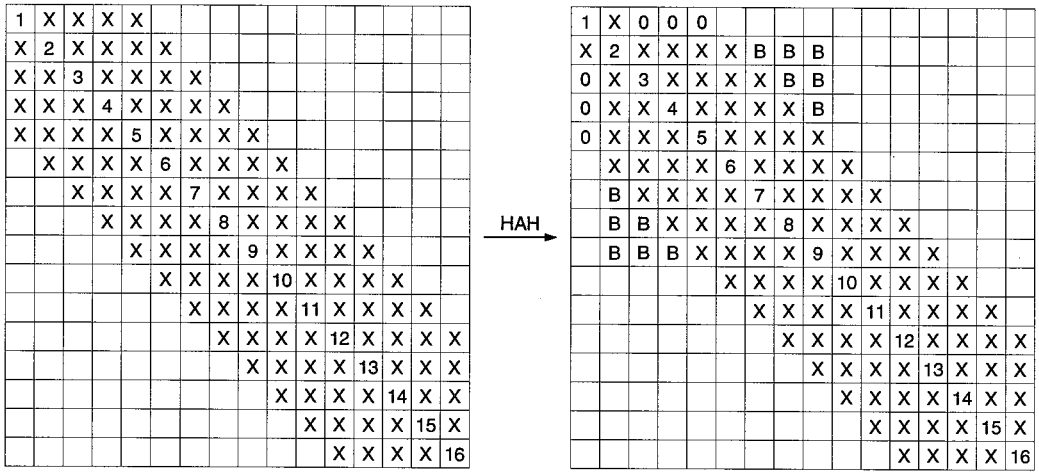


Figure 3.6: Lang's Algorithm Produces Triangular Bulges

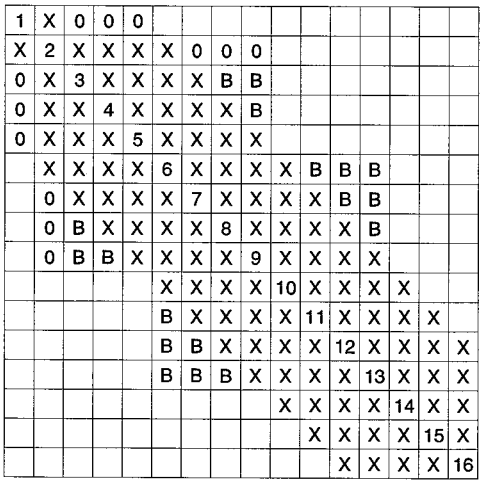


Figure 3.7: Chasing One Column of a Triangular Bulge

of each triangular bulge has been eliminated, Lang's algorithm reduces the band's next nonzero column to tridiagonal form and chases the first columns of its associated triangular bulges. Continuing in this fashion the algorithm completes a matrix's tridiagonalization. Lang's modified bulge chasing procedures do not reestablish the matrix's original banded form between the elimination of successive columns of the band. Chasing the first column of each triangular bulge, however, prevents bulges from growing beyond their original $(b-1) \times (b-1)$ size and gradually marches them off the end of the matrix.

Lang's algorithm is primarily intended for parallel implementation. Its use of Householder transformations allows almost all computation to be performed using level 2 BLAS [DDHD90, DDHH88, LHKK79]. As a result, Lang's algorithm is more conducive to parallel implementation than R-S, which uses level 1 BLAS type operations exclusively. Unfortunately, Lang's algorithm requires almost twice the storage of R-S to handle its sequence of large triangular bulges. In addition, Lang reports that the computational requirements of his algorithm are approximately $6bn^2$ flops. Although these requirements are close to $F^{\mathrm{G}}_{\text{R-S}}$, Lang's algorithm requires 50% more flops than the fast Givens variant of R-S. As a result, theoretically the Rutishauser-Schwarz tridiagonalization is sequentially more efficient than Lang's algorithm. Despite increased flop requirements, Lang reports sequential experiments in which his algorithm runs faster than EISPACK's BANDR on a single node of an iPSC/860 hypercube parallel computer. It appears this discrepancy is a machine dependent anomaly in which the cache of an iPSC node is able to exploit the improved data locality of level 2 BLAS. In addition, Lang's implementation is based on optimized assembly-coded BLAS routines.

More recently, work by Bischof and Sun [BS92] generalizes Lang's algorithm, developing a novel framework for the tridiagonalization of symmetric banded matrices. Bischof and Sun's approach reduces the band of the matrix to tridiagonal form in one or more bandwidth reducing stages. Each stage of their algorithm eliminates the outer d $(1 \leq d < b)$ subdiagonals of the current band using a Lang type algorithm. The only difference between a partial reduction and Lang's algorithm is the number of nonzeros eliminated from each column of the band. By cleverly selecting sequences of partial band reductions, Bischof

and Sun are able to reduce tridiagonalization storage requirements to R-S levels or improve algorithm complexity by 10–25% relative to Lang. Despite these improvements, the complexity of the R-S algorithm remains minimal and of these algorithms R-S is the best generally applicable sequential densely banded tridiagonalization algorithm.

3.5 Generalization of Band-Preserving Tridiagonalization Techniques

The discussion of Section 3.4 explored the tridiagonalization of densely banded matrices. For general sparse symmetric matrices we can extend the band-preserving techniques of Rutishauser and Schwarz to form the following two stage sparse tridiagonalization algorithm.

1. $A := P^T AP$, where P is a bandwidth reducing permutation matrix.
2. Tridiagonalize A using an enhanced form of the R-S algorithm.

We begin this section with a short introduction to bandwidth reducing reorderings and successful heuristics for their computation. The following subsection describes an enhanced variant of the Rutishauser-Schwarz algorithm and explores the reduction characteristics of the two stage sparse tridiagonalization algorithm. Finally, we briefly explore a similar two stage sparse reduction based on Lang type algorithms.

3.5.1 Bandwidth Reducing Sparse Matrix Preorderings

To reduce the bandwidth of symmetric sparse matrices we use orthogonal similarity transformations of the form $P^T AP$, where P is a carefully selected permutation matrix. The problem of minimizing the bandwidth of a matrix by permuting rows and columns is NP-hard [Pap76] and is known to remain so even for symmetric sparse matrices whose associated undirected graph is a tree with all vertices of degree ≤ 3 [GGJK78]. Many heuristic algorithms have been suggested for the identification of bandwidth reducing preorderings [Cut72, CCDG82]. Two of the most widely accepted algorithms are the *reverse*

	GPS Bandwidth Smaller	RCM Bandwidth Smaller	GPS and RCM Bandwidths Equal
# of Problems	91	11	13

Table 3.5: Comparing GPS and RCM Preorderings for the Harwell–Boeing Test Suite

Cuthill-McKee (or RCM) [GL78b] and *Gibbs-Poole-Stockmeyer* (or GPS) [GPS76a, Lew82] algorithms. The experience of Gibbs et al [GPS76b] suggests that GPS most frequently provides the smallest bandwidth preorderings over a wide range of problems. Our experimentation with Section 2.2's test suite of 115 Harwell–Boeing test problems, summarized in Table 3.5, supports this conclusion. Consequently, for experimentation we typically select GPS and often reference it as the *bandwidth reducing preordering* in subsequent discussion.

3.5.2 The Sparse Rutishauser-Schwarz Algorithm

The two stage tridiagonalization algorithm takes limited advantage of matrix sparsity if R-S treats the band of the preordered matrix as dense. In this case complete reliance is placed upon the preordering algorithm to exploit sparsity, but for many problems the band of the preordered matrix is relatively sparse prior to reduction. (See Table 3.6.) To take advantage of band zeros, three modifications could be made to the basic band-preserving tridiagonalization algorithm.

1. Avoid constructing and applying transformations to eliminate band or bulge entries that are already zero.

2. Exploit *zeroing entries* (see Section 2.3) that are zero by performing row and column exchanges instead of using the general form of the Givens transformation.

3. Apply each nontrivial transformation to only those lower triangular entries whose column(row) index is in the unioned sparsity structure of the two modified rows(columns).

Both Schwarz's code [Sch71] and EISPACK's BANDR [GBDM77] check if the bulge or band entry is already zero before performing an elimination. These codes, however, are primarily intended for densely banded matrices and neither incorporates the second or third modification. Enhancing the two stage tridiagonalization algorithm by all three sparsity modifications produces a new algorithm subsequently referred to as the sparse Rutishauser-Schwarz algorithm or simply sparse R-S.

When the band of the permuted matrix is sparse, sparse R-S enjoys an additional computational advantage. As shown in Section 3.4.1, we can identify the bulge chasing path associated with a particular band nonzero $A_{i,k}$ prior to its elimination. $A_{i,k}$'s bulge chasing path consists of edges in the sequence

$$< i', (i+b) >, < (i+2b)', (i+b) >, < (i+2b)', (i+3b) >, \ldots$$

and terminates when the next edge in the series is absent from E. The Rutishauser-Schwarz algorithm uses one bulge chasing transformation for each edge in the path during $A_{i,k}$'s elimination.

For a densely banded reduction a bulge chasing path always terminates at a node, x or x', satisfying $x + b > n$. When the band is sparse, however, the bulge chasing path may terminate before this condition is satisfied, reducing bulge chasing requirements. For example, consider R-S's elimination of $A_{4,1}$ from the small sparsely banded matrix ($b = 3$) in Figure 3.8. The bulge chasing path in the associated bipartite graph starts with edge $< 4', 7 >$. Because $< 10', 7 > \notin E$, the bulge chasing path terminates at node 7 and $A_{4,1}$'s elimination requires a single bulge chasing transformation. In contrast, the elimination of the same entry from a densely banded matrix of equal dimension and bandwidth requires four bulge chasing transformations.

Although the R-S algorithm removes fill created outside the band with bulge chasing transformations, it allows fill entries within the band to accumulate. Unfortunately, the unreduced portion of a typical sparse matrix's band fills quickly during band-preserving tridiagonalization and there is little opportunity for enhancements exploiting band sparsity. As an example, consider applying sparse R-S to the 5-point model problems of Section 3.1.

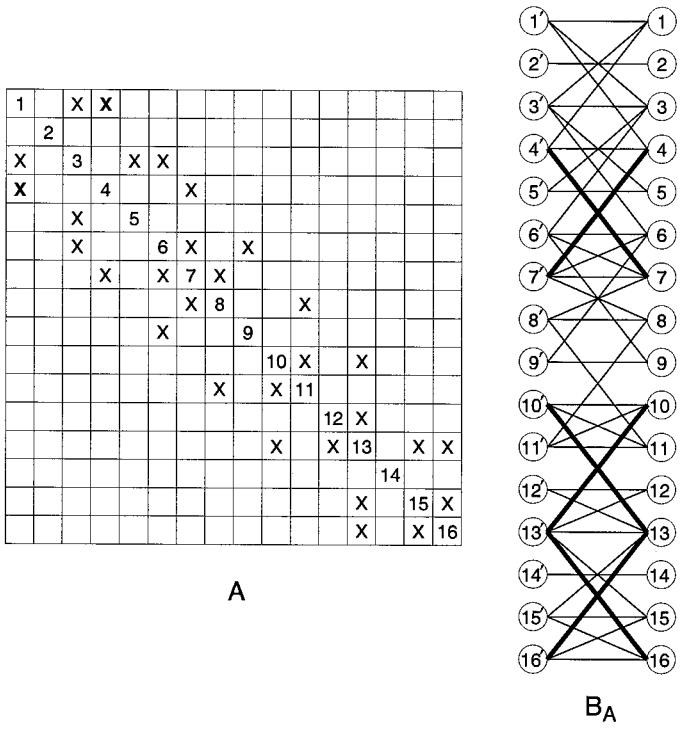


Figure 3.8: $A_{4,1}$'s Truncated Bulge Chasing Path

After $b-1$ columns of a 5-point problem have been reduced to tridiagonal form, the remainder of the band is completely filled in. The preponderance of fill within the band is largely due to the sequences of bulge chasing transformations required by the elimination of band nonzeros. Once the matrix's band has been filled there is no further opportunity to exploit sparsity beyond the densely banded form of the remaining submatrix. Consequently, the flop and transformation requirements of a 5-point problem's reduction are identical in the highest order terms to $F^{G}_{R\text{-}S}$ and $T_{R\text{-}S}$ for a densely banded matrix of equivalent order and bandwidth.

The speed with which the band of a 5-point problem fills is typical of most large symmetric problems. Table 3.6 provides fill data for three additional sparse problems, from Section 2.2's sparse matrix test suite, preordered by GPS and tridiagonalized by `Trisymb`'s symbolic implementation of sparse R-S. Despite starting with relatively sparse bands, the unreduced portion of each problem's band is completely filled well before b columns are

Problem	n	bandwidth	Initial Off-Diagonal Band Density	# of columns (rows) eliminated before band is full.
5-Pt Problems	b^2	b	$\sim 2/b$	b-1
PLAT1919	1919	80(GPS)	10.1%	31
NOS3	960	65(GPS)	12.4%	19
BCSSTK09	1083	95(GPS)	8.8%	18

Table 3.6: Band Filling Characteristics of Sparse R-S

reduced to tridiagonal form.

Not all sparse symmetric problems can be permuted to obtain a relatively small bandwidth. As for densely banded matrices, a computational trade-off exists between the dense Givens or Householder and the band-preserving sparse R-S reduction algorithms. Given the speed with which a typical sparse band fills, the transition point for most sparse problems must be close to the $b = n/3$ value observed for densely banded matrices. Once again, however, the lower storage requirements of sparse R-S may influence algorithm selection.

In summary, typically high levels of fill severely limits the number of opportunities for sparse R-S to utilize band sparsity. In general, sparse R-S is almost completely reliant on bandwidth reducing preorderings to exploit sparsity and the second stage of the algorithm is only slightly superior to a densely banded R-S tridiagonalization approach.

3.5.3 Sparse Band Extensions of Lang's Algorithm

Section 3.4.2 describes Lang's reduction algorithm for the tridiagonalization of densely banded matrices. This section explores the potential of extending Lang's techniques to create a sparse R-S like two stage reduction algorithm for general sparse symmetric matrices. Once again, the algorithm's first stage preorders the sparse matrix to reduce bandwidth, while the second stage of the tridiagonalization employs a modified form of Lang's algorithm that attempts to exploit band sparsity. To improve the construction and application of Householder transformations for a sparse band, we envisage modifying Lang's algorithm with changes analogous to the first and third enhancements of R-S made in Section 3.5.2.

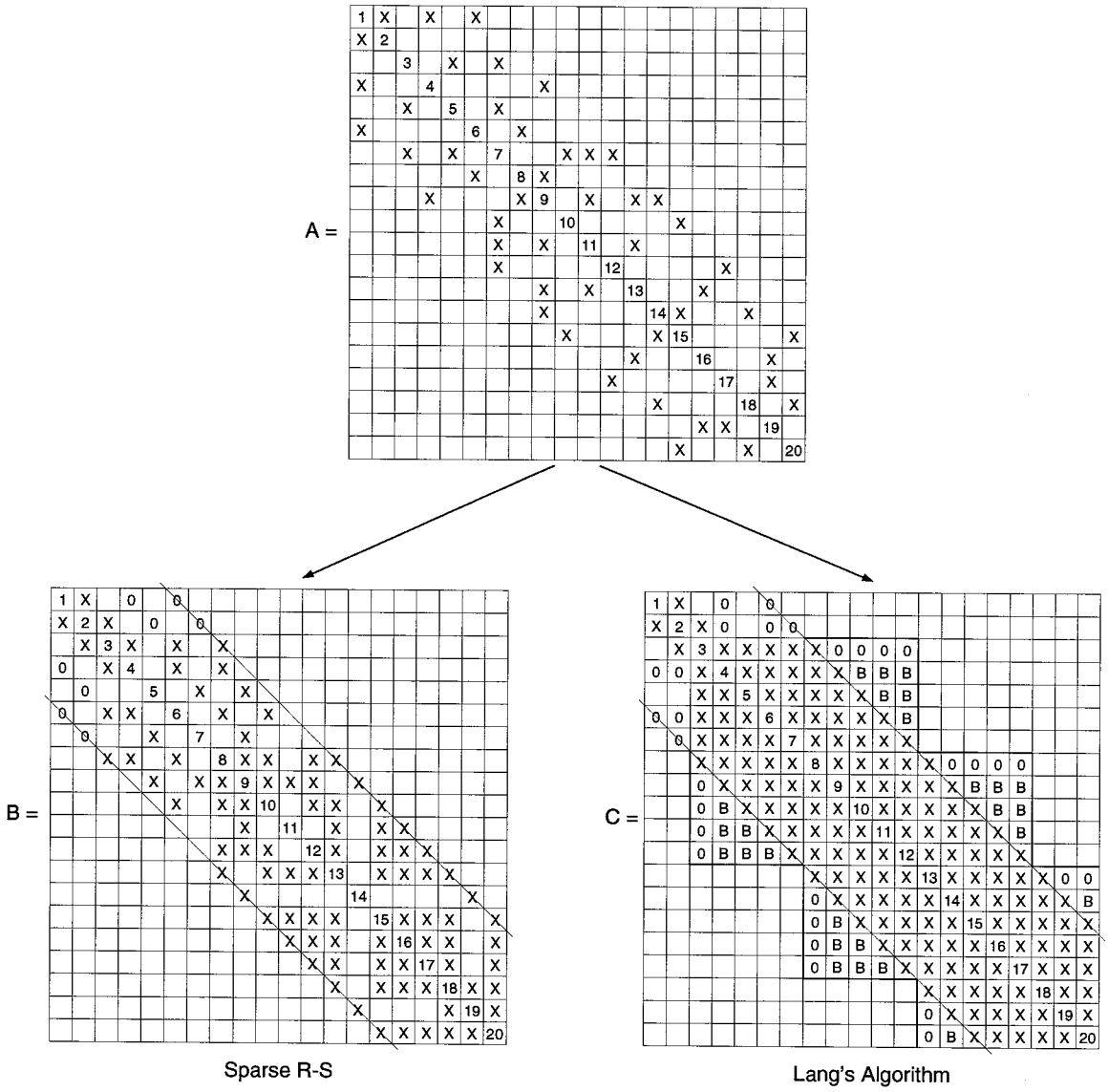


Figure 3.9: A Partial Tridiagonalization, Sparse R-S versus Lang's Algorithm

To overcome its inherently higher algorithmic complexity, the sparse Lang algorithm must significantly improve the exploitation of band sparsity relative to R-S. Discussion in Sections 2.3 and 2.7.3, however, has shown that a Householder transformation generally produces more fill entries than equivalent sequences of Givens transformations. As a result, the sparse Lang algorithm typically produces fill in the unreduced portion of the band even more rapidly than sparse R-S. Figure 3.9 illustrates two partial reductions of a small sparse matrix, A, which we assume has been preordered to reduce bandwidth. Matrix B illustrates the relatively sparse band remaining after R-S has reduced the first two columns of the band to tridiagonal form. The unreduced portion of the band does not become dense until R-S eliminates the third band nonzero from column 5. In contrast, reducing the first two columns of the band with Lang's algorithm completely fills the remainder of the band and there is no further opportunity to exploit band sparsity.

The results of this small example generalize to large practical sparse problems. Although sparse R-S has difficulty exploiting band sparsity, sparse extensions of Lang's algorithm exhibit more prolific fill characteristics and take even less advantage of band sparsity. In the following chapter we introduce an alternative to sparse R-S, which more successfully utilizes band sparsity to significantly improve reduction efficiency.

Chapter 4

Bandwidth Contraction Algorithms for Sparse Tridiagonalization

As shown in the previous chapter, attempts to adapt Givens Reduction for the tridiagonalization of sparse symmetric matrices are confronted with overwhelming levels of fill entries that quickly destroy matrix sparsity. Alternatively, the sparse R-S algorithm accepts the inevitability of fill entries, but actively restricts their accumulation to the band of a permuted matrix. Unfortunately, the band typically fills quickly and sparse R-S places almost complete reliance on the preordering algorithm to exploit matrix sparsity. This chapter presents alternative approaches to sparse tridiagonalization. They also use bandwidth reducing preorderings and band-preserving reduction techniques, but reorder the elimination sequence to more fully exploit internal band sparsity.

We begin this chapter with the development of our Bandwidth Contraction algorithm for the tridiagonalization of symmetric sparse matrices. Combining this successful algorithm with the sparse R-S algorithm, we then produce an effective hybrid tridiagonalization algorithm. The following section describes key aspects of the numerical implementations of both algorithms. Using these implementations, the chapter's final section describes extensive experimentation with the Bandwidth Contraction and hybrid tridiagonalization algorithms. In comparison to the Rutishauser-Schwarz approach, both algorithms are shown to dra-

matically reduce the computational requirements of sparse tridiagonalization.

4.1 Bandwidth Contraction

The sparse Bandwidth Contraction algorithm, or BC, is similar in many respects to sparse R-S. Both algorithms employ bandwidth reducing preorderings and constrain the accumulation of fill entries to a maintainable substructure of the original matrix. The Bandwidth Contraction algorithm, however, improves the efficiency of sparse tridiagonalization by rearranging the elimination of nonzeros to exploit a commonly observed characteristic of sparsely banded matrices.

4.1.1 Motivation

The sparse tridiagonalization techniques explored in this section are motivated by the following observation. A bandwidth reducing preordering frequently produces a permuted matrix whose profile consists of varying length *spikes* of nonzeros extending from the main diagonal. The longest spike defines the bandwidth of the permuted matrix, as shown in Figure 4.1. For many practical problems the spikes of the permuted matrix are of dramatically different length. As an example, the black dots in Figure 4.2's plot illustrate the nonzero sparsity structure of the Harwell–Boeing problem CAN 268 preordered by GPS. The new sparse tridiagonalization approach attempts to exploit variation in spike length. Although fill cannot be avoided, the bandwidth of the matrix could be significantly reduced with relatively few transformations, if the ends of the longest spikes could be clipped off at low cost before the contracted band becomes dense.

4.1.2 The Sparse Bandwidth Contraction Algorithm

There are a number of ways that the ends of the profile's longest spikes could be clipped off with Givens transformations. One way to approach the clipping process is to rearrange the elimination order of a band-preserving tridiagonalization so that the matrix is reduced to tridiagonal form diagonal by diagonal (outermost diagonal first), rather than column

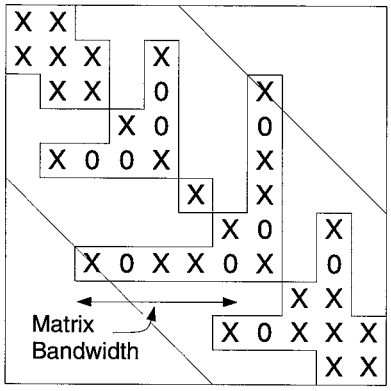


Figure 4.1: Matrix Bandwidth and Spike Length

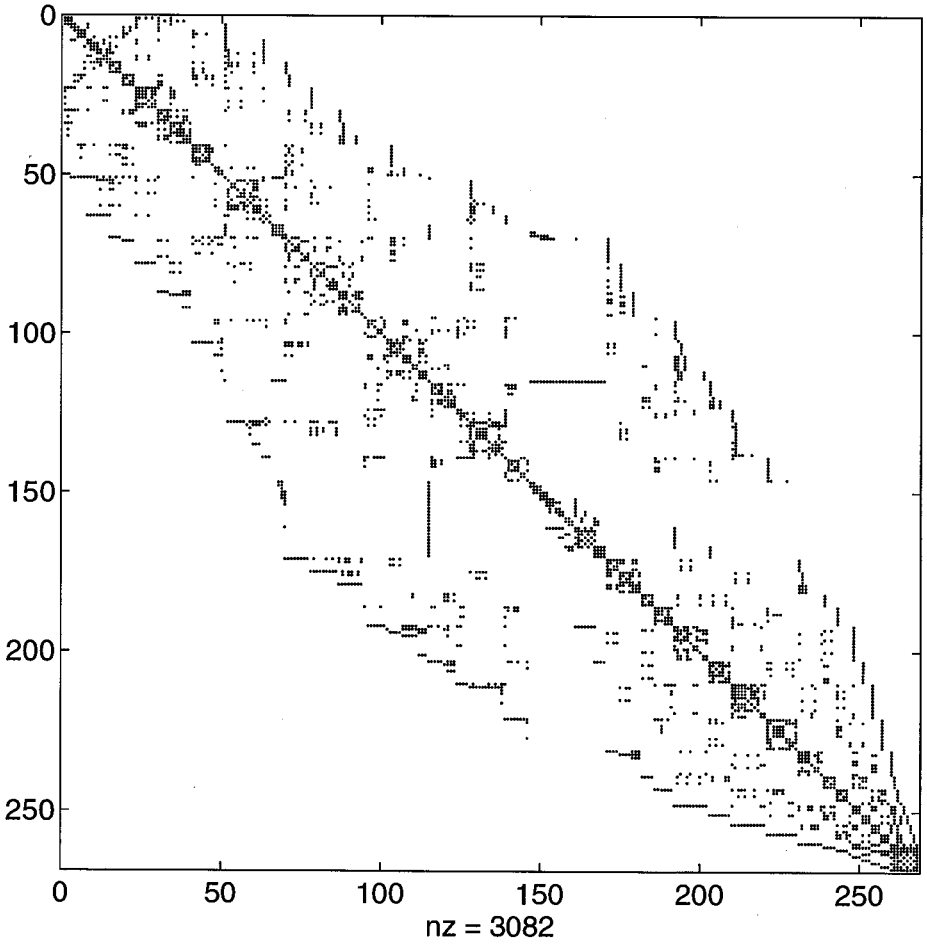


Figure 4.2: The Sparsity Structure of CAN 268 with a GPS Preordering

1. $A := P^T A P$, where P is a bandwidth reducing permutation matrix.
2. $b :=$ bandwidth(A)
3. FOR $\widehat{b} := b$ DOWNTO 2 DO /*Tridiagonalize A.*/
   FOR $col := 1$ TO $n - \widehat{b}$ DO
   IF $A_{col+\widehat{b},col} \neq 0$ THEN /*Zero $A_{col+\widehat{b},col}$.*/
   IF $A_{col+\widehat{b}-1,col} = 0$ THEN
   Exchange rows/columns $(col + \widehat{b})$ and $(col + \widehat{b} - 1)$ in A.
   ELSE
   $A := G(col + \widehat{b}, col + \widehat{b} - 1, \theta)^T \; A \; G(col + \widehat{b}, col + \widehat{b} - 1, \theta)$
   (Exploit band sparsity of modified rows and columns.)
   IF bandwidth(A) $> \widehat{b}$ THEN
   Chase bulges with additional adjacent Givens
   transformations or row/column exchanges.
   ENDIF /*Outermost IF*/

Figure 4.3: The Sparse Bandwidth Contraction Algorithm

by column as in sparse R-S. A diagonally-oriented band-preserving tridiagonalization for densely banded symmetric matrices has been previously considered [Sch63, Wil65], but was superseded on sequential machines by the Rutishauser-Schwarz's column-oriented reduction [Rut63, Sch71]. (See Section 4.2.1 for a detailed comparison of algorithm complexity.) To our knowledge, however, no one has considered the relative merits of the two reduction paradigms extended for general application to sparse symmetric matrices. (For densely banded matrices, the LAPACK [ABB+92] project considered the relative merits of these two algorithms implemented on vector machines.)

As shown in Figure 4.3, our sparse Bandwidth Contraction algorithm (BC) uses the diagonally-oriented spike clipping process to completely tridiagonalize a sparse symmetric matrix. To exploit band zeros, the three modifications made to the basic R-S algorithm in Section 3.5.2 are also incorporated into the Bandwidth Contraction algorithm.

The Bandwidth Contraction algorithm begins by symmetrically permuting the matrix to reduce bandwidth. Then, starting with $A_{b+1,1}$, the band's outermost diagonal is scanned

for its first nonzero entry. This entry is eliminated using either an adjacent Givens transformation or a row/column exchange, depending upon the nonzero status of the *zeroing entry.* If a nonzero entry is created beyond the current bandwidth, the bulge is chased off the end of the matrix as described in Section 3.4.1. The scanning and reduction of the outermost diagonal continues until all nonzeros have been eliminated. At this point the current bandwidth of the matrix is reduced by one and the reduction process continues with the next diagonal.

As for the sparse R-S algorithm, Bandwidth Contraction uses adjacent transformations exclusively, avoiding the creation of multiple entry bulges and the concomitant extra bulge chasing transformations. The additional bulge chasing transformations not only increase computational costs, they also accelerate the introduction of fill entries into the band. An added complication for a diagonally-oriented tridiagonalization is that nonadjacent transformations may reintroduce fill entries in previously zeroed positions of the diagonal, severely restricting the utility of this transformation class.

The small example in Figure 4.4 demonstrates some of the potential difficulties associated with nonadjacent transformations. When the Bandwidth Contraction algorithm eliminates entry $A_{5,1}$ and chases the associated bulge, it uses one row/column exchange and two nontrivial adjacent transformations, producing matrix B. Alternatively, suppose we decide to eliminate $A_{5,1}$ with the nonadjacent transformation $G(2,5,\theta)^T A G(2,5,\theta)$ to avoid having to eliminate a nonzero from column 1, and chase its associated bulge, during the next diagonal's reduction. Application of this transformation creates a three entry bulge as shown in intermediate matrix C. Chasing this multiple entry bulge off the end of the matrix to regain the matrix's original banded form requires 11 additional nontrivial transformations. As shown by matrix D, these transformations completely fill the band beyond row and column 5. Consequently, using a nonadjacent transformation in this manner is definitely not cost-effective.

This discussion is not intended to completely rule out the use of nonadjacent transformations. There are special sparsity patterns for which nonadjacent transformations are

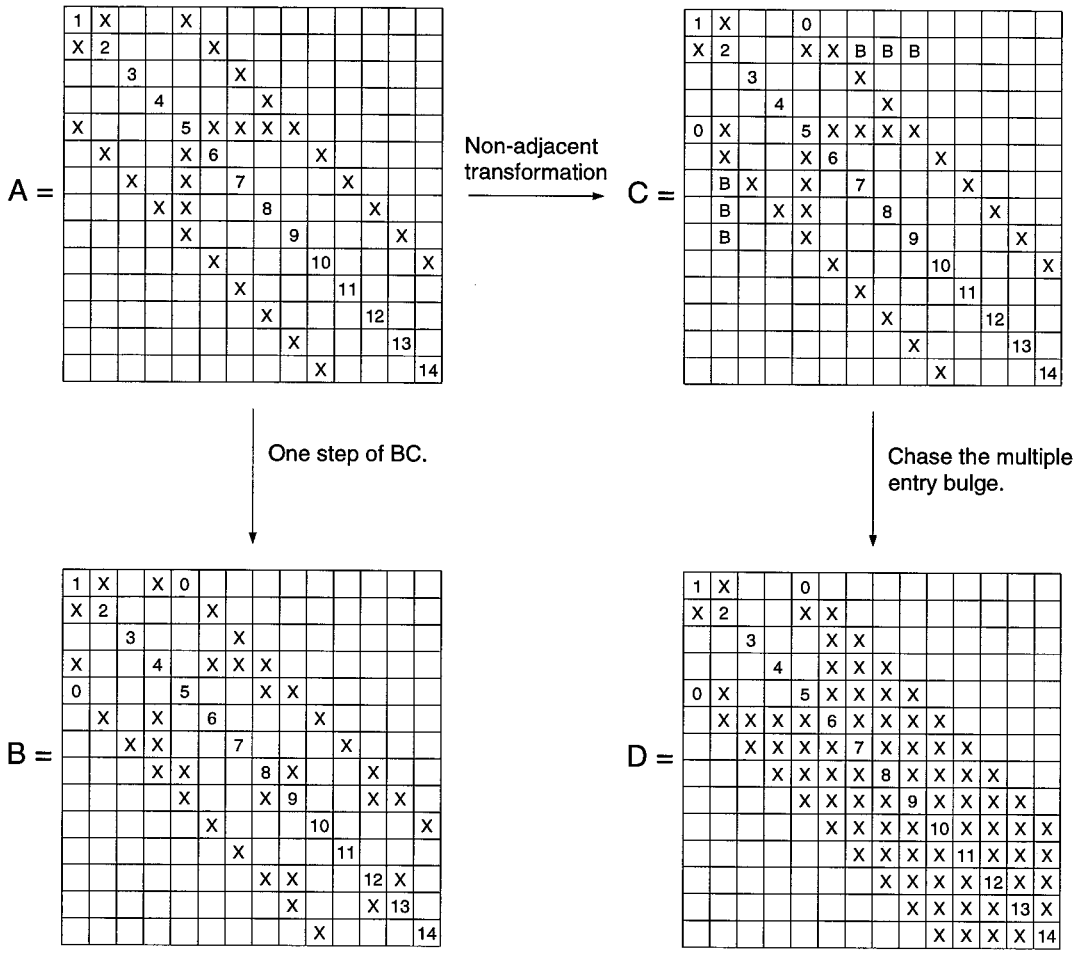


Figure 4.4: A Nonadjacent Transformation Example

beneficial. Future study will explore the potential role of nonadjacent transformations in more sophisticated or special purpose bandwidth contraction algorithms.

4.1.3 A Demonstration of Bandwidth Contraction

To illustrate the potential effectiveness of Bandwidth Contraction, we have manipulated the small contrived example in Figure 4.5 with the symbolic reduction tool `Xmatrix`. The top matrix in Figure 4.5 shows the original sparse matrix, A, with its nonzero entries indicated by "X"s and the numbered diagonal. It is assumed that A has already been permuted to reduce its bandwidth to 6. Two additional matrices, C and D, illustrate A after a

partial reduction by sparse R-S or Bandwidth Contraction. In both C and D a "0" marks the positions of eliminated band nonzeros. Finally, reported flop counts assume that both reductions employ fast Givens transformations.

Matrix C illustrates A after sparse R-S has reduced its first three columns to tridiagonal form. Despite the highly sparse nature of the original problem, the remainder of the band is almost completely filled. The entire tridiagonalization uses 8 row/column exchanges and 132 nontrivial transformations, requiring a total of 7232 flops.

Matrix D illustrates A after the Bandwidth Contraction algorithm has eliminated the three outermost nonzero diagonals and contracted the bandwidth to 3. Although the elimination of nonzeros once again produces fill entries within the band relatively quickly, the algorithm is able to efficiently exploit the sparsity of the band away from the main diagonal. For example, the Bandwidth Contraction algorithm eliminates the entire 6^{th} and 5^{th} subdiagonals of the band at the relatively low cost of 216 flops, using 7 row/column exchanges and 4 nontrivial transformations. The complete tridiagonalization uses 12 row/column exchanges and 163 nontrivial transformations, requiring a total of 6537 flops. For this example, the computational requirements of tridiagonalization with Bandwidth Contraction, as measured by flop counts, are approximately 9.6% lower than for the sparse R-S approach.

It is important to note that the number of nontrivial transformations used by a tridiagonalization is a misleading metric of algorithm performance. The Bandwidth Contraction algorithm requires more transformations, but generally fewer nonzeros are modified by each nontrivial transformation, permitting a lower total flop count.

The key to the success of the Bandwidth Contraction algorithm is the elimination of the outermost diagonals at low cost. As the result of several contributing factors, Bandwidth Contraction is able to exploit and prolong the advantages of sparsity in the outermost subdiagonals. First, when BC requires nontrivial adjacent transformations, they are often applied to well separated pairs of rows and columns, insulating the effects of fill from one transformation sequence to the next. In contrast, sparse R-S's initial band zeroing transformations and associated bulge chasing transformations are applied to groups of neighboring

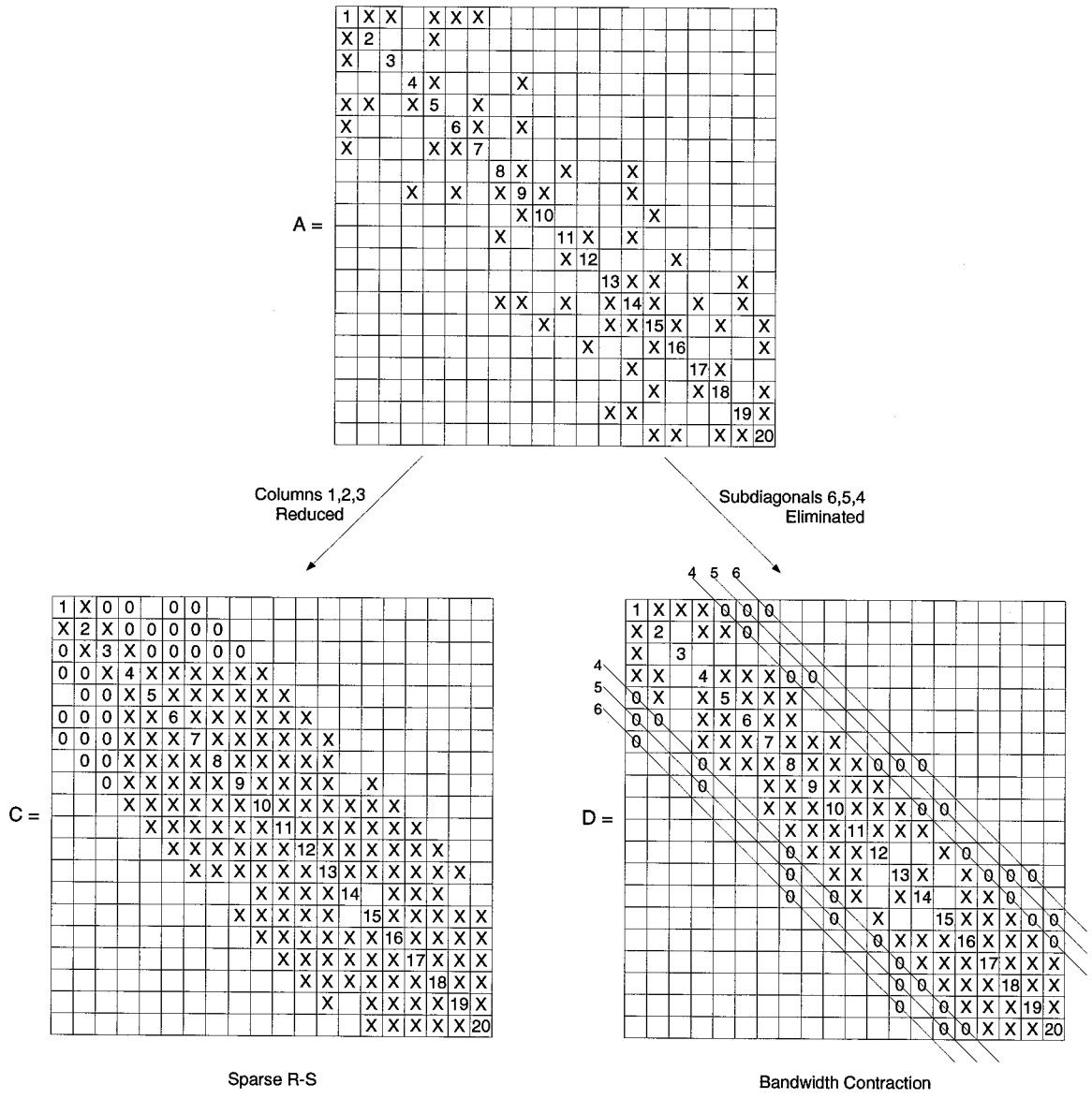


Figure 4.5: A Partial Tridiagonalization, Sparse R-S versus Bandwidth Contraction

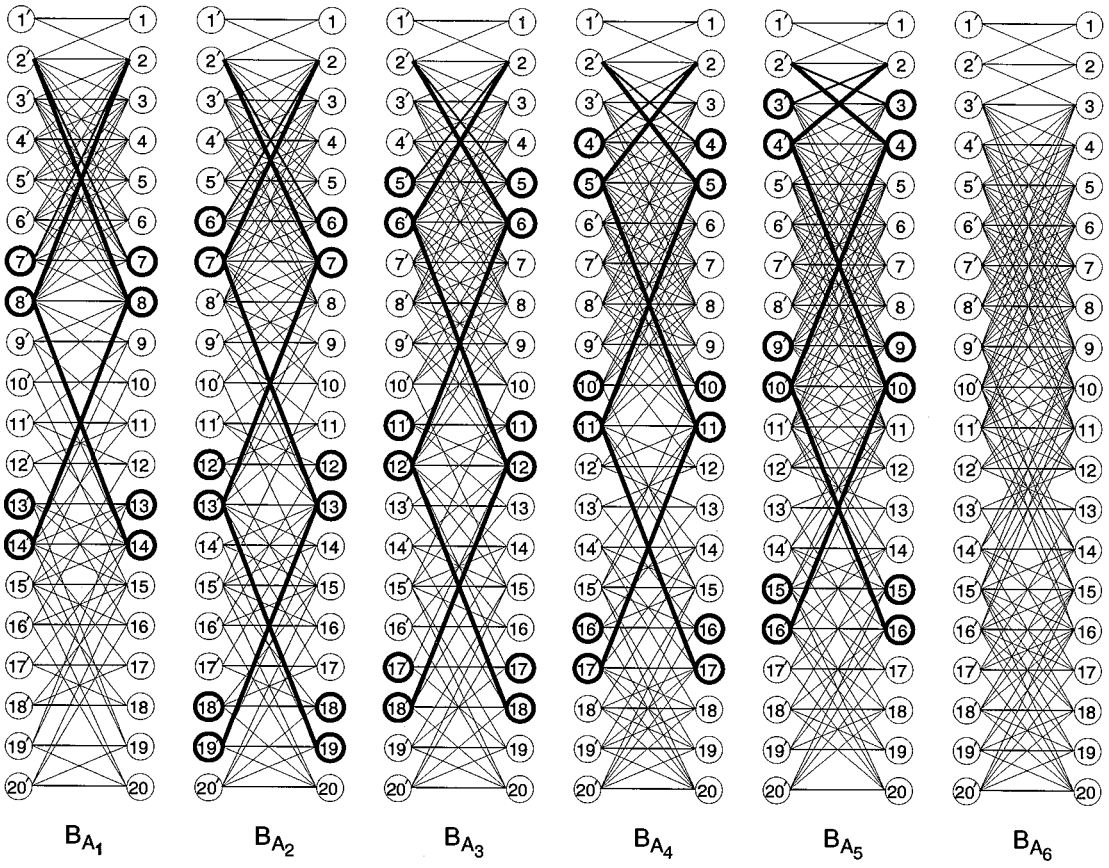


Figure 4.6: Cascading Fill Entries

rows and columns. These initial transformations produce a cascade of fill entries, typically not observed for BC, which quickly fills the band despite a matrix's initial sparsity.

As an example, bipartite graphs B_{A_1} through B_{A_5} in Figure 4.6 model the sparsity structures of Figure 4.5's example just before sparse R-S symmetrically eliminates nonzeros $A_{8,2}$, $A_{7,2}$, $A_{6,2}$, $A_{5,2}$, and $A_{4,2}$ respectively from column 2. The final bipartite graph, B_{A_6}, models A's sparsity structure after the completion of column 2's reduction. Graphs B_{A_1} to B_{A_5} each highlight the edges corresponding to the band nonzero pair selected for elimination, their bulge chasing paths, and the pairs of nodes modified by the associated band zeroing and bulge chasing transformations. We observe that in neighboring bipartite graphs the pairs of highlighted nodes share a common node. Consequently, nonzeros are

carried from one transformation sequence to the next, either through adjacency set union or exchange, producing a cascade of fill entries. Comparing B_{A_1} and B_{A_6}, the elimination of column 2 substantially reduces band sparsity, introducing 48 new band nonzeros.

In addition to reducing instances of fill cascading, BC's diagonally-oriented reduction requires fewer bulge chasing transformations for initial band nonzero eliminations. As BC eliminates the outermost diagonal, bulge chasing sequences must shorten, while the length of the sequences used by sparse R-S remains relatively constant as it reduces the first few columns. Consequently, the initial stage of the Bandwidth Contraction algorithm produces fewer band fill entries.

Avoiding fill entries prolongs the survival of sparsity in the outermost diagonals and improves the efficiency of BC's reduction in several ways. First, BC must eliminate fewer band nonzeros to effect each diagonal's reduction. In addition, the sparsity of the outermost diagonals permits Bandwidth Contraction to cheaply eliminate many band nonzeros and associated bulges, without fill, using row/column exchanges. Finally, the sparsity of the outermost diagonals often produces truncated bulge chasing paths (see Section 3.5.2), reducing transformation requirements and fill production.

In summary, the advantages of BC's sparse reduction outlined in the preceding discussion often allow it to perform a partial tridiagonalization that significantly contracts a matrix's bandwidth at low cost before producing a densely banded intermediate matrix. In general, the relative success of the sparse tridiagonalization algorithms depends on problem specific sparsity structures. The extensive experimental analysis of Section 4.4 confirms the relative advantage experienced by the Bandwidth Contraction algorithm for many practical sparse problems.

4.2 A Hybrid Tridiagonalization Algorithm

The goal of this dissertation is to produce generally applicable sparse eigenvalue routines. Unfortunately, the computational requirements of sparse Bandwidth Contraction are not smaller than those of sparse R-S for every problem's banded sparsity structure. In this

section we combine the individual strengths of each reduction algorithm to produce the versatile Hybrid Bandwidth Contraction tridiagonalization algorithm, or HYBBC*, which is suitable for the reduction of a broad range of sparse symmetric matrices.

4.2.1 Motivation

Two metrics of tridiagonalization algorithm cost are flop and transformation counts F and T. As demonstrated in the previous section, the Bandwidth Contraction algorithm may be able to significantly reduce the bandwidth of a sparsely banded matrix at relatively low cost. Consequently, for sparsely banded matrices Bandwidth Contraction flop counts are smaller than for sparse R-S, despite larger transformation counts. If the band of a matrix is dense, however, Rutishauser and Schwarz's column-oriented band-preserving tridiagonalization is superior in both measures of work.

Table 4.1 provides formulae for the tridiagonalization costs for the fast Givens Bandwidth Contraction variant applied to a densely banded, symmetric matrix of bandwidth b. Using Section 2.4's analysis framework, we provide a comprehensive analysis leading

F_{BC}^{FG}	T_{BC}
$\left(4b-4+10\sum_{k=2}^{b}\left(\frac{1}{k}\right)\right)n^2+\left(22-16b-3b^2\right)n$ $+\frac{2b^3}{3}+\frac{5b^2}{2}+\frac{11b}{6}-5+\sum_{k=2}^{b}((\frac{10}{k}+2)C_{BC})$	$\frac{n^2\sum_{k=2}^{b}(\frac{1}{k})-(b-1)n+\sum_{k=2}^{b}\frac{C_{BC}}{k}}{2}$

Table 4.1: Tridiagonalization Costs of the Bandwidth Contraction Algorithm for a Densely Banded Matrix.

to these results [Cav93]. The analysis assumes that $b < (n+1)/2$ and that an equal proportion of the two types of fast Givens transformations are employed. The analysis ignores the potential cost of the periodic rescaling required by fast Givens transformations. C_{BC} is the nonanalytic term $Mod(n,k)(k-Mod(n,k))$. When $b \ll n$ the $Mod(n,k)$ terms can be safely ignored without incurring significant errors. Comparison of $F_{\text{BC}}^{\text{FG}}$ and $F_{\text{R-S}}^{\text{FG}}$, from Equation 3.1, shows that the flop requirements of the Bandwidth Contraction algorithm are larger than for the R-S algorithm applied to the same problem. To demonstrate

*In [Cav94] we also refer to this algorithm with the acronym BANDHYB.

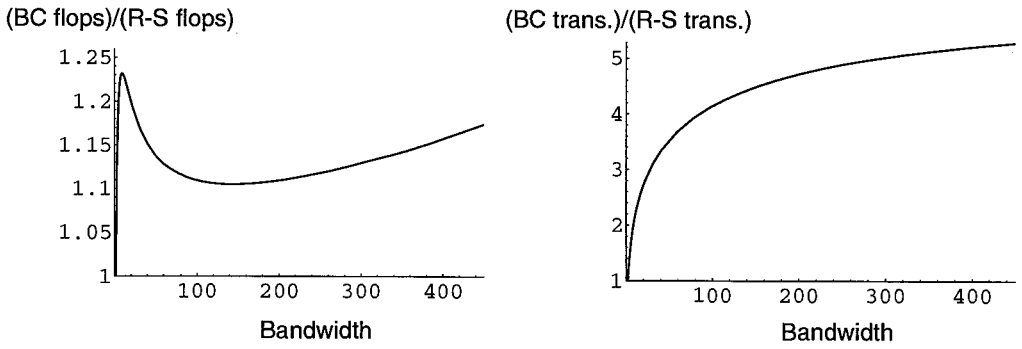


Figure 4.7: The Flop and Transformation Requirements of BC relative to R-S for a Densely Banded Matrix, n=1000.

the potential difference in tridiagonalization costs, the first graph in Figure 4.7 plots the flop requirements of Bandwidth Contraction, normalized by $F_{\text{R-S}}^{\text{FG}}$, against bandwidth for a densely banded matrix.

While $F_{\text{BC}}^{\text{FG}}$ is typically 10 to 25% larger than $F_{\text{R-S}}^{\text{FG}}$, for problems with nontrivial bandwidth the difference between T_{BC} and $T_{\text{R-S}}$ can be much greater, as shown by the second graph in Figure 4.7. At first glance there may seem to be an inconsistency in our analysis. As mentioned in Section 4.1.3, however, generally transformation counts are a misleading metric of tridiagonalization costs. $F_{\text{R-S}}^{\text{FG}}$ and $F_{\text{BC}}^{\text{FG}}$ are closer than predicted by T_{BC} and $T_{\text{R-S}}$, because as Bandwidth Contraction reduces a matrix's bandwidth, the number of nonzeros modified by each transformation generally declines. The computational effort of applying later transformations is reduced, while for the R-S algorithm the cost of each transformation remains relatively constant.

As an aside to our discussion of sequential algorithms, vector machines complicate the relative efficiency analysis of tridiagonalization algorithms. In general, vector machines put more weight on T relative to F. During the development of LAPACK's [ABB+92] replacement for BANDR, SSBTRD, several band-preserving tridiagonalization algorithms were tested on vector machines [DC92]. In very general terms, for small n (less than 50) or matrices with moderate bandwidth ($20 \leq b < 50$), it was found that vectorized code based

on a diagonally-oriented elimination is the fastest approach. For other densely banded matrices, variants of a column-oriented tridiagonalization are more efficient. Emphasizing the importance of good performance for large n and small bandwidth, SSBTRD is based on the column-oriented, vectorized algorithm of Kaufman [Kau84].

4.2.2 The Hybrid Bandwidth Contraction Algorithm

The observations of the previous subsection suggest a hybrid tridiagonalization algorithm. While the band of the intermediate reduction matrix remains sufficiently sparse, the hybrid algorithm employs the sparse Bandwidth Contraction scheme. When it is found that BC can no longer efficiently exploit band sparsity, the reduction switches to sparse R-S to complete the tridiagonalization. To avoid redundant elimination of band nonzeros, the hybrid algorithm always completes a nonzero diagonal's reduction before switching to sparse R-S.

The most sensitive design issue for this hybrid algorithm is the selection of a generally applicable strategy to govern the transition between the algorithm's two stages. Ideally the hybrid algorithm should switch to sparse R-S at the transition bandwidth, b^t, minimizing the following problem dependent cost function.

$$\text{Cost\_BC}(b \rightarrow b^t) + \text{Cost\_R-S}(b^t \rightarrow 1)$$

The matrix's original bandwidth is b and functions Cost_BC() and Cost_R-S() represent the computational requirements of BC and sparse R-S performing the indicated reduction in bandwidth. Using fill level monotonicity arguments, it is not difficult to see that a transition strategy could minimize this function by comparing the following reduction costs before each diagonal's elimination. (b^c represents the bandwidth of the reduction's current intermediate matrix.)

$$\begin{aligned} C1 &= \text{Cost\_BC}(b^c \rightarrow (b^c - 1)) + \text{Cost\_R-S}((b^c - 1) \rightarrow 1) \\ C2 &= \text{Cost\_R-S}(b^c \rightarrow 1) \end{aligned}$$

As long as $C1 < C2$ the hybrid algorithm should continue the reduction with the sparse Bandwidth Contraction algorithm.

When the band of the intermediate matrix is dense, we can determine the flop components of $C1$ and $C2$ exactly using the analysis results of Sections 3.4.1 and 4.2.1. While zero entries remain in the band, however, these analyses do not apply and accurate resolution of the cost comparison is difficult. The computational requirements of BC and sparse R-S are influenced by both the nonzero density of the band and problem specific sparsity structures. The design of transition strategies that take into account a band's particular sparsity structure is hindered by a lack of model problems that exhibit all significant sparsity characteristics of practical problems. In addition, a formal analysis of BC or R-S's reduction of problems with general sparsity patterns is not feasible. Alternatively, we consider transition strategies that provide an approximate solution to the comparison of costs $C1$ and $C2$ by thresholding some measure of band density or "fullness".

There are many possible metrics for measuring the fullness of the contracted band. The most obvious choice is to directly monitor the number of nonzero entries. Maintaining such detailed knowledge of the band's sparsity pattern, however, may require significant levels of overhead. Instead, we suggest regulating the transition between Bandwidth Contraction and sparse R-S by a threshold on the number of nonzero entries in the outermost nonzero subdiagonal. The transition is made when the number of nonzeros is greater than some fraction of the subdiagonal's length. As shown below, monitoring the number of nonzeros in the next subdiagonal can be cheaply integrated into Bandwidth Contraction. Equally important, this transition regulation technique provides a good approximation of measuring true band density. As shown in Section 4.3.2, sparsity within the band is best exploited at the transformation level, which is controlled to a large extent by the number of zeros in the outermost diagonal. In addition, if BC continues the reduction once the outermost diagonal is full, or nearly so, each successive subdiagonal eliminated will have a similar density and the band will quickly fill.

The pseudocode in Figure 4.8 describes the Hybrid Bandwidth Contraction tridiagonalization algorithm (HYBBC). The second stage of the hybrid algorithm is identical to the description of sparse R-S in Section 3.5.2, except it does not perform an addition preordering. Let *threshold* be the fraction of the outermost diagonal that must be nonzero before the

1. $A := P^T AP$, where P is a bandwidth reducing permutation matrix.

2. $b :=$ bandwidth(A)

3. /*Initialize *nzcnt* for the outermost diagonal.*/
   $nzcnt := 0$
   FOR i:= 1 TO n-b DO
   IF $A_{i+b,i} \neq 0$ THEN $nzcnt := nzcnt+1$

4. (a) $b^c := b$

   (b) /*While the matrix is not tridiagonal and the threshold has not been*/
   /*met, eliminate the outermost nonzero diagonal.*/
   WHILE ($(b^c \geq 2)$ AND $(nzcnt < (threshold * (n - b^c)))$) DO

   i. $nzcnt := 0$

   ii. FOR $col := 1$ TO $n - b^c$ DO
   IF $A_{col+b^c,col} \neq 0$ THEN /*Zero $A_{col+b^c,col}$.*/
   IF $A_{col+b^c-1,col} = 0$ THEN
   Exchange rows/columns $(col + b^c)$ and $(col + b^c - 1)$ in A.
   ELSE
   $A := G(col + b^c, col + b^c - 1, \theta)^T \; A \; G(col + b^c, col + b^c - 1, \theta)$
   (Exploit band sparsity of modified rows and columns.)
   IF bandwidth(A) $> b^c$ THEN
   Chase bulges with additional adjacent Givens
   transformations or row/column exchanges.
   ENDIF /*Outermost IF*/
   IF $A_{col+b^c-1,col} \neq 0$ THEN $nzcnt := nzcnt+1$

   iii. IF $A_{n,n-b^c+1} \neq 0$ THEN $nzcnt := nzcnt+1$

   iv. $b^c := b^c - 1$

   (c) IF $b^c > 1$ THEN complete tridiagonalization with sparse R-S.

Figure 4.8: The Hybrid Bandwidth Contraction Tridiagonalization Algorithm

transition to sparse R-S is made and let *nzcnt* be the number of nonzeros in the next subdiagonal. The algorithm is able to check the nonzero status of entry $A_{col+b^c-1,col}$ after the elimination of entry $A_{col+b^c,col}$ because the entries of row $(b^c + col - 1)$ will not be modified again during the reduction of the current outermost diagonal. The resource requirements of the band density metric are minimal—one additional integer variable, and during each diagonal's reduction $n - b^c$ comparisons and at most $n - b^c + 1$ integer operations.

4.2.3 Performance of the Hybrid Tridiagonalization Algorithm

Consider the application of the Hybrid Tridiagonalization algorithm to matrix A of Figure 4.5 with a *threshold* of 0.85. In the first stage of the tridiagonalization BC reduces the three outermost nonzero subdiagonals, producing matrix D of Figure 4.5. The Hybrid Tridiagonalization algorithm then transfers control to sparse R-S to complete the reduction to tridiagonal form. Table 4.2 summarizes the computational requirements of all three sparse tridiagonalization algorithms, assuming the use of fast Givens transformations. The

Method	Row/Column Exchanges	Nontrivial Transformations	Flops
Sparse R-S	8	132	7232
Bandwidth Contraction	12	163	6537
Hybrid Tridiagonalization	12	136	5880

Table 4.2: Tridiagonalization Summary for a Small Sparse Example

hybrid algorithm requires approximately 19% and 10% fewer floating point operations than sparse R-S and Bandwidth Contraction respectively. It is interesting to note that the total number of nontrivial transformations required by the Hybrid Tridiagonalization algorithm is significantly lower than for Bandwidth Contraction and only marginally higher than for sparse R-S.

The test problem of Figure 4.5 is obviously a trivial example. The experiments described in Section 4.4, however, show that the Hybrid Tridiagonalization algorithm dramatically reduces the computational requirements of sparse tridiagonalization for a wide range of

sparse problems.

4.3 Numerical Implementations of BC and HYBBC

This section describes features of numerical implementations of the Bandwidth Contraction and Hybrid Bandwidth Contraction tridiagonalization algorithms. Following a general description of both implementations, we study the ability of individual nontrivial Givens transformations to exploit band sparsity. The outcome of this study influences the sparsity techniques employed by both implementations and dictates the form of the banded data structure used by the routines. Finally, we outline new rescaling techniques for implementations using fast Givens transformations to effect diagonally-oriented reductions.

4.3.1 Implementation Basics

The first stage of both BC and HYBBC preorders a sparse problem to reduce bandwidth. We rely upon existing preordering algorithm implementations to conduct this phase of the reduction and concentrate upon the implementation of each tridiagonalization algorithm's second stage. The implementation of the Bandwidth Contraction algorithm was created by rewriting EISPACK's FORTRAN routine BANDR (an R-S code) to perform a sparse, diagonally oriented, band-preserving tridiagonalization. Using this new routine, we implemented the Hybrid Bandwidth Contraction algorithm by augmenting the BC routine with the described thresholding strategy, and a transition to a modified version of BANDR that omits initializations. The hybrid algorithm switches to a column-oriented scheme when the band is dense, or nearly so. Given the speed with which a typical sparse band fills during an R-S reduction (see Section 3.5.2), using a sparse R-S code for this portion of HYBBC is not warranted. Otherwise the implementations of BC and HYBBC closely follow the algorithms in Figures 4.3 and 4.8 with one exception. That is, based on the outcome of the following section's study, we implement the Bandwidth Contraction algorithm with only two of the three sparse algorithm modifications listed in Section 3.5.2.

4.3.2 Dense Versus Sparse Band Transformations

Unlike the first two sparsity modifications listed in Section 3.5.2, exploiting the sparsity of a pair of rows or columns during the application of a transformation requires significant overhead. To determine if the potential savings are worthy of the increased overhead, we conduct experiments with the symbolic reduction tool `Trisymb`. For 15 larger problems, Table 4.3 compares the flop requirements of a HYBBC variant that fully exploits the unioned sparsity structure of modified rows and columns, with a second HYBBC variant that treats the band as dense while applying a nontrivial transformation. Accounting procedures differ between the two simulations, but sparsity structures of the intermediate reduction matrices are identical. The simulated reductions model fast Givens transformations exclusively. The bandwidth of the original matrix preordered by GPS is given by b^{GPS} and b^t is the transition bandwidth.

Going from dense to sparse transformations, savings of 12–21.5% in the Bandwidth Contraction portion of the reduction are observed for 3 problems, but for the remaining matrices savings are less than 5%. Considering the cost of the entire hybrid tridiagonalization, the potential savings of sparse transformations are very small. The largest reduction is 1.1% and for the remaining problems the potential savings are 1% or lower. If standard Givens transformations replace fast Givens transformations, the results are essentially unchanged.

This study clearly shows band sparsity is best exploited at the transformation level by identifying entries that are already zero or that the algorithm can eliminate with an adjacent row/column exchange. Considering the storage and computational overhead required by a sparse data structure, performing sparse transformations is not beneficial to BC or HYBBC performance, and will not be pursued by the sparse tridiagonalization implementations described in this dissertation. In future work, however, sparse transformations will be reevaluated for the special case in which a partial bandwidth contraction is the end goal.

Name N, b^{GPS}	Transformation Type	b^t	Flops				
			Total	% Flop Reduction	BC	% Flop Reduction	R-S
ERIS1176	Dense Band	1	2.351e+08	—	2.351e+08	—	0
1176, 100	Sparse Band	1	2.329e+08	0.92	2.329e+08	0.92	0
BCSPWR08	Dense Band	8	5.635e+08	—	4.637e+08	—	9.98e+07
1624, 108	Sparse Band	8	5.578e+08	1.0	4.58e+08	1.2	9.98e+07
BCSSTK09	Dense Band	57	2.762e+08	—	1.441e+07	—	2.62e+08
1083, 95	Sparse Band	57	2.74e+08	0.78	1.226e+07	14.9	2.62e+08
BCSSTK12	Dense Band	1	4.462e+08	—	4.462e+08	—	0
1473, 62	Sparse Band	1	4.451e+08	0.25	4.451e+08	0.25	0
CAN 1054	Dense Band	8	3.034e+08	—	2.615e+08	—	4.2e+07
1054, 112	Sparse Band	8	3.02e+08	0.47	2.6e+08	0.55	4.2e+07
DWT 361	Dense Band	12	7.684e+06	—	8.127e+05	—	6.87e+06
361, 14	Sparse Band	12	7.645e+06	0.50	7.741e+05	4.8	6.87e+06
DWT 1242	Dense Band	12	2.787e+08	—	1.954e+08	—	8.33e+07
1242, 91	Sparse Band	12	2.768e+08	0.69	1.935e+08	0.99	8.33e+07
DWT 1007	Dense Band	12	1.016e+08	—	4.697e+07	—	5.46e+07
1007, 34	Sparse Band	12	1.013e+08	0.26	4.67e+07	0.57	5.46e+07
LSHP 577	Dense Band	3	3.213e+07	—	2.695e+07	—	5.18e+06
577, 25	Sparse Band	3	3.199e+07	0.42	2.681e+07	0.50	5.18e+06
NOS3	Dense Band	40	1.534e+08	—	5.594e+06	—	1.48e+08
960, 65	Sparse Band	40	1.527e+08	0.44	4.924e+06	12.0	1.48e+08
GR 30 30	Dense Band	30	1.024e+08	—	2.782e+06	—	9.96e+07
900, 49	Sparse Band	30	1.018e+08	0.58	2.185e+06	21.5	9.96e+07
PLAT1919	Dense Band	11	7.934e+08	—	6.087e+08	—	1.85e+08
1919, 80	Sparse Band	11	7.893e+08	0.53	6.045e+08	0.69	1.85e+08
SSTMODEL	Dense Band	1	1.555e+09	—	1.555e+09	—	0
3345, 82	Sparse Band	1	1.548e+09	0.46	1.548e+09	0.46	0
1138 BUS	Dense Band	3	2.148e+08	—	1.946e+08	—	2.02e+07
1138, 126	Sparse Band	3	2.124e+08	1.1	1.922e+08	1.2	2.02e+07
BCSSTK28	Dense Band	24	1.437e+10	—	1.238e+10	—	1.99e+09
4410, 323	Sparse Band	24	1.422e+10	1.0	1.223e+10	1.2	1.99e+09

Table 4.3: A Comparison of Two Symbolic Implementations of HYBBC using Dense or Sparse Band Fast Givens Transformations.

4.3.3 A Densely Banded Data Structure

As a consequence of the preceding section's study, our implementations of BC and HYBBC keep the densely banded data structure of BANDR. By exploiting the symmetry of sparse problems and similarity transformations, these algorithms need only consider transformation modifications to the lower triangular portion of each matrix. The main diagonal and each subdiagonal of the band's lower triangular portion is stored in a separate column of an $n \times (b+1)$ double precision array. The storage of bulge entries does not necessitate an additional column of storage. Because the storage requirements of BANDR are essentially identical to those of the BC and HYBBC implementations, the experimental analysis of Section 4.4 concentrates on the CPU requirements of each routine.

4.3.4 Rescaling and Fast Givens Transformations

To improve efficiency BANDR and our implementations of BC and HYBBC use fast Givens transformations in place of classical Givens transformations. As shown in Equation 2.7, each fast Givens transformation applied to the matrix under reduction must also update a diagonal matrix, D, associated with the reduction. At the end of the reduction D updates the tridiagonal matrix to complete the sequence of fast Givens transformations. (See Equation 2.8.)

If the fast Givens transformation $M^T A M$ modifies rows and columns i and j of A, the updated diagonal matrix, $\widehat{D}$, takes the following form. Assuming $D = \text{Diag}(d_1, \ldots, d_n)$,

$$\begin{aligned} \widehat{D} &= M^T D M \\ &= \text{Diag}\left(d_1, \ldots, d_{i-1}, d_i(1+\gamma), d_{i+1}, \ldots, d_{j-1}, d_j(1+\gamma), d_{j+1}, \ldots, d_n\right). \end{aligned} \tag{4.1}$$

As shown in Section 2.4.2, M can take one of two forms. By carefully selecting the appropriate fast Givens transformation type, we can ensure each transformation's "growth factor" $(1+\gamma)$ is bounded by 2. Of course, modifying one of D's diagonal entries by s transformations may result in its growth by a factor of 2^s. For reductions requiring large numbers of transformations, periodic rescaling of D is necessary to avoid overflow. The remainder

of this subsection provides a general outline of the rescaling techniques employed by the R-S, BC and HYBBC implementations. As discussed in Section 2.4.2, to permit direct comparison of our codes to BANDR we do not employ the recently developed fast plane rotations [AP94] that avoid explicit periodic rescaling. Implementing BC and HYBBC and algorithms of subsequent chapters using Anda and Park's fast plane rotations with dynamic scaling is an interesting task for future research.

All three implementations avoid difficulties with overflow by carefully restricting the worst case growth between rescaling episodes and by selecting an appropriate rescaling constant for entries that experience too much growth. During R-S's column-oriented reduction, monitoring the potential growth of D's entries is straightforward. When R-S eliminates one column from the band, at most two transformations modify each of D's diagonal entries. Thus if the maximum permissible growth between rescalings is 2^{64}, we should schedule rescaling episodes after the elimination of each block of 32 columns. Suppose D's entries must always remain less than 2^{128} to avoid overflow. If during each rescaling all entries $D_{i,i}$ with $|D_{i,i}| > 2^{64}$ are rescaled by 2^{64}, D's entries cannot overflow before the next rescaling.

The diagonally-oriented reduction of Bandwidth Contraction requires a complete reformulation of BANDR's rescaling techniques. To simplify the rescaling process we ignore band sparsity and assume BC eliminates each entry with a nontrivial transformation. If b^c is the current bandwidth, during the elimination of a contiguous block of b^c entries from the outermost nonzero diagonal, at most 2 transformations modify each diagonal entry of D. In a similar fashion to BANDR's techniques, monitoring the number of blocks of b^c entries eliminated permits appropriate scheduling of rescaling episodes. This monitoring procedure is complicated by the completion of a diagonal's reduction, but these aspects of the implementation will not be detailed at this time. It is important to note, however, that as the reduction contracts the band and b^c grows smaller, the frequency of rescaling episodes increases. As a result, BC typically requires more rescaling episodes than BANDR. This result is not surprising considering BC uses significantly more transformations.

The implementation of HYBBC inherits the rescaling techniques of its constituent algo-

rithms with one modification. Just before switching from BC to R-S, the hybrid algorithm applies an extra rescaling episode to ensure a successful transition between the two rescaling procedures.

For all three implementations the cost of monitoring D's growth and the scheduling of rescaling episodes is insignificant. The cost of the rescaling procedures is dominated by the actual rescaling itself. During a rescaling episode each implementation scans D and rescales those entries that could potentially overflow before the next rescaling. In addition, the corresponding row and column of the intermediate reduction matrix are also rescaled to maintain the integrity of the similarity reduction. Even though BC and HYBBC may require significantly more rescaling than BANDR, Section 4.4's extensive experimentation with practical sparse problems shows rescaling accounts for a very small portion of the total computational requirements of tridiagonalization. For three problems HYBBC spends between 1 and 2% of its total reduction time rescaling, but for the remaining 112 problems the relative cost of rescaling is less than 1%.

4.4 Experimentation

This section describes extensive experimentation with implementations of the Bandwidth Contraction and Hybrid Bandwidth Contraction algorithms. After briefly describing the testing environment and the selection of HYBBC's *threshold* value, we analyze test results comparing our implementations to EISPACK's BANDR. The section concludes by investigating the correlation between preorderings and tridiagonalization performance.

4.4.1 The Testing Environment

To compare the computational requirements of our implementations to BANDR (an R-S code), all three routines are applied to the 115 symmetric sparse problems in the Harwell–Boeing test suite of Section 2.2. Unless otherwise specified, each problem is preordered to reduce bandwidth using Lewis's implementation of GPS [Lew82].

All testing was conducted on a Sun SPARCstation 2 with 16 MBytes of main memory.

All routines were compiled by the standard Sun FORTRAN compiler with full object code optimization requested. The reported CPU second timings, produced using the system routine `etime`, include both the user and system time for tridiagonalization, excluding the preordering stage.

4.4.2 Threshold Selection for HYBBC

HYBBC uses a transition strategy that thresholds the number of nonzeros in the outermost diagonal of the current band. For our experimentation we want to identify a value of the constant *threshold* (see Figure 4.8) that provides acceptable transition bandwidths for a broad spectrum of problems. Our research of effective threshold selection techniques is inconclusive, however, and for these experiments the transition to a column-oriented tridiagonalization is made when the outermost subdiagonal is full. With specialized knowledge of a particular problem's sparsity structure other values of *threshold* may be preferred.

In our experience, *threshold* values less than 1.0 can improve the performance of HYBBC for some sparse problems. For example, with *threshold*=1.0 HYBBC chooses a transition bandwidth of 11 for PLAT362. As shown by Figure 4.9, this problem's tridiagonalization would profit from an earlier transition at bandwidth 24. (The flop requirements of Figure 4.9's reductions were provided by the symbolic reduction tool `Trisymb`.) Despite the discrepancy between these transition bandwidths, HYBBC still comes within 3.1% of the optimal reduction using a *threshold* of 1.0.

Even with *threshold* set to 1.0, the transition strategy may also direct HYBBC to switch to R-S when continued bandwidth contraction would benefit the reduction. For example, HYBBC makes an early transition for large 5-point problems. In some sense the reduction of 5-point problems provide a worst case example of an early reduction transition. With a standard lexicographic ordering, the outermost diagonal of a 5-point problem is dense, but the remainder of the band is zero except for the main diagonal and its adjacent subdiagonal. Consider HYBBC's reduction of the 5-point problem with $b = 30$ ($n = 900$). Despite a relatively sparse band, HYBBC immediately switches to R-S at bandwidth 30, independent

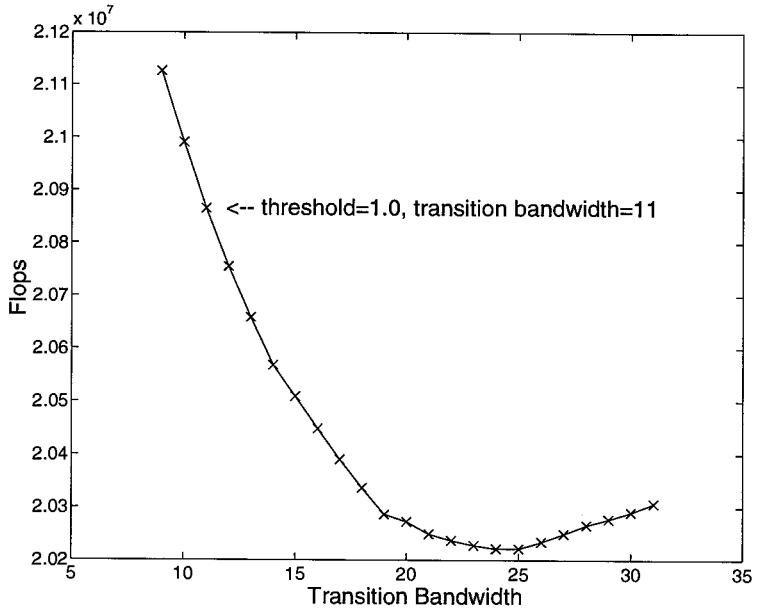


Figure 4.9: The Variance of PLAT362's Tridiagonalization Cost with Transition Bandwidth

of the value of *threshold*. As shown by Figure 4.10, the ideal transition bandwidth is 23. HYBBC's transition strategy, however, still allows the reduction to be within 3% of the optimal tridiagonalization.

These two examples demonstrate that HYBBC may make both an early transition when the band remains relatively sparse or a late transition when the outermost diagonal is not full but the band is relatively dense. Choosing a *threshold* of 1.0 seems to be an appropriate compromise. As described in Section 4.2.2, formal analysis of HYBBC's transition for general sparse problems is very difficult. Chapter 5 introduces a second hybrid tridiagonalization algorithm that lends itself to formal transition analysis, permitting the development of more precise transition strategies. Despite HYBBC's rudimentary transition strategy, this chapter's extensive experimentation shows that in the worst case HYBBC is almost always comparable to BANDR, but typically it is significantly more efficient.

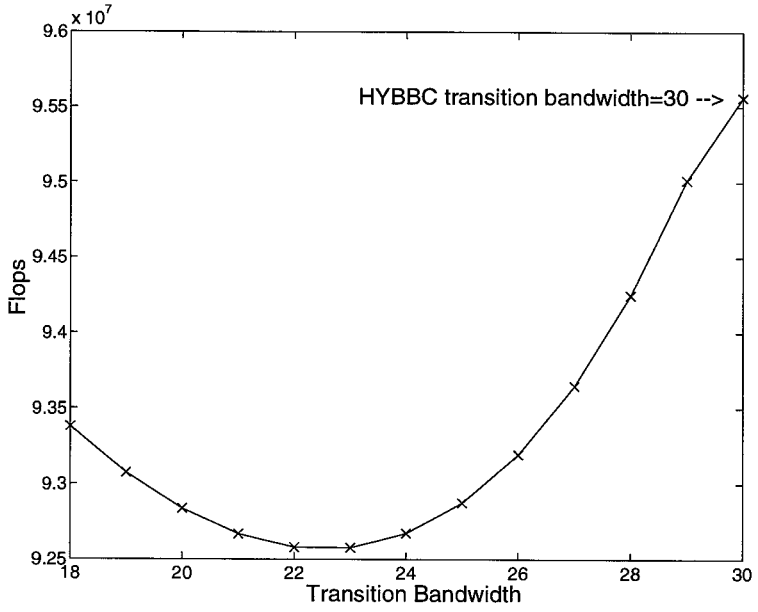


Figure 4.10: The Variance of Tridiagonalization Cost with Transition Bandwidth for a 5-Point Problem, $b = 30$

4.4.3 Numerical Testing Results

The implementations of BC and especially HYBBC are very successful relative to EISPACK's BANDR. For 98 of the 115 problems tested the hybrid tridiagonalization algorithm significantly reduces CPU requirements. For this group of problems, reductions in CPU time range from a low of 6.6% to a high of 63.3%. For the 70 test problems with more than 400 nodes, HYBBC requires on average 31.1% fewer CPU seconds than BANDR. The histogram in Figure 4.11 summarizes the distribution of CPU time reductions achieved by HYBBC for this group of 70 problems.

Table 4.4 summarizes the tridiagonalization of 20 test problems for which HYBBC is especially successful. The table provides tridiagonalization times for each implementation, with the CPU requirements of HYBBC's two reduction phases given in parentheses. For future reference the table also includes transformation totals. The two values in parentheses below each transformation total provide the number of bulge chasing transformations and row/column exchanges included in the total count. The symbols b^{GPS} and b^t refer to the

	Transformation Totals ($\times 10^3$) (bulge chasing, row/col. exch.)			Tridiagonalization Times (sec)			
Name n, b^{GPS}, b^t	BANDR	BC	HYBBC	BANDR	BC	HYBBC (BC, R-S)	%CPU Reduct.
685 BUS 685, 78, 3	226.2 (176.7,0)	633.6 (609.2,23.52)	594.7 (570.3,23.51)	67.7	26.4	26.0 (22.5,3.5)	61.6%
GR 30 30 900, 49,30	392.1 (350.2,0)	1235 (1203,25.84)	425.5 (394.1,24.76)	77.9	59.2	57.9 (4.1,53.8)	25.7%
NOS3 960, 65, 40	449.6 (390.3,0)	1531 (1486,23.20)	486.9 (442.5,22.52)	128.6	85.8	83.1 (5.9,77.2)	35.4%
DWT 1005 1005, 106, 5	487.9 (388.4,0)	1595 (1542,24.93)	1352 (1299,24.93)	209.1	88.7	86.8 (75.1,11.7)	58.5%
CAN 1054 1054, 112, 8	547.5 (436.9,0)	2039 (1965,13.86)	1573 (1500,13.85)	237.5	151.7	147.6 (128.1,19.5)	37.8%
CAN 1072 1072, 156, 8	567.1 (413.5,0)	2225 (2128,17.76)	1744 (1647,17.76)	331.9	193.7	189.2 (169.0,20.2)	43.0%
BCSSTK09 1083, 95, 57	567.6 (479.4,0)	2165 (2091,40.26)	642.1 (568.2,38.86)	234.3	158.5	150.8 (15.4,135.4)	35.7%
1138 BUS 1138, 126, 1	633.0 (499.1,0)	1991 (1925,73.73)	1991 (1925,73.73)	305.3	111.7	112.2 (112.2,0)	63.3%
ERIS1176 1176, 100, 1	643.3 (534.5,0)	2007 (1950,19.47)	2007 (1950,19.47)	291.3	118.9	120.1 (120.1,0)	58.8%
DWT 1242 1242, 91, 12	752.4 (645.0,0)	2533 (2469,38.96)	1623 (1560,38.92)	278.6	149.2	142.6 (101.5,41.1)	48.8%
BCSPWR07 1612, 103, 1	1271 (1113,0)	3998 (3910,79.14)	3998 (3910,79.14)	604.4	233.5	232.8 (232.8,0)	61.5%
BCSPWR09 1723, 116, 1	1451 (1261,0)	4771 (4665,117.8)	4771 (4665,117.8)	730.4	312.4	313.1 (313.1,0)	57.1%
PLAT1919 1919, 80, 10	1801 (1653,0)	6286 (6171,34.89)	4399 (4283,34.79)	706.7	440.0	424.6 (335.6,89.0)	39.9%
BCSSTK26 1922, 245, 18	1827 (1388,0)	7858 (7583,151.3)	5008 (4734,151.2)	1864	1018	1001 (839.5,161.6)	46.3%
DWT 2680 2680, 65, 6	3480 (3311,0)	11240 (11120,26.67)	9025 (8912,26.66)	1103	678.7	664.9 (548.3,116.6)	39.7%
ZENIOS 2873, 30, 1	5.283 (2.136,0)	5.532 (4.119,3.976)	5.532 (4.119,3.976)	1.25	0.81	0.86 (0.86,0)	31.2%
SSTMODEL 3345, 82, 1	3603 (3385,0)	11850 (11700,87.27)	11850 (11700,87.27)	1450	856.3	857.8 (857.8,0)	40.8%
LSHP3466 3466, 61, 3	5854 (5648,0)	20240 (20070,25.43)	19240 (19070,25.43)	1791	1448	1442 (1346,95.8)	19.5%
BCSSTK24 3562, 312, 42	6302 (5243,0)	29730 (29020,232.9)	14880 (14180,231.3)	8990	5417	5329 (4000,1329)	40.7%
BCSSTK28 4410, 323, 24	9646 (8280,0)	45730 (44710,603.3)	28100 (27080,603.3)	14643	9236	9118 (7916,1202)	37.7%

Table 4.4: Selected Tridiagonalization Summaries

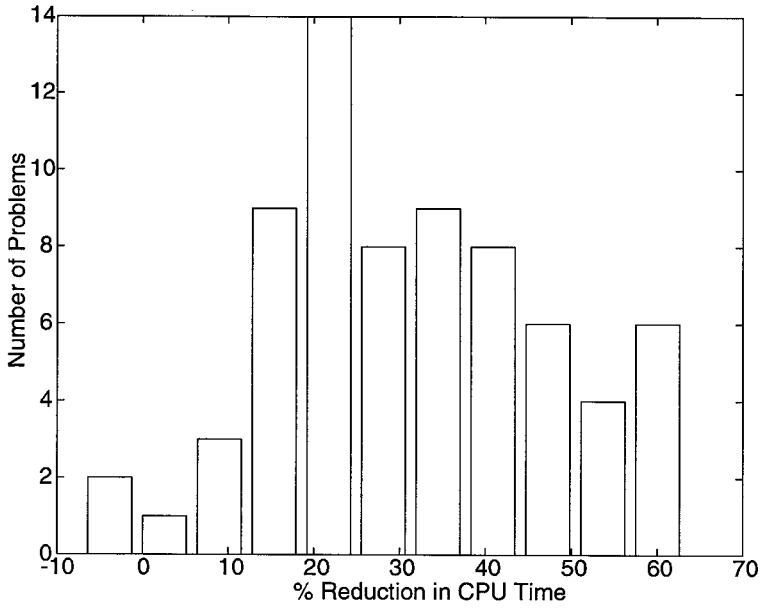


Figure 4.11: The Distribution of HYBBC's Improved Reduction Performance for Problems with $n > 400$

bandwidth of the GPS preordering and HYBBC's transition bandwidth respectively. The final column of the table reports the reduction in CPU time achieved by HYBBC relative to BANDR. For this group of problems HYBBC exhibits a mean reduction in CPU time of 44.2%.

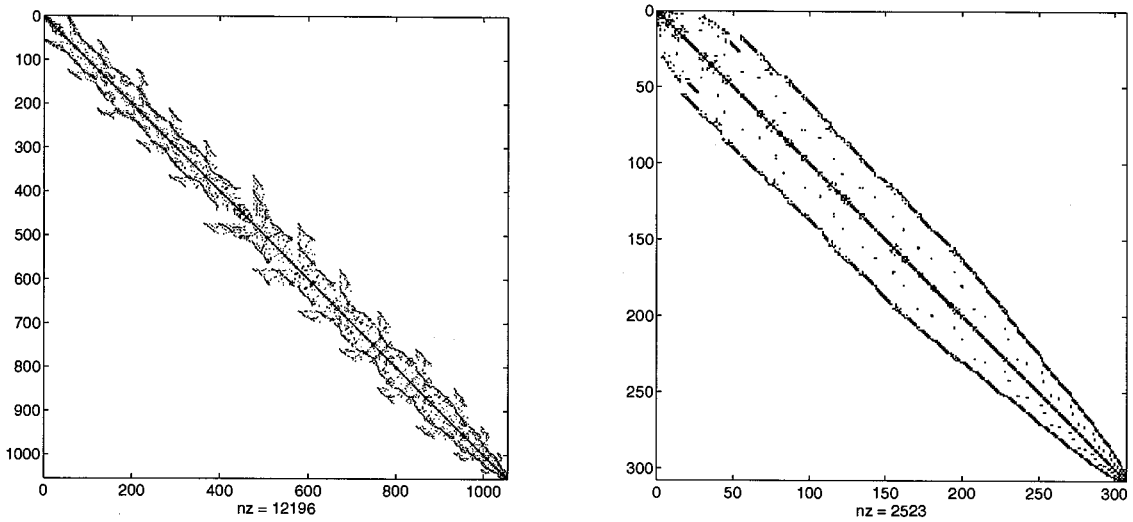
Of the 17 test problems for which HYBBC shows little or no improvement, 14 are very small problems, but 3 matrices have between 400 and 1000 nodes. As shown by Figure 4.11, HYBBC is actually slower than BANDR for 2 problems with $n > 400$. For one problem, DWT 992, HYBBC is only 1.3% slower than BANDR, but for NOS6 HYBBC requires approximately 7% additional CPU seconds. Of all 115 test problems, NOS6 is the only problem for which HYBBC significantly increases CPU requirements. The HYBBC reduction of a few very small matrices also shows a small increase in CPU requirements relative to BANDR. When the computational requirements of these reductions are checked with `Trisymb`, however, we find HYBBC requires fewer flops and timing discrepancies are due to the clock's granularity.

Name	n	b^{GPS}	Tridiagonalization Times (sec) BANDR	BC	HYBBC	% CPU Reduction BANDR→HYBBC
DWT..361	361	14	3.29	3.94	3.14	4.6

Table 4.5: DWT 361 Tridiagonalization Summary

Although the performance of BC is similar to HYBBC for most of the problems listed in Table 4.4, for some problems BC is substantially slower than BANDR. As an example, consider the tridiagonalization times of DWT 361 given in Table 4.5. BC requires significantly more CPU time than does BANDR. Such a performance degradation is understandable given the theoretical analysis of densely banded tridiagonalizations provided in Sections 3.4.1 and 4.2.1. Although initially 74% of the permuted problem's band entries are zero, BC's reduction of the first nonzero diagonal dramatically fills the band, resulting in a nonzero band density of .95. As a result, the band is dense for most of the reduction by the Bandwidth Contraction algorithm. In contrast, HYBBC only reduces 2 diagonals with BC before detecting the absence of band sparsity and switching to R-S to complete the reduction. Except for NOS6, the hybrid routine always has comparable CPU requirements to BANDR in the worst case; for the majority of sparse problems it is significantly faster.

For 40 of the 115 problems tested, HYBBC with a *threshold* of 1.0 uses the Bandwidth Contraction algorithm for the entire tridiagonalization process. The band of many of these problems becomes quite dense towards the end of the reduction and the tridiagonalization might have benefited by an earlier switch to BANDR. As an extreme example, once again consider the reduction of NOS6 by HYBBC. The initial steps of the Bandwidth Contraction algorithm reducing the bandwidth from 16 to 12 fills most of the permuted band's entries. Due to idiosyncrasies of the sparsity structure of NOS6, however, the outermost diagonal never completely fills. Consequently, HYBBC conducts the entire reduction using BC, even though the band is essentially dense for the majority of the reduction. These results confirm the observations of Section 4.4.2. Although HYBBC's transition regulating criterion works well for most problems, there is room for improvement using problem specific threshold



CAN 1054 DWT 307

Figure 4.12: Examples of Sparsity Structures Effecting Bandwidth Contraction Performance

values or employing more sophisticated transition strategies.

4.4.4 Sparsity Structures, Preorderings and Bandwidth Contraction Performance

As shown previously, the performance of BC and HYBBC is dependent on problem specific sparsity structures. In general, the class of sparsity structures ideally suited to Bandwidth Contraction concentrates nonzeros near the main diagonal and exhibit increased sparsity towards the outermost diagonals. We have seen that BC and HYBBC can dramatically reduce tridiagonalization requirements by exploiting sparsity away from the main diagonal. For example, Figure 4.12 plots the sparsity structure of CAN 1054 with a GPS preordering. HYBBC is able to exploit this problem's increased sparsity away from the main diagonal, and reduce tridiagonalization requirements by 37.8% relative to BANDR.

In contrast, sparsity structures that concentrate nonzeros in the band's outermost diagonals hamper the spike clipping techniques of Bandwidth Contraction. For this class of problems the outermost diagonals cannot be reduced at low cost and nontrivial transformations rapidly fill the band. The plot of sparse matrix DWT 307 in Figure 4.12 provides an

example of this sparsity structure class. The initial band nonzero density for this problem is a relatively sparse 9.24%. Unfortunately, the GPS preordering concentrates nonzeros in the band's outermost diagonals, making it difficult for HYBBC to exploit band sparsity. For this problem HYBBC shows a more modest reduction in CPU requirements of 10% relative to BANDR.

Of course, not all matrices can be permuted to simultaneously reduce bandwidth and produce sparsity structures conducive to Bandwidth Contraction. Whenever possible, however, the ideal preordering algorithm would concentrate sparsity near the main diagonal without sacrificing bandwidth. As a consequence of its preordering techniques using rooted level structures, GPS has a propensity to position nonzeros away from the main diagonal. Barnard et al [BPS93] demonstrate that spectral preorderings can exhibit complementary characteristics, concentrating nonzeros tightly about the main diagonal. Currently this preordering algorithm is designed to minimize matrix profile, not bandwidth. Pothen suggests, however, the possibility of using a local reordering strategy to refine preorderings for bandwidth reduction, without significantly altering the preordering characteristics beneficial to Bandwidth Contraction [Pot93]. Continuing the investigation of the relationship between this ordering scheme, and others, and Bandwidth Contraction performance will be an interesting avenue for future research.

Although HYBBC implements efficient techniques for clipping the longer spikes ex-

			Tridiagonalization Times (sec)			
			BANDR		HYBBC	
Name n	b^{GPS} (b^t)	b^{ND} (b^t)	GPS	ND	GPS	ND
BCSSTK09 1083	95 (57)	980 (33)	234.3	964.4	150.8	375.6
DWT 1007 1007	34 (12)	894 (22)	72.2	715.3	50.2	137.4
PLAT1919 1919	80 (10)	1891 (72)	706.7	6877.0	424.6	2729.0
BCSSTK27 1224	45 (25)	778 (31)	140.4	1145.7	113.7	264.6

Table 4.6: Sparse Tridiagonalization, GPS versus Nested Dissection

Preordering Method	b	BANDR Time	HYBBC Time
GPS	27	43.1	34.4
RCM	46	70.0	36.3
GK	40	59.8	34.7

Table 4.7: The Effects of a Poor Preordering on DWT 878

tending from the main diagonal, it is important to reiterate that the primary objective of preordering must be to reduce bandwidth. It is not beneficial to search for preorderings with long spikes or a wide variation in spike length as the first priority. For example, *nested dissection* [GL78a] permutations of sparse matrices often produce spikes of widely varying length, with the longest spikes towards the bottom of the matrix. But the bandwidth of such preorderings is much larger than for GPS. Although HYBBC takes good advantage of the increased sparsity away from the main diagonal for Nested Dissection preorderings, Table 4.6 demonstrates that it is much better to choose bandwidth reducing preorderings.

More direct examples of banded-like ordering are given by considering the tridiagonalization of DWT 878 using GPS, RCM [GL78b], and Gibbs-King (or GK) [Gib76, Lew82] preorderings summarized in Table 4.7. GK and RCM are likely to have longer spikes (higher bandwidth) but better profile than GPS. They are exactly the kinds of orderings one might use to get longer spikes. Switching from a GPS ordering to either RCM or GK, the CPU requirements of BANDR closely mirror the large increase in bandwidth. The "spike clipping" process of HYBBC, however, is able to efficiently exploit the increased band sparsity presented by RCM and GK, resulting in only marginal increases in CPU requirements.

Chapter 5

Split Bandwidth Contraction Algorithms for Sparse Symmetric Matrices

In Chapter 4 we demonstrated the ability of Bandwidth Contraction's diagonally-oriented reduction to exploit band sparsity and dramatically reduce tridiagonalization costs for a wide range of practical sparse problems. In addition, we combined BC with the Rutishauser-Schwarz algorithm to produce the versatile hybrid sparse tridiagonalization algorithm HYBBC. This hybrid algorithm exploits the differing reduction characteristics of BC and R-S to further improve the reduction of a typical sparse matrix and in the worst case guarantee reduction costs comparable to R-S. This chapter expands upon these successful techniques, developing second generation sparse algorithms for bandwidth contraction and tridiagonalization.

We begin with the development of the Split Bandwidth Contraction algorithm (or SBC), which enhances band sparsity exploitation using *bidirectional elimination* techniques. The novel Hybrid Split Bandwidth Contraction algorithm (or HYBSBC), described in the following section, incorporates many aspects of HYBBC's approach. To improve sparsity exploitation, however, HYBSBC replaces the BC stage with the Split Bandwidth Contraction algorithm. As a result, the new hybrid algorithm lends itself to formal transition analysis, permitting the development of the Δ-*transition strategy* for precise regulation of the algo-

rithm's transition bandwidth. The chapter's next section outlines implementations of both the SBC and HYBSBC algorithms. Finally, extensive experimentation demonstrates that SBC can dramatically reduce the cost of partial sparse bandwidth contractions, while HYBSBC substantially reduces sparse tridiagonalization costs relative to HYBBC and BANDR.

5.1 The Split Bandwidth Contraction Algorithm

For most sparse problems the requirements of bulge chasing transformations dominate the tridiagonalization costs of the Bandwidth Contraction algorithm. For example, on average 96.2% of the transformations employed by the 20 Bandwidth Contraction reductions in Table 4.4 eliminate bulge entries. This section introduces a novel band-preserving, diagonally-oriented reduction similar to BC. To improve reduction performance, however, the Split Bandwidth Contraction algorithm modifies the sequence of transformations used to eliminate each sparse diagonal. These *bidirectional elimination* techniques dramatically reduce bulge chasing requirements by taking additional advantage of band sparsity.

5.1.1 Row and Column-Oriented Adjacent Transformations

Two types of adjacent transformations are necessary for SBC's diagonally-oriented reduction. Consider the elimination of nonzero T from Figure 5.1's sparse matrix. Assuming the *zeroing entry* must be within the current band, only two adjacent transformations, $G_1(8,9,\theta_1)^T A G_1(8,9,\theta_1)$ and $G_2(5,6,\theta_2)^T A G_2(5,6,\theta_2)$, can be constructed for T's elimination. The zeroing entries for these transformations are marked by $\mathtt{Z}_1$ and $\mathtt{Z}_2$ respectively. We choose to classify both similarity transformations according to which of their rotations eliminates T from the the lower triangular portion of the matrix. The first transformation eliminates T from the lower triangle by modifying rows 8 and 9 with rotation $G_1^T(8,9,\theta_1)$ applied to the left of the matrix. Consequently, we classify this transformation as a *row-oriented* adjacent transformation. In contrast, the second transformation eliminates T from the lower triangular portion by applying $G_2(5,6,\theta_2)$ to the right of the matrix. Because this rotation modifies a pair of columns, we classify the second transformation as *column-*

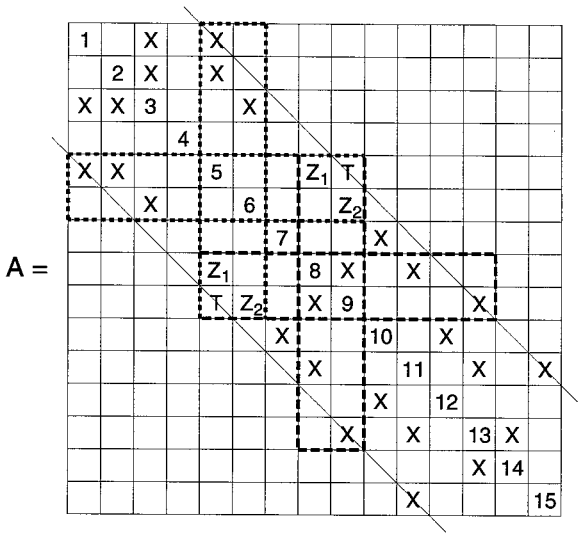


Figure 5.1: Row and Column-Oriented Transformations

oriented.

Applying either transformation to Figure 5.1's small example creates bulge entries. As expected, the bulge entry pair associated with the row-oriented transformation lies further down the band from T in positions $A_{13,8}$ and $A_{8,13}$. The bulges created by the column-oriented transformation, however, lie above T in positions $A_{6,1}$ and $A_{1,6}$.

5.1.2 Bidirectional Elimination

The Bandwidth Contraction algorithm eliminates each nonzero diagonal from top to bottom using row-oriented adjacent transformations exclusively. Alternatively, we propose starting the reduction in the middle of each diagonal, eliminating nonzeros in both directions, as depicted by Figure 5.2, using both nontrivial Givens transformations and row/column exchanges. This bidirectional elimination annihilates band nonzeros below the midpoint with row-oriented transformations and chases their bulge entries off the end of the matrix in the usual fashion. To eliminate the nonzeros of the diagonal's top half, however, a bidirectional elimination employs column-oriented transformations. The bulges created by these eliminations are chased off the top of the matrix with additional column-oriented transformations.

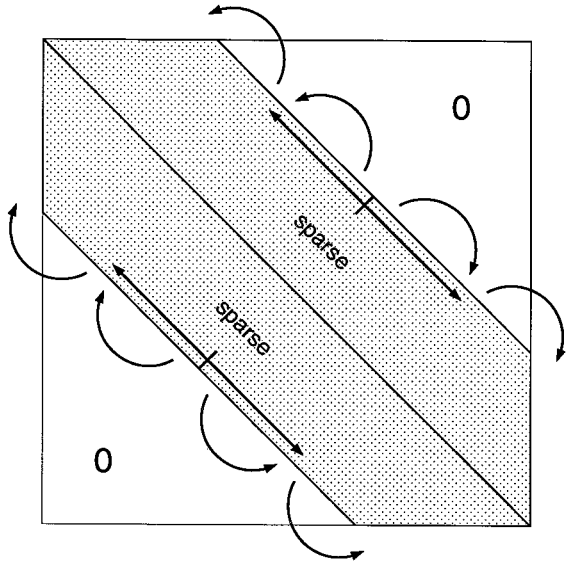


Figure 5.2: Bidirectional Elimination

As a result, bulge chasing sequences associated with the nonzeros above the diagonal's midpoint are significantly shortened. In fact, relative to BC these techniques potentially halve the total number of bulge chasing transformations required for each diagonal's reduction.

Unfortunately, cyclic elimination dependencies prevent the use of bidirectional elimination techniques for dense diagonals. Rather than develop a formal proof of this intuitive result, we manipulate the fragment of a lower triangular dense diagonal in Figure 5.3 to demonstrate the insurmountable difficulties associated with initiating its bidirectional elimination. For this example we assume the exclusive use of nontrivial adjacent transformations, but similar demonstrations permitting row/column exchanges reach the same conclusions. Without loss of generality suppose the reduction begins above the midpoint by eliminating T with a column-oriented transformation. This elimination creates a bulge as illustrated by the second band fragment in Figure 5.3. At this point the reduction could proceed in one of two ways. The elimination of the dense diagonal could continue above the midpoint, but first we must eliminate the bulge to avoid its growth. As shown in Figure 5.3, however, zeroing B with either a column or row-oriented transformation reestablishes the diagonal's

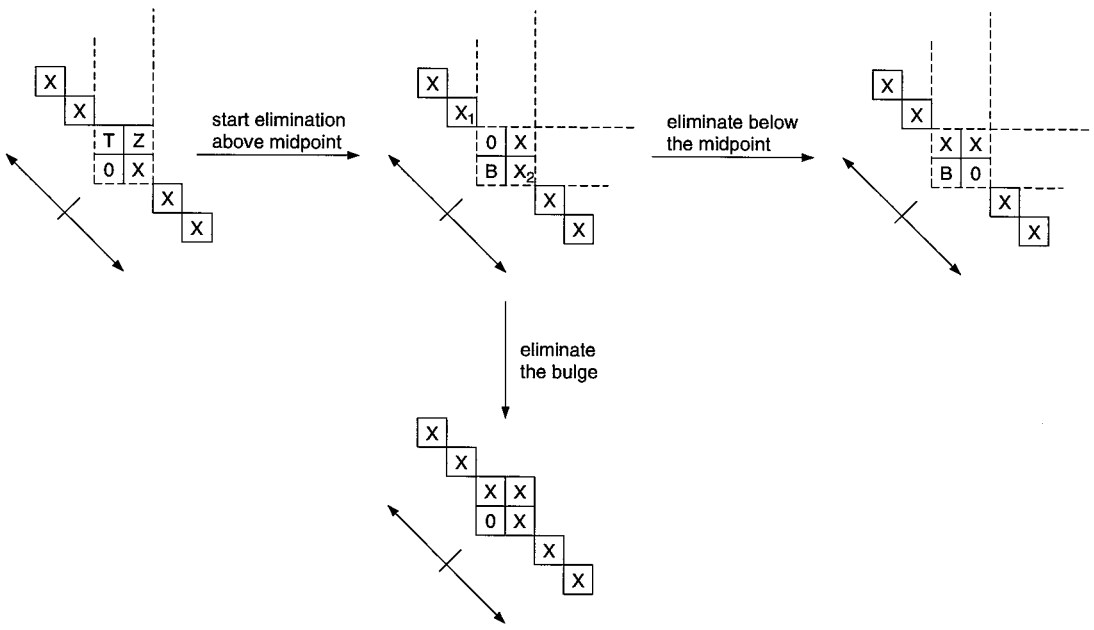


Figure 5.3: Cyclic Dependencies of a Dense Diagonal

original nonzero structure. Alternatively, we could attempt to initiate the elimination below the midpoint without removing the bulge. Although the row-oriented transformation eliminating X_2 does not cause bulge growth, it recreates a nonzero in the previously zeroed entry above the midpoint. Continuing the reduction below the midpoint requires the elimination of the bulge, which once again reproduces the diagonal's original sparsity structure. Although this is not an exhaustive example, it turns out there is no practical technique for starting a bidirectional elimination that circumvents the cyclic dependencies of a dense diagonal.

The bidirectional elimination of a sparse diagonal does not experience cyclic dependencies if the reduction appropriately exploits the diagonal's zero entries. The key to a successful bidirectional elimination is a region of a sparse diagonal we refer to as a *split-point*. A split-point is a block of 1 or more zero entries in the outermost nonzero diagonal of the current band. If a bidirectional elimination begins above and below a split-point, the diagonal's reduction can proceed in both directions with complete independence. The zero

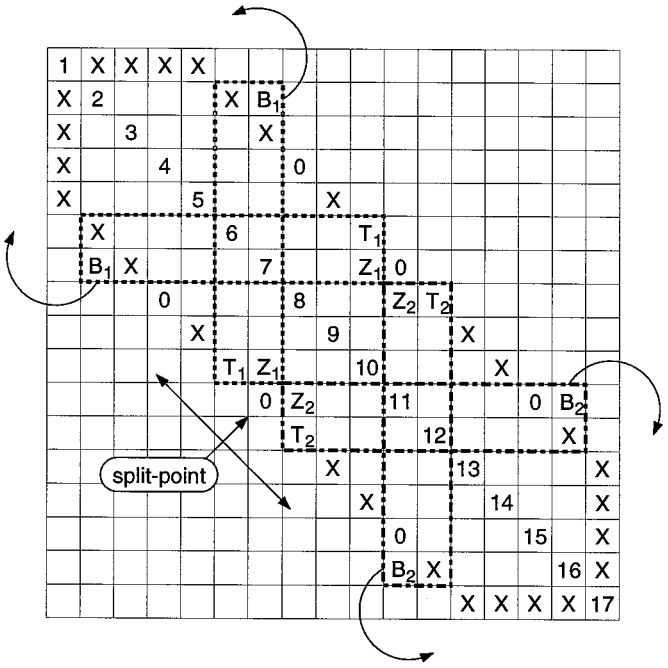


Figure 5.4: Split-points and Independent Transformations

entries of the selected split-point prevent the inappropriately positioned bulges that created the cyclic reduction dependencies for a dense diagonal. In addition, transformations applied above and below the split-point modify independent subsets of band entries. For example, in Figure 5.4 there is no overlap between the dashed rectangles outlining the extent of the row and column-oriented transformations eliminating T_2 and T_1.

When a sparse diagonal contains a single contiguous block of zero entries the split-point is fixed. Other sparse diagonals may have many split-points to choose from. The optimum split-point that minimizes their reduction costs is dependent on problem specific sparsity structures and may be difficult to determine. Section 5.1.4 explores four split-point selection strategies that attempt to minimize the reduction costs associated with a sequence of bidirectional eliminations.

5.1.3 A Formal Presentation of SBC

The Split Bandwidth Contraction algorithm is similar in many respects to BC and inherits all the advantages enjoyed by its predecessor for a sparse band. To exploit band nonzeros, for example, SBC incorporates the three techniques for exploiting band sparsity included in Section 4.1.2's description of BC. As detailed in Figure 5.5, however, SBC also employs the bidirectional elimination techniques of Section 5.1.2 to take additional advantage of band sparsity and reduce the bulge chasing requirements of each diagonal's reduction.

Once again, SBC begins by symmetrically permuting the sparse matrix to reduce bandwidth. Beginning with the outermost nonzero diagonal of the lower triangular portion of the band, SBC then searches for an appropriate split-point with the routine *find_split-point.* If the outermost diagonal is dense this routine returns 0, causing SBC to skip step 3b and revert to a standard BC reduction. Otherwise *find_split-point* returns the column index of a zero in the selected block of zero entries. The specific split-point selection algorithm used by this routine is detailed in Section 5.1.4, which explores split-point selection strategies that attempt to minimize reduction costs.

With the split-point assigned, SBC scans the diagonal for its first nonzero entry above the split-point. This entry is eliminated using an adjacent column-oriented row/column exchange or nontrivial transformation, depending upon the nonzero status of the zeroing entry $A_{col+b^c,col+1}$. If this transformation creates a nonzero beyond the limits of the current band, the bulge is chased off the top of the matrix with additional column-oriented transformations, as described in Section 5.1.2. The scanning and reduction process continues until the portion of the diagonal above the split-point has been eliminated. The algorithm then switches to a more familiar diagonally-oriented reduction using row-oriented transformations to eliminate nonzeros below the split-point. Upon completion of the diagonal's elimination, SBC decrements the current bandwidth, b^c, and begins the bidirectional elimination of the next diagonal. As defined in Figure 5.5, SBC is a tridiagonalization algorithm. Although it efficiently tridiagonalizes sparse matrices, SBC also effectively exploits band sparsity while performing partial bandwidth contractions. The final bandwidth of SBC's

1. $A := P^T AP$, where P is a bandwidth reducing permutation matrix.
2. $b :=$ bandwidth(A)
3. FOR $b^c := b$ DOWNTO 2 DO /*Tridiagonalize A.*/
   (a) $sp =$ *find_split-point*(A, b^c)
   (b) FOR $col := sp - 1$ DOWNTO 1 DO /*Eliminate above the split-point*/
       IF $A_{col+b^c,col} \neq 0$ THEN /*Zero $A_{col+b^c,col}$.*/
           IF $A_{col+b^c,col+1} = 0$ THEN
               Exchange rows/columns (col) and $(col+1)$ in A.
           ELSE
               $A := G(col, col+1, \theta)^T \; A \; G(col, col+1, \theta)$
           IF bandwidth$(A) > b^c$ THEN
               Chase bulges with additional column-oriented adjacent
               Givens transformations or row/column exchanges.
       ENDIF /*Outermost IF*/
   (c) FOR $col := sp + 1$ TO $n - b^c$ DO /*Eliminate below the split-point*/
       IF $A_{col+b^c,col} \neq 0$ THEN /*Zero $A_{col+b^c,col}$.*/
           IF $A_{col+b^c-1,col} = 0$ THEN
               Exchange rows/columns $(col + b^c)$ and $(col + b^c - 1)$ in A.
           ELSE
               $A := G(col + b^c, col + b^c - 1, \theta)^T \; A \; G(col + b^c, col + b^c - 1, \theta)$
           IF bandwidth$(A) > b^c$ THEN
               Chase bulges with additional row-oriented adjacent
               Givens transformations or row/column exchanges.
       ENDIF /*Outermost IF*/

Figure 5.5: The Split Bandwidth Contraction Algorithm

	$F^{\text{FG}}_{\text{SBC1}}$	T_{SBC1}
if $1 \leq m \leq b$	$(4+\frac{10}{b})n^2 - (\frac{20m}{b} + 6b + 8m + 10)n$ $+ (10 + \frac{20}{b})m^2 + (6b - \frac{20}{b} + 8)m + 2b^2 + 3b + \frac{10}{b} - 1$ $+ 2C_{SBC}(5 + 2b - C_{SBC}(2 + \frac{5}{b}))$ $+ 2\widehat{C}_{SBC}(5 + b - \widehat{C}_{SBC}(1 + \frac{5}{b}))$	$\frac{n^2}{2b} - (\frac{1}{2} + \frac{m}{b})n + \frac{m^2}{b}$ $+ (1 - \frac{1}{b})m + \frac{1}{2b} - \frac{1}{2}$ $+ \frac{C_{SBC}}{2}(1 - \frac{C_{SBC}}{b})$ $+ \frac{\widehat{C}_{SBC}}{2}(1 - \frac{\widehat{C}_{SBC}}{b})$
if $b < m \leq \lceil \frac{n-b}{2} \rceil$	$(4+\frac{10}{b})n^2 - (\frac{20m}{b} + 6b + 8m + 10)n$ $+ (8 + \frac{20}{b})m^2 + (8b - \frac{20}{b} + 12)m + 2b^2 + b + \frac{10}{b} - 3$ $+ (\frac{10}{b} + 2)(C_{SBC}(b - C_{SBC}) + \widehat{C}_{SBC}(b - \widehat{C}_{SBC}))$	Unchanged

Table 5.1: The Cost of SBC Eliminating the Outermost Nonzero Diagonal of a Banded Model Problem

reduction can be freely selected from the range $1 \leq b^f < b$.

Formal analysis of the SBC algorithm is confronted by the same difficulties as an analysis of Bandwidth Contraction when considering the reduction of matrices with general symmetric sparsity patterns. We can obtain useful complexity results, however, for a special family of bandwidth b, order n symmetric problems. The bands of the problems are densely populated except for a single zero entry, a split-point, in both the upper and lower outermost nonzero diagonals of the band. The third parameter defining each matrix in this family of model problems is the column index, m, of the zero in the lower triangular portion of the band.

After SBC reduces the outermost diagonal of a problem in this class, the contracted band is completely dense and inaccessible to bidirectional elimination techniques. The elimination of this single diagonal, however, provides a worst case analysis for SBC's reduction of the outermost diagonal of identically sized sparse problems using a split-point in column m. Table 5.1 provides flop and transformation count formulas for this reduction, $F^{\text{FG}}_{\text{SBC1}}$ and T_{SBC1} computed using the enhanced analysis framework of Section 2.4.3. Without loss of generality, the analysis assumes $1 \leq m \leq \lceil \frac{n-b}{2} \rceil$. To apply the analysis to a model problem with its split-point in the diagonal's bottom half, simply consider the computationally equivalent problem found by reflecting the split-point's position about the diagonal's midpoint. Due to difficulties with operation count summations, we actually performed one

F_{BC1}^{FG}	T_{BC1}
$(4+\frac{10}{b})n^2-(6b+10)n+2b^2+3b$ $+2Mod(n,b)(1+\frac{5}{b})(b-Mod(n,b))$	$\left(\frac{1}{2b}\right)(n(n-b)+Mod(n,b)(b-Mod(n,b)))$

Table 5.2: The Cost of BC Eliminating the Outermost Nonzero Diagonal of a Densely Banded Problem

analysis when the split-point is in the first b columns of the diagonal and a separate analysis when $b < m \leq \lceil \frac{n-b}{2} \rceil$. In addition, the analysis assumes $b < n/3$ and that an equal proportion of the two types of nontrivial fast Givens transformations are employed. The analysis also ignores the potential cost of the periodic rescaling required by fast Givens transformations. Finally, C_{SBC1} and $\widehat{C}_{SBC1}$ are the nonanalytic terms $Mod(m-1,b)$ and $Mod(n-m,b)$.

For comparison Table 5.2 provides flop and transformation counts for the reduction of one diagonal of a densely banded symmetric matrix using the Bandwidth Contraction algorithm. (These results were extracted from BC's tridiagonalization analysis detailed in [Cav93].) It is difficult to compare these results to the complicated formulas for $F_{\mathrm{SBC1}}^{\mathrm{FG}}$ and T_{SBC1}. Alternatively, the graphs in Figure 5.6 plot SBC's flop and transformation counts, normalized by F_{BC1}^{FG} and T_{BC1}, against m for a model problem with $n = 1000$ and $b = 100$. When the split-point is close to the center of the diagonal (m=450) SBC dramatically reduces computational requirements and, as expected, BC requires almost twice as many flops and transformations. As the split-point is moved towards the top of the diagonal, however, the relative advantage of SBC gradually diminishes and eventually the computational requirements of SBC converge to those of BC. For a dense diagonal the SBC and BC reductions are identical. The general trends exhibited by these plots are largely insensitive to changes in the relative size of n and b.

5.1.4 Split-Point Selection Strategies

The bidirectional elimination of a sparse diagonal may have many alternative split-points to choose from. Surprisingly, it is often difficult to choose the split-point that minimizes

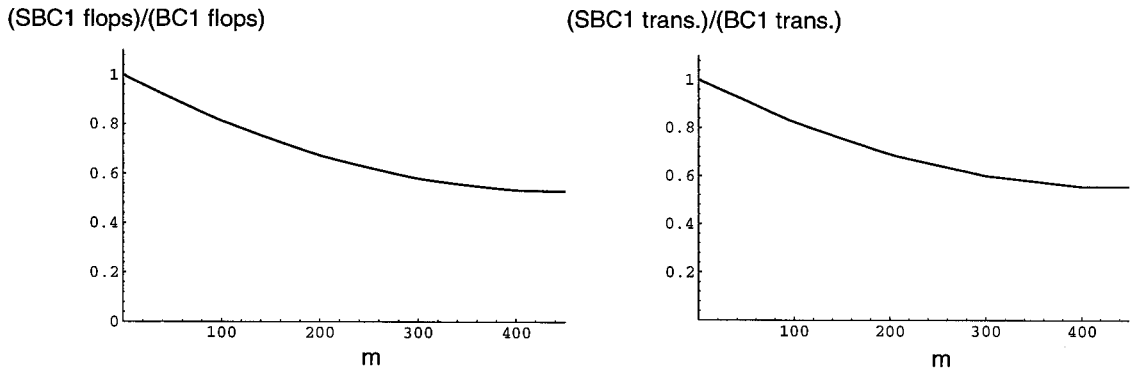


Figure 5.6: The Flop and Transformation Requirements of SBC relative to BC for n=1000 and b=100.

the cost of reducing a sparse problem's outermost nonzero diagonal. A simple selection strategy chooses the split-point with minimum *displacement*. A split-point's displacement is defined as $|mid - sp|$, where sp and mid are the column indexes of a split-point and the diagonal's midpoint respectively. As shown in the previous subsection, the bidirectional reduction costs of a diagonal with a single zero entry are minimal when the split-point is at the diagonal's midpoint. For diagonals with multiple split-points, however, choosing the split-point with minimal displacement may not always be optimal. For example, the outermost diagonal in the band of the small sparse problem in Figure 5.7 has two split-points. The minimum displacement selection strategy chooses the centered split-point in column 9. With this split-point the diagonal's bidirectional elimination uses 11 nontrivial fast Givens transformations, requiring 553 flops. In contrast, the bidirectional elimination using the split-point in column 4 requires 7 transformations and 345 flops.

In general a diagonal's optimum split-point depends on several contributing factors.

1. The distribution of nonzeros in the diagonal.
2. The distribution of nonzeros in the remainder of the band.
3. Problem specific sparsity structures of the sparse band.

Unfortunately, these factors interact in a very complex manner, and without potentially

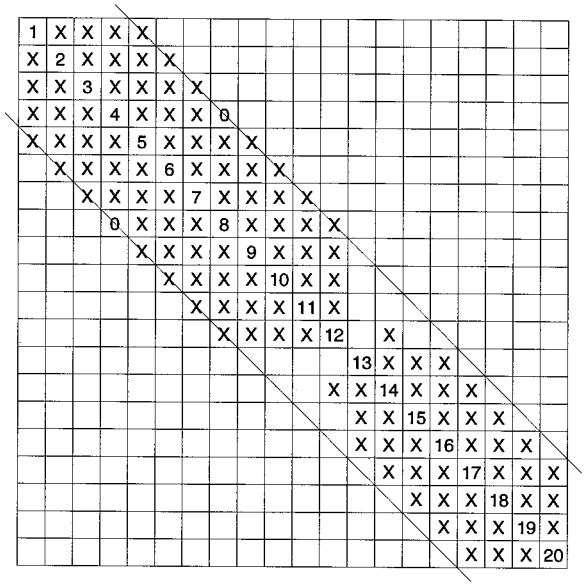


Figure 5.7: Reduction Performance and Split-point Selection

expensive modeling of a diagonal's reduction, it is difficult to consistently improve upon the minimum displacement approach. For example, without modeling it is generally not known to what extent a diagonal will fill during its reduction. Choosing the minimum displacement split-point, however, minimizes reduction costs in the worst case when the diagonal under reduction experiences high levels of fill. In our experience Figure 5.7's example is an exceptional case. The minimum displacement split-point selection strategy is generally applicable and typically finds the optimal or near optimal split-point for a single diagonal's reduction.

The Split Bandwidth Contraction algorithm, of course, typically reduces a sequence of nonzero diagonals. In this case the split-point selected for one diagonal's elimination may impact the quality of split-points available for the reduction of subsequent diagonals. To perform an efficient bandwidth contraction, we seek a sequence of split-points that globally minimizes reduction costs. Many different factors affect the sparsity of reduction intermediates and the location of their split-points. The remainder of this subsection briefly investigates four global selection strategies, which explore trade-offs between the displace-

Problem	n	b^{GPS}	Problem	n	b^{GPS}	Problem	n	b^{GPS}
DWT 361	361	14	CAN 1054	1054	112	BCSSTK12	1473	62
LSHP 577	577	25	BCSSTK09	1083	95	BCSPWR08	1624	108
GR 30 30	900	49	1138 BUS	1138	126	PLAT1919	1919	80
NOS3	960	65	ERIS1176	1176	100	SSTMODEL	3345	82
DWT 1007	1007	34	DWT 1242	1242	91	BCSSTK28	4410	323

Table 5.3: Test Problems for Splitpoint Selection Experimentation

ment of selected split-points, the position of future split-points, fill, and reduction costs. Because it is difficult to theoretically determine the relative importance of factors affecting contraction performance, we evaluate the merits of each approach using `Trisymb` experimentation with the subset of 15 problems from the Harwell–Boeing test suite listed in Table 5.3.

5.1.4.1 Minimum Displacement Split-Point Selection

Our first global split-point selection strategy simply applies the minimum displacement selection criterion to each intermediate matrix, relying upon its efficient elimination of each diagonal to approximate a global minimization of reduction costs. As a typical split bandwidth contraction proceeds, a block of nonzeros builds in the center of the outermost diagonal of successive intermediate matrices, forcing split-points away from the midpoint. For the moment when the two split-points bordering this central nonzero block have equal (minimal) displacement, the selection strategy breaks ties arbitrarily. We use this version of the Split Bandwidth Contraction algorithm, referred to simply as SBC, to benchmark each of the three remaining global selection strategies.

5.1.4.2 Unidirectional Split-Point Selection

While experimenting with a `Trisymb` implementation of SBC, we observed that a few problems experience higher levels of band fill with SBC than BC. The largest discrepancies in the band density of corresponding intermediate matrices are often associated with ex-

tended periods of *split-point flipping*. During an episode of split-point flipping SBC selects split-points from alternating sides of the growing block of nonzeros straddling the midpoint of the outermost diagonal of successive intermediate matrices. Without going into great detail, split-point flipping can increase intermediate fill levels by

1. changing the orientation of transformations eliminating nonzeros from particular regions of the band.
2. modifying the order in which the outermost diagonal's entries are eliminated.

If we curtail split-point flipping, perhaps fill could be reduced and reduction performance improved. To investigate the relationship between split-point flipping, fill and reduction performance, we propose a second global strategy, the *unidirectional* split-point selection strategy. (The corresponding version of the reduction algorithm is referred to as USBC.)

The unidirectional selection strategy chooses minimal displacement split-points subject to the following constraint. After selecting its first off-centre minimum displacement split-point, the strategy must choose split-points from the same side of the midpoint for the reduction of subsequent diagonals. When the selection process encounters a diagonal without a split-point in this region, it may consider split-points from the remainder of the diagonal.

To compare the relative merits of USBC and SBC, we performed partial bandwidth contractions for each of the 15 problems in Table 5.3, using `Trisymb` symbolic implementations of both algorithm variants. In isolation from other reduction factors, eliminating split-point flipping may reduce fill production, but the unidirectional selection strategy typically forces USBC to select split-points of significantly higher displacement, increasing bulge chasing requirements. The fill accompanying these extra transformations typically neutralizes the potential band fill savings of unidirectional split-point selection. Consequently, USBC does not reduce the contraction costs of any problem relative to SBC. In fact, for 5 problems: CAN 1054, DWT 1242, GR 30 30, SSTMODEL and 1138 BUS, USBC increases flop requirements by 11–76%. For these problems USBC frequently chooses split-points with 3 to 4 times the displacement of SBC's split-point selection for the corresponding reduction inter-

mediate. Because fill and band nonzeros interior to the band of the original matrix reduce the sparsity of the contracted band's outermost diagonal, there is little chance of USBC exhausting the split-points on one side of the midpoint and finding small displacement split-points remaining in the other half of the diagonal. In conclusion, these experiments clearly demonstrate that selecting minimum displacement split-points is more cost effective than attempting to avoid flipping with a unidirectional scheme.

5.1.4.3 Damped Split-Point Selection

The difficulties of the unidirectional split-point selection strategy result from dramatic increases in split-point displacement. The unidirectional strategy, however, is one extreme of a range of selection strategies. This section reexamines the trade-offs between fill, flipping and split-point displacement using a generalization of the unidirectional strategy that *damps* the frequency of split-point flipping incidents to varying degrees. With this scheme we seek a compromise between the unidirectional and minimum displacement split-point selection strategies that improves reduction performance.

The *damped* split-point selection strategy controls the selection of each diagonal's split-point using the hysteretic process defined by the following conditional statement. (The corresponding version of the Split Bandwidth Contraction algorithm is referred to as DSBC.)

IF $(DF * |D_{SS}|) \leq |D_{OS}|$ THEN
    select split-point SS
ELSE
    select split-point OS

SS and OS are the minimum displacement split-points on the same side and opposite side of the midpoint as the split-point chosen for the previous diagonal's reduction. The displacement of split-points SS and OS is given by D_{SS} and D_{OS}. The roles of *opposite* and *same* side split-point are undefined when

- SBC is reducing the outermost diagonal of the original band.
- a centred split-point is available in the current diagonal.

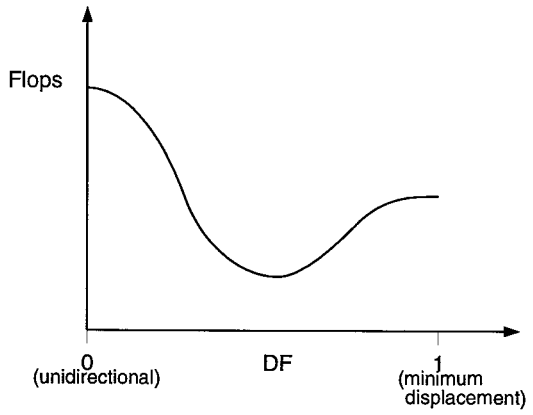


Figure 5.8: Desirable DSBC Reduction Characteristics

- the previous diagonal was eliminated using a centred split-point.
- split-points are unavailable in one half of the current diagonal.

In these special cases the diagonal's minimum displacement split-point is chosen.

The *damping factor*, DF, regulates the amount by which the displacement of the selected split-point may exceed the minimum displacement to avoid flipping. ($0 \leq DF \leq 1$) For example, if $DF = 0.5$ then $|D_{SS}|$ must be more than twice $|D_{OS}|$ before split-point OS is selected. When $DF = 0$ the damped split-point selection strategy degenerates into the unidirectional scheme, while at $DF = 1$ it is equivalent to the minimum displacement strategy.

Once again, we explored the relative merits of DSBC by performing partial bandwidth contractions of Table 5.3's 15 sparse problems with a `Trisymb` symbolic implementation of the algorithm. The goal of the DSBC algorithm is to damp split-point flipping, reducing or delaying the introduction of band fill to improve contraction efficiency. Although the unidirectional selection strategy increases flop requirements, we pursue the possibility that partially damping flipping results in flop savings, as depicted in Figure 5.8. Unfortunately, extensive experimentation did not reveal any consistent improvements in contraction performance independent of the damping factor. In fact, the contractions of

most problems were unchanged or showed slight increases in flops for moderate damping factors ($0.5 \leq DF \leq 1.0$) relative to SBC. Further reduction of the damping factor causes significant increases in flop requirements for the four problems on which USBC performed the most poorly, while the remaining problems were largely unaffected. No problem exhibits a trend towards reduced contraction costs. In conclusion, these experiments clearly demonstrate that amongst the split-point selection alternatives presented by DSBC, choosing split-points with minimum displacement must be the paramount objective.

5.1.4.4 Induced Split-Point Selection

As discussed earlier in Section 5.1, to improve the efficiency of sparse bandwidth contraction, it is important to reduce bulge chasing requirements. The damped split-point selection strategy tried to reduce bulge chasing by restricting split-point flipping to control band fill. Alternatively, *induced split-point selection* techniques ignore potential increases in band density and focus directly upon the development and use of small displacement split-points. In fact, the induced split-point selection strategy encourages *split-point flipping* in the hope of producing a set of smaller displacement split-points for the reduction of subsequent diagonals.

There are several factors motivating the investigation of induced split-point selection techniques. First, we hope this scheme will distribute bulge chasing more evenly, encouraging a balanced distribution of nonzero entries and future split-points across the midpoints of a band's outermost diagonals. As observed for applications of the damped split-point selection strategy, unevenly distributed split-points can result in unidirectional split-point selection behavior.

In addition, while the bands of intermediate matrices remain relatively sparse, split-point flipping can reduce the displacement of split-points available for the reduction of subsequent diagonals. Consider the reduction of the outermost nonzero diagonal from a sparsely banded matrix of bandwidth b, and for the moment ignore the original nonzero structure of diagonal $b-1$. As diagonal b is reduced using adjacent nontrivial transforma-

tions and row/column exchanges its nonzero entries, and possibly split-points, are generally mapped into neighboring positions in column $b-1$. In special circumstances split-points available in diagonal $b-1$ will have a smaller displacement than their ancestors in diagonal b. For example, an unused split-point in the current diagonal may also exist in the next diagonal, but with a smaller displacement, if a split-point from the opposite side of the midpoint is chosen for the current diagonal's reduction. The following discussion outlines several circumstances under which split-points are shifted by SBC's elimination of one diagonal.

As for previous selection strategies, we assume the split-point selected for each diagonal's reduction is best chosen from the pair of minimal displacement split-points straddling the diagonal's midpoint. We refer to this pair of split-points as:

SPA - The smallest displacement split-point above the midpoint of the outermost diagonal.

SPB - The smallest displacement split-point below the midpoint of the outermost diagonal.

The evolution of SPA and SPB's displacements is closely related to the following problem.

- As SBC contracts the bandwidth of a sparse matrix, does the length of the contiguous nonzero block encompassing the midpoint of the outermost diagonal grow monotonically?

Although growth in the central block of nonzeros is a dominant reduction trend, there are special situations in which the nonzero block can shrink slightly. Figure 5.9 illustrates a small example in which reducing the bandwidth from 3 to 2 with SBC decreases the length of the central nonzero block by one and moves SPA closer to the midpoint.

In this example, the reduced length of the nonzero block, and part of SPA's reduced displacement, results from the elimination of the second band nonzero above the split-point with a row/column exchange. As illustrated in Figure 5.10, application of this transformation switches diagonal 3's fourth nonzero above the split-point into position $A_{5,3}$, thus avoiding its own explicit reduction and separating it from what remains of the central nonzero block.

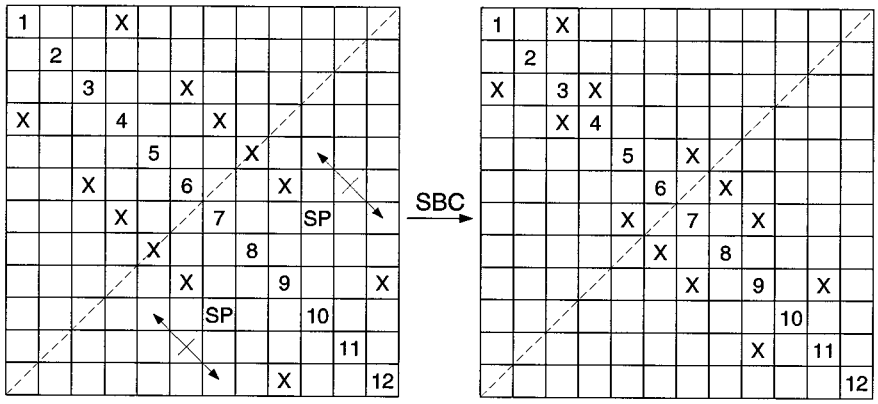


Figure 5.9: Shrinking the Central Nonzero Block

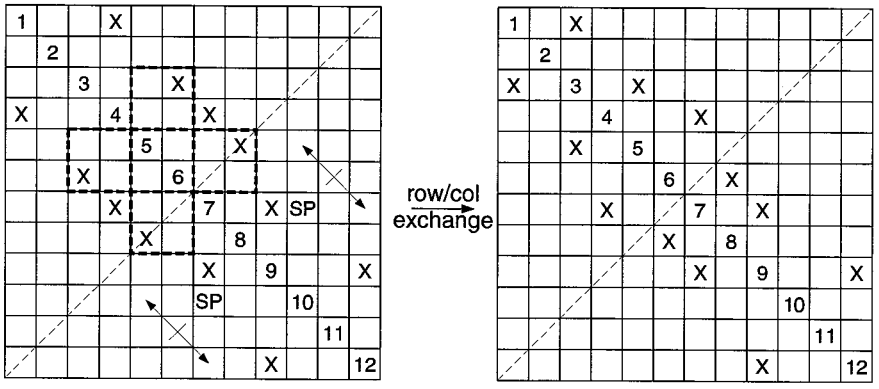


Figure 5.10: Shifting a Split-Point with a Band Elimination

Let *NBL* be the length of the central nonzero block in the outermost diagonal of a bandwidth b sparse matrix. In general, when $NBL < b$ there is no opportunity of reducing a split-point's displacement in this fashion. However, if $NBL \geq b$ then once the central block has been reduced to length b by the diagonal's partial reduction, a split-point shift can occur if the next nonzero in the central block is reduced with a row/column exchange. Of course, these conditions are not sufficient to guarantee a split-point shift. Fill entries or nonzero entries already existing in the $(b-1)^{th}$ subdiagonal could disrupt the shifting process.

The previous discussion describes why SPA moves one row towards the midpoint in Figure 5.9, but what about the other half row of the reported shift? Consider a more general case in which SPB is used to reduce the current diagonal and SPA is at least two zero entries in length. In this case if SPA is not destroyed by fill entries, or existing nonzeros in the neighboring subdiagonal, it will move at least 0.5 rows closer to the midpoint in adjacent diagonal, as shown in Figure 5.11, independent of the type of column-oriented transformations employed. Given this result it might be tempting to choose a series of split-

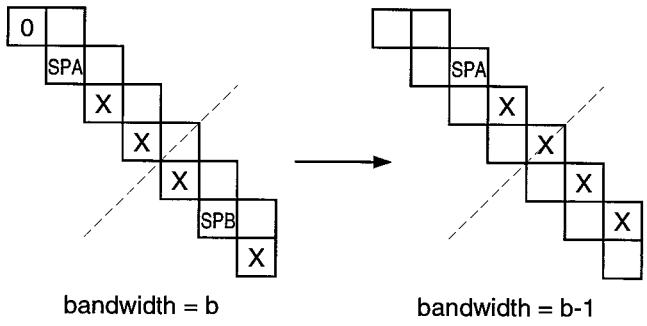


Figure 5.11: The Normal Reduction Shift of a Lower Triangular Split-Point

points from one side of center to move split-points on the other side closer to the center. As shown in Sections 5.1.4.2 and 5.1.4.3, however, split-point selection schemes with a strong unidirectional flavor are unsuccessful.

When $NBL > b$ bulge chasing row/column exchanges may also shift split-points in a similar fashion to shifts resulting from row/column exchanges eliminating band nonzeros.

Figure 5.12 demonstrates the mechanism by which a bulge chasing rotation shifts a split-point. The elimination of the first nonzero above the split-point creates a bulge entry as shown in matrix A of Figure 5.12. When this entry is chased with a row/column exchange, the transformation switches $A_{7,4}$ off the main diagonal and moves SPA closer to the midpoint, creating matrix B.

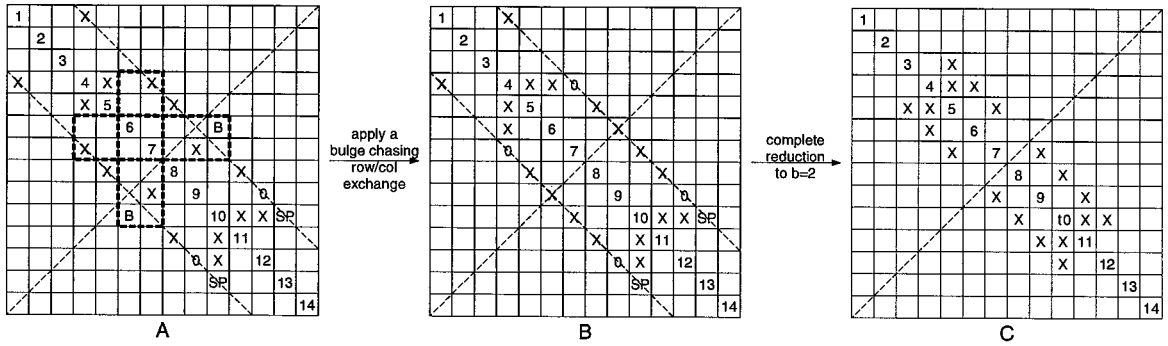


Figure 5.12: Multiple Split-Point Shifts

Continuing this reduction illustrates the potential of distinct shifting events combining to shift a split-point multiple positions during the reduction of a single diagonal. As the result of a bulge chasing shift, a band elimination shift and the normal half row shift of a diagonal's reduction, SPA's displacement reduces from 3 in matrix A to 0.5 in the bandwidth 2 intermediate matrix C.

It is important to reiterate, however, that the location in the neighboring diagonal to which a split-point is *shifted* may already have nonzero entries or fill may destroy the split-point during the first diagonal's reduction. As the separation of SPA and SPB grows beyond the current bandwidth, there are more opportunities for fill to destroy split-points during shifting. In addition, appropriate positioning of zero entries is essential to permit the use of the row/column exchanges needed for bulge chasing and band elimination shifts.

Having demonstrated several circumstances under which split-point SPA (SPB) can be shifted closer to the midpoint by using SPB (SPA), we are ready to consider the form of the induced split-point selection strategy more closely. Given the difficulties encountered by

unidirectional schemes, encouraging split-point flipping seems a reasonable way to exploit split-point shifting. For example, suppose SPA and SPB are single zero split-points of equal displacement, and SPA is used to reduce the current subdiagonal, b. In diagonal $(b-1)$ the new SPA must have a larger displacement, so it seems reasonable to use SPB for diagonal $(b-1)$'s reduction. Assuming SPA is not destroyed during $(b-1)$'s reduction, it will be available for the elimination of diagonal $(b-2)$ with a potentially reduced displacement. This example does not imply, however, that the induced strategy should select split-points from strictly alternating sides of the midpoints of a sequence of diagonals. Unlike this case, there may be a wide discrepancy between the displacements of SPA and SPB. If split-points are chosen in an alternating fashion, a split-point with an unacceptably large displacement might be chosen over a split-point of significantly smaller displacement. A solution to these difficulties is to promote split-point flipping with careful regulation of the selection process to avoid split-points with excessively large displacements.

In a similar fashion to the damped selection strategy, the induced split-point selection strategy controls the selection of each diagonal's split-point with the following hysteretic process. (We refer to the corresponding version of the Split Bandwidth Contraction algorithm as ISBC.)

IF $(IF * |D_{OS}|) \leq |D_{SS}|$ THEN
    select split-point OS
ELSE
    select split-point SS

Once again, SS and OS are the minimum displacement split-points on the same side and opposite side of the midpoint as the split-point chosen for the previous diagonal's reduction. D_{SS} and D_{OS} represent the displacements of split-points SS and OS. In the special cases when the roles of *opposite* and *same* side are undefined (see page 102), the diagonal's minimum displacement split-point is chosen.

The *inducement factor*, IF, regulates the amount by which the displacement of OS may exceed the minimum displacement and still permit split-point flipping. $(0 \leq IF \leq 1)$ When

$IF = 1$ the induced split-point selection strategy is equivalent to the minimum displacement strategy, while if $IF = 0$ split-points are selected from alternating sides of the midpoint.

As for previous selection strategies, we explored the relative merits of ISBC by performing partial bandwidth contractions of Table 5.3's 15 sparse problems with a `Trisymb` symbolic implementation. Each problem was contracted to a predetermined bandwidth using a variety of inducement factors in the range $0 \leq IF \leq 1$ and compared to SBC's contraction of the same problem.

Independent of the inducement factor and test problem, however, we did not observe significant reductions in the computational requirements of Split Bandwidth Contraction using ISBC. In fact, even with reasonably large inducement factors, ISBC significantly increased the bandwidth contraction costs of some problems relative to SBC. For example, contracting NOS3 to bandwidth 40 using ISBC and $IF = 0.7$ increases the flop count by 36% relative to standard SBC. The flop count totals of ISBC, however, are typically within 1% of SBC's costs when $0.80 \leq IF \leq 1$. Large flop count increases are usually restricted to smaller inducement factors for which ISBC permits the selection of much higher displacement split-points to enable flipping. Small inducement factors do not always result in dramatic increases in computational requirements. In fact, BCSSTK12, DWT 1007, LSHP 577 and PLAT1919 are relatively immune to inducement factor variation.

The goal of the induced split-point selection strategy is to reduce bulge chasing requirements by choosing a sequence of split-points with minimal total displacement. To meet this goal it encourages split-point flipping to exploit shifting and to bulge chasing balance across the midpoints of the outermost diagonals. Split-point shifting was observed during the reduction of a majority of the test problems. In most cases, however, displacement savings resulting from the selection of shifted split-points just managed to offset the extra displacement of split-points chosen to provide split-point flipping. In other cases the extra displacement sacrificed to induce a split-point flip far exceeds any reduction in the displacement of subsequent split-point selections. Similarly, induced split-point selection improved the balance bulge chasing for many problems, but its impact on the contraction of practical

problems was lower than anticipated.

In conclusion, exploring the induced split-point selection strategy has clearly demonstrated, once again, that the paramount concern of split-point selection must be to choose split-points of local minimum displacement. Although the induced split-point selection strategy was not successful, through these investigations we have learned much about the dynamics of the Split Bandwidth Contraction algorithm.

5.1.4.5 The Final Form of the Split-Point Selection Strategy

The introduction to Section 5.1.4 identified several factors affecting the performance of split bandwidth contractions. The four global selection strategies investigated in previous subsections explored trade-offs between these factors from the perspective of split-point selection. Among the presented alternatives, these studies have clearly shown that selecting a minimum displacement split-point for each diagonal's elimination is the best overall strategy. It might be possible to design special selection strategies to more fully exploit common sparsity structure characteristics of specific classes of matrices, but the minimum displacement split-point selection strategy results in an efficient, generally applicable method.

As described in Section 5.1.4.1, the minimum displacement strategy currently arbitrarily breaks ties between two split-points of equal (minimal) displacement. Alternatively, we propose resolving these equivocal selections with either damped or induced split-point techniques. Neither technique improves upon the minimum displacement strategy when employed as a global strategy, but the damped split-point selection strategy generally degrades the performance of a contraction less than the induced approach. Consequently, the final version of SBC's *find_split-point* routine (see Section 5.1.3) employs the minimum displacement selection strategy and in the case of ties selects the split-point on the same side of the midpoint as the split-point chosen for the previous diagonal.

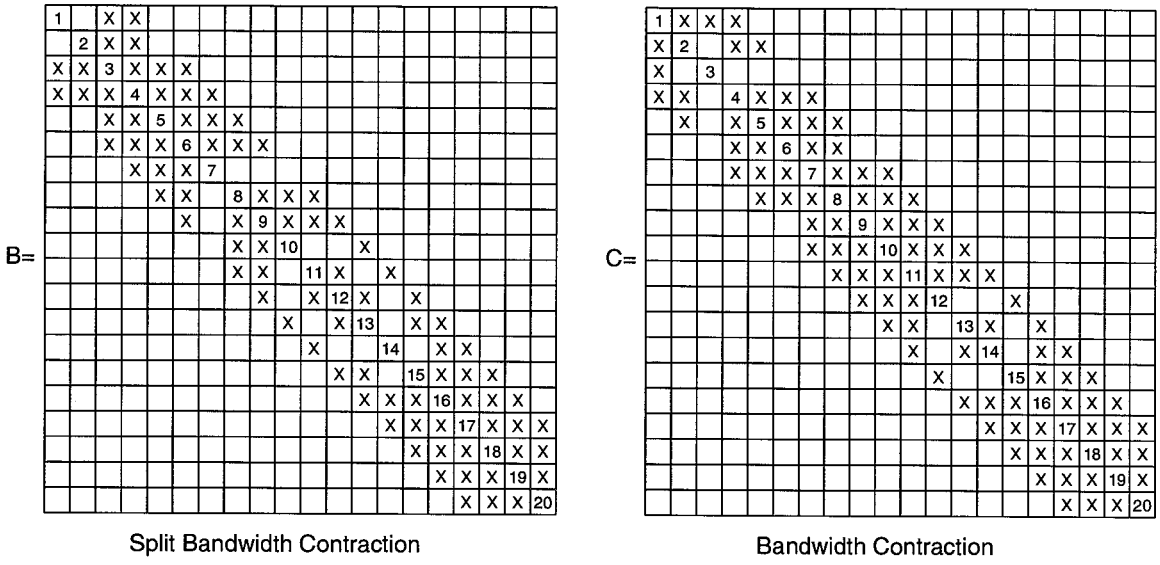


Figure 5.13: A Partial Bandwidth Contraction

5.1.5 Performance of the Split Bandwidth Contraction Algorithm

To illustrate the effectiveness of the Split Bandwidth Contraction algorithm we performed experiments with the symbolic reduction tools `Xmatrix` and `Trisymb`, comparing SBC's flop and transformation requirements to those of BC, as well as sparse R-S and HYBBC. All results assume the use of fast Givens transformations and sparsity exploitation as described by each algorithm's definition.

Consider SBC's reduction of the small sparse example discussed in Sections 4.1.3 and 4.2.3, and illustrated by matrix A of Figure 4.5. Figure 5.13 illustrates the intermediate reduction matrix resulting from SBC's elimination of the three outermost nonzero diagonals of the band using the lower triangular split-points in columns 7, 9 and 8, as chosen by SBC's routine *find_split-points*. For comparison, Figure 5.13 also illustrates the corresponding intermediate matrix from BC's reduction of the same problem. For this partial contraction BC uses 12 row/column exchanges and 16 nontrivial transformations, requiring 736 flops. Bidirectional elimination techniques, however, enable SBC to more fully exploit the sparsity of the band away from the main diagonal. SBC uses 10 row/column exchanges and 12

Method	Row/Column Exchanges	Nontrivial Transformations	Flops
Sparse R-S	8	132	7232
BC	12	163	6537
HYBBC	12	136	5880
SBC	12	130	5026

Table 5.4: Tridiagonalization Summary

nontrivial transformations, reducing computational requirements by 24% to 560 flops. In addition, Table 5.4 compares the transformation and flop requirements tridiagonalizing this small example with SBC, sparse R-S, BC and HYBBC. The Split Bandwidth Contraction algorithm requires 30.5%, 23.1% and 14.5% fewer floating point operations than sparse R-S, BC and HYBBC respectively. Finally, we note SBC requires even fewer nontrivial transformations than sparse R-S.

As demonstrated by the previous example, SBC effectively exploits band sparsity to efficiently tridiagonalize sparse matrices. However, we envisage SBC's most important contribution will be as a partial bandwidth contraction technique used by the first half of a hybrid tridiagonalization algorithm (See Section 5.2.), and as a preprocessing technique for sparse linear systems solvers and other banded eigenvalue routines such as BQR [GBDM77] or parallel implementations of Lang's densely banded tridiagonalization [Lan92]. To gauge the potential of SBC in this role, we conducted an extensive comparison of partial BC and SBC contractions using `Trisymb` implementations and the practical problems of Section 2.2's test suite of Harwell–Boeing sparse problems. For each problem, both partial bandwidth contractions continue until reaching the Δ-transition bandwidth of the Hybrid Split Bandwidth Contraction algorithm to be discussed in Section 5.2.2. All problems were preordered with GPS.

These symbolic experiments confirm that relative to BC the Split Bandwidth Contraction algorithm significantly reduces the flop requirements of partial reductions. For the 70 test problems of order 400 or greater, SBC requires on average 17% fewer floating point operations than BC. From this subset of problems SBC reduces the flop requirements of

Name n, b^{GPS}, b^f	Transformation Totals (bulge chasing, row/col. exch.) BC	SBC	Flops ($\times 10^6$) BC	SBC	%Flop Reduction
NOS5 468, 88, 49	21031 (14121,3308)	14360 (7694,3146)	7.5415	4.5001	40.3%
BCSSTK19 817, 18, 3	365696 (358087,1185)	197039 (189348,248)	27.731	15.118	45.5%
DWT 1007 1007, 34, 19	76910 (71201,2682)	21554 (17952,70)	14.064	3.9477	71.9%
CAN 1072 1072, 156, 48	289032 (239795,17051)	180585 (128192,16221)	145.15	86.232	40.6%
1138 BUS 1138, 126, 11	733720 (679001,71770)	482193 (422750,60327)	135.36	90.61	33.1%
ERIS1176 1176, 100, 4	1304873 (1251204,19457)	627337 (575732,70193)	201.04	72.857	63.8%
BCSPWR06 1454, 100, 12	1319287 (1254062,92591)	646685 (583013,68994)	269.94	121.35	55.1%
BCSSTK11 1473, 62, 33	354180 (326338,1397)	184820 (157859,1074)	119.62	61.780	48.4%
PLAT 1919 1919, 80, 30	889023 (828261,30884)	540000 (480506,8813)	309.76	193.76	37.5%
BCSSTK25 15439, 238, 162	16171900 (15611226,484692)	10273284 (9668324,519577)	22543	13838	38.6%

Table 5.5: Selected Partial BC and SBC Contraction Summaries

20 sparse problems by more than 35% relative to BC, and reductions of 40–50% are common. These results are consistent with the theoretical analysis of Section 5.1.3, which predicted that a well centred split-point permitted SBC to eliminate a nonzero diagonal from the special model problem with approximately half the computational effort of BC. This theoretical result, however, does not bound the maximum flop count reductions for these practical problems, which range as high as 71.9% for DWT 1007. SBC's reduction of this problem is able to take extra advantage of band sparsity to further shorten the length of bulge chasing sequences and dramatically improve performance.

Table 5.5 summarizes the computational requirements of partial BC and SBC contractions for 10 select test problems. The symbols b^{GPS} and b^f refer to the bandwidth of the GPS preordering and the final bandwidth of the partial contraction respectively. The two values in parentheses below each transformation total provide the number of bulge chasing transformations and row/column exchanges included in the total count. The final column

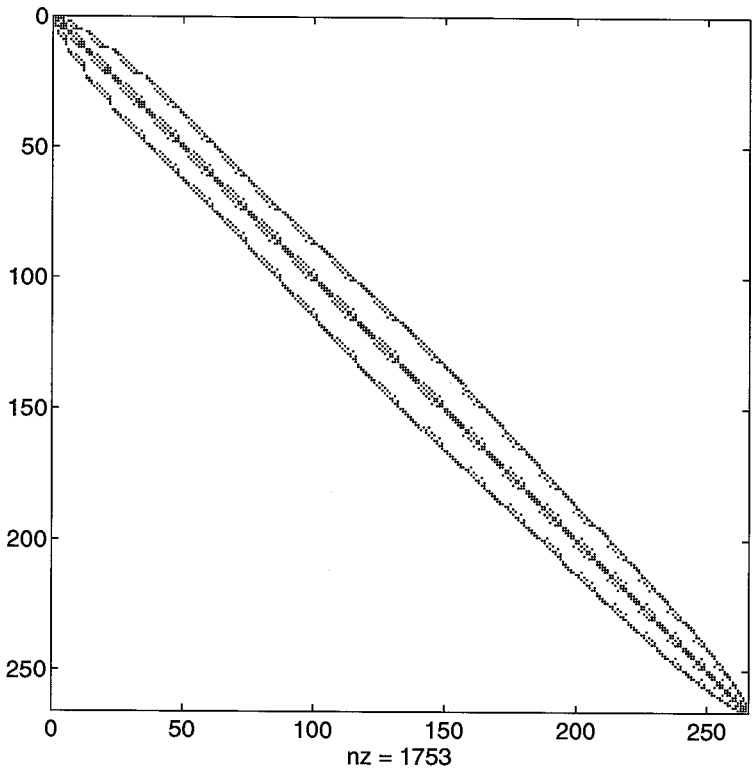


Figure 5.14: The Sparsity Structure of LSHP 265 with a GPS Preordering

of the table reports the reduction in flop counts achieved by SBC as a percentage of BC's costs.

SBC does not improve the efficiency of partial bandwidth contractions for all sparse problems. For example, the BC and SBC reductions of an L-shaped problem (LSHP 265, LSHP 406, ...) are equivalent, because of their special sparsity structures under GPS preorderings. As demonstrated by the plot of LSHP 265's sparsity structure in Figure 5.14, the envelope encompassing the nonzeros of the band is bow shaped. Its outermost nonzero diagonal consists of a single contiguous block of nonzeros straddling the midpoint. Consequently, when SBC chooses the split-point at the top end of this nonzero block, all entries above the split-point are zero and the diagonal's reduction reverts to a unidirectional elimination. Similarly, SBC's reduction of subsequent diagonals cannot exploit bidirectional techniques and the SBC and BC contractions use identical sequences of transformations.

With different preordering, such as GK, SBC is able to take better advantage of band sparsity than BC.

For 8 of the 115 test problems SBC actually requires more floating point operations than BC. For 7 problems from this group, SBC exhibits moderate increases in flop requirements ranging from 2–12%. Although SBC generally requires fewer transformations for these matrices, peculiar characteristics of their sparsity structures result in a lower proportion of row/column exchanges than in BC's reduction. As a result, SBC may actually require more nontrivial transformations than BC, causing higher flop requirements. The tendency of SBC to trade row/column exchanges for nontrivial transformations also affected problems for which SBC significantly reduces flop requirements. In these cases, SBC does not meet performance improvement expectations that are based solely on the displacement of each diagonal's split-point. Currently, we do not have a general understanding of this complex phenomenon, and must deal with each problem on an individual basis.

For the final problem of this group, BCSSTK09, SBC increases flop requirements by a factor of more than 2.5 relative to BC. To understand this dramatic increase we examine the distinctive sparsity structure of BCSSTK09 illustrated in Figure 5.15. As SBC reduces the first few nonzero diagonals from the band, split-points are available close the midpoint of each diagonal. Consequently, band nonzeros in *region B* are eliminated with column-oriented transformations, producing band fill in *region A*. This increases the local bandwidth of *region A*, gradually forcing nonzeros outwards to meet the edge of the contracting band. During BC's elimination of the same diagonals, transformations do not impinge upon *region A* and fill in this region is avoided until the bandwidth has been further contracted. As the effects of SBC's fill in *region A* cascade into the remainder of the band, however, SBC exhibits significantly higher intermediate band densities. For example, once the bandwidth of BCSSTK09 has been reduced from 95 to 76, SBC's intermediate reduction matrix has 1.87 times as many band nonzeros as BC's corresponding reduction intermediate. Consequently, each diagonal requires more band zeroing transformations and longer bulge chasing sequences, employing a dramatically higher proportion of nontrivial transformations. When BCSSTK09 is reduced by the numeric implementation of SBC discussed

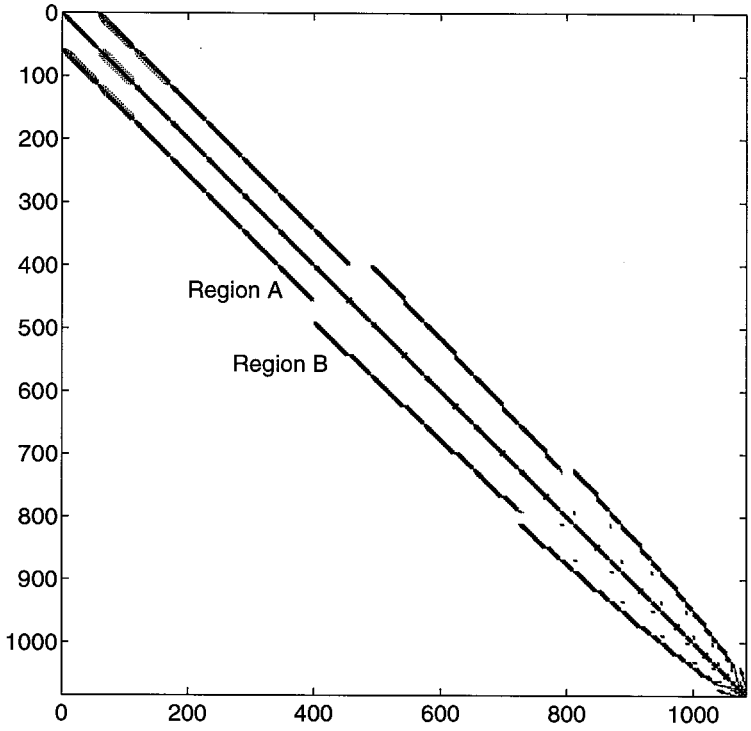


Figure 5.15: The Sparsity Structure of BCSSTK09 with a GPS Preordering

in Section 5.3, the discrepancy between the SBC and BC contractions is less pronounced, because row/column exchanges are no longer without cost. In this case, SBC requires only 16.5% more CPU seconds than BC.

With the exception of the previous groups of problems, the bidirectional elimination techniques of SBC generally reduce transformation requirements significantly. As the result of fewer nontrivial transformations, we anticipated, but did not observe, comparable reductions in intermediate band densities. This apparent inconsistency is due to *transformation saturation.* During a typical reduction, SBC updates particular pairs of rows and columns in the band with many nontrivial transformations. Only a few of these transformations, however, may actually produce fill entries within a row/column pair. Suppose a nontrivial transformation unions the sparsity structure of a pair of rows and columns. Subsequent transformations applied to the same pair will not affect their sparsity structures

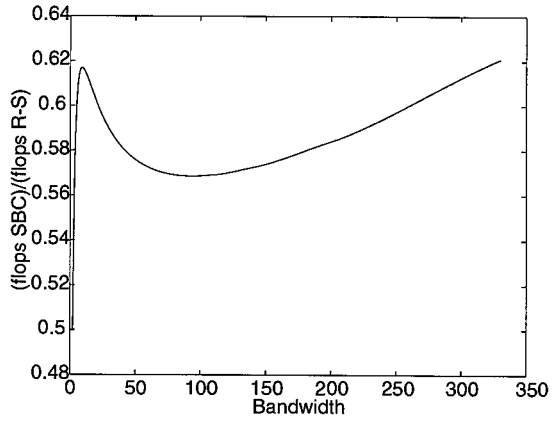
unless between transformations the sparsity of one of the rows or columns in the pair has been unioned with a third row or column. As a result, significantly lower transformation levels may not be accompanied by similar reductions in the band density of intermediates matrices. Despite reduced numbers of transformations, there may still be a sufficient variety of transformations to saturate the band and produce similar levels of band fill.

This section has evaluated the Split Bandwidth Contraction algorithm by manipulating sparsity structures with symbolic implementations. Discussion in Section 5.4.1 confirms the general characteristics of these results with numerical routines for SBC.

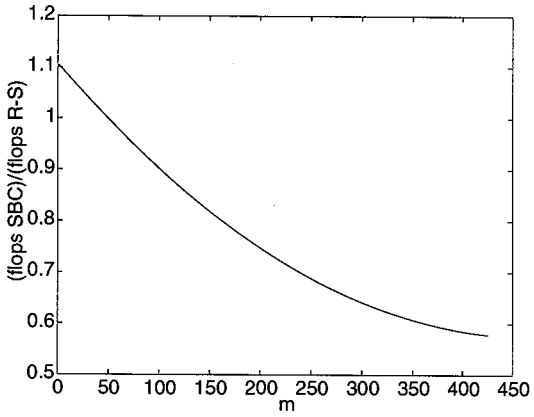
5.2 The Hybrid Split Bandwidth Contraction Algorithm

Chapter 4's Hybrid Bandwidth Contraction algorithm, HYBBC, combines the BC and R-S algorithms to produce an effective two stage tridiagonalization. HYBBC's first stage employs BC, to exploit its efficient contraction of a sparse band, but switches to R-S once the contracted band becomes too dense and the costs of completing the tridiagonalization with BC overtake those of R-S. As shown in Section 5.1, however, partial bandwidth contractions using the bidirectional elimination techniques of the Split Bandwidth Contraction algorithm are often dramatically more efficient than BC's unidirectional elimination. This observation suggests replacing BC with SBC to create a second generation tridiagonalization algorithm, the Hybrid Split Bandwidth Contraction algorithm or HYBSBC.

To produce an efficient and versatile tridiagonalization algorithm, HYBSBC must also switch to R-S when band fill reduces the effectiveness of the SBC contraction and R-S can more efficiently complete the reduction. Consider SBC's reduction of a special family of symmetric model problems with n=1000 and various bandwidths. Each matrix is densely banded, but we assume under an SBC reduction that a single split-point in column m of the lower triangular portion is available for each diagonal's elimination. Of course, this is an unlikely sequence of sparsity structures for SBC to encounter, because it is only possible if numerical cancellation provides the appropriately positioned split-points. Using the complexity results for R-S and SBC in Tables 3.4 and 5.1, the first graph in Figure 5.16



Centred Split-Points



Each Diagonal is eliminated by a split-point in column m. b=150

Figure 5.16: The Tridiagonalization Flop Requirements of SBC Relative to R-S for a Densely Banded Model Problem. n=1000

plots the tridiagonalization flop requirements of SBC with centred split-points, normalized by the cost of R-S, against the bandwidth of the model problem. As expected, this plot shows that as long as SBC finds a well centred split-point for each diagonal's reduction, it provides tridiagonalizations with significantly lower flop requirements than R-S. These densely banded model problems, however, provide a worst case analysis. If SBC finds a well centred split-point for each diagonal while tridiagonalizing a sparsely banded problem, it typically enjoys even lower flop requirements relative to R-S.

For the reduction of practical problems SBC cannot expect to always find a well centred split-point for the elimination of each diagonal. Like BC, SBC must also contend with fill entries inside the contracting band. As discussed in Sections 5.1.4.1 and 5.1.4.4, during SBC's reduction of a typical sparse problem an expanding block of nonzeros, straddling the centre of the outermost diagonal of successive intermediate matrices, forces split-points away from the midpoint. For example, Figure 5.17 plots the displacements of the split-points used by SBC during its reduction of BCSPWR08 from bandwidth 108 to 12. As predicted by the analysis of Section 5.1.4.4, split-point displacements decrease occasionally during the reduction, but split-points of higher and higher displacement is the dominant

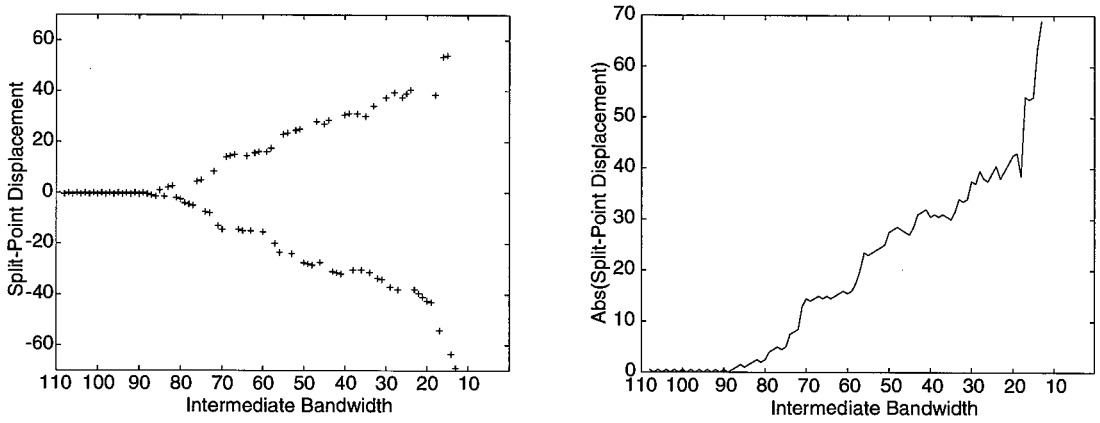


Figure 5.17: Split-Point Displacements of SBC's Partial Contraction of BCSPWR08

trend. The speed with which split-points are forced away from the midpoint is a measure of SBC's relative success.

If split-points are sufficiently far from the midpoint, the cost of using SBC to complete the tridiagonalization may rise above that of R-S. For example, the second graph in Figure 5.16 plots the tridiagonalization flop requirements of SBC, normalized by the costs of R-S, as a function of split-point positioning for the special model problem with b=150. Each SBC reduction assumes the lower triangular split-point for each diagonal's elimination is in the same column, m. It is not surprising that for sufficiently small m SBC's flop counts exceed those of R-S, considering SBC converges to the BC algorithm as split-points with larger and larger displacement are employed. In fact, Figure 4.7 demonstrates that when each diagonal's split-point is in column 1, and SBC is equivalent to BC, SBC may require more than 20% additional flops relative to R-S for select densely banded problems. Consequently, if the Hybrid Split Bandwidth Contraction algorithm is to be efficient and generally applicable, it must be able to detect SBC's inability to effectively reduce the current intermediate matrix and switch to R-S to complete the tridiagonalization.

The remainder of this section provides a formal description of the HYBSBC algorithm before developing the Δ-transition strategy, which precisely regulates the transition between the algorithm's SBC and R-S stages.

5.2.1 An Algorithmic Framework for HYBSBC

To formalize the previous discussion, Figure 5.18 provides a pseudocode framework defining the Hybrid Split Bandwidth Contraction algorithm. Once again, HYBSBC begins by symmetrically permuting the sparse matrix to reduce bandwidth. While the matrix is not in tridiagonal form and the transition condition has not been met, the algorithm then contracts the band diagonal by diagonal using the bidirectional elimination techniques of SBC. The column index, *sp*, of the lower triangular split-point used for a diagonal's elimination is returned by the function *find_split-point* defined in Section 5.1.4.5. Once the transition bandwidth b^t identified by the function *transition* has been reached, sparse R-S completes the tridiagonalization if $b^t > 1$. This second stage of the algorithm is identical to Section 3.5.2's description of sparse R-S, except HYBSBC does not perform an additional preordering.

As for the HYBBC algorithm, a sensitive design issue is the selection of an appropriate transition strategy. For example, suppose we apply HYBSBC to the small sparse problem in Figure 4.5 and use a transition strategy that chooses $b^t = 3$. In this case HYBSBC's tridiagonalization uses 11 row/column exchanges and 127 nontrivial transformations, increasing the cost of tridiagonalization relative to SBC by 9% to 5473 flops. To avoid such inefficiencies and to take full advantage of SBC, the following subsection develops the *Δ-transition strategy*, which is shown to select optimal transition bandwidths for practical sparse problems.

5.2.2 The Δ-Transition Strategy

The design of transition strategies for the HYBBC algorithm was hindered by our inability to cheaply estimate the cost of eliminating a single diagonal with BC from a general sparsely banded matrix. Consequently, HYBBC employs a relatively unsophisticated transition scheme based on density measures of the outermost nonzero diagonal. In contrast, HYBSBC lends itself to formal analysis. By exploiting characteristics of a typical SBC reduction, the Δ-transition strategy is able to use complexity analyses of SBC and R-S to precisely

1. $A := P^T AP$, where P is a bandwidth reducing permutation matrix.
2. $b :=$ bandwidth(A)
3. (a) $b^c := b$
   (b) $sp =$ *find_split-point*(A, b^c)
   (c) /*While the matrix is not tridiagonal and the transition condition has*/
   /*not been met, eliminate the outermost nonzero diagonal with SBC.*/
   WHILE ($(b^c \geq 2)$ AND (NOT *transition*(n, b^c, sp))) DO
      i. FOR $col := sp - 1$ DOWNTO 1 DO /*Eliminate above the split-point*/
         IF $A_{col+b^c,col} \neq 0$ THEN /*Zero $A_{col+b^c,col}$.*/
            IF $A_{col+b^c,col+1} = 0$ THEN
               Exchange rows/columns (col) and $(col+1)$ in A.
            ELSE
               $A := G(col, col+1, \theta)^T \; A \; G(col, col+1, \theta)$
            IF bandwidth(A) $> b^c$ THEN
               Chase bulges with additional column-oriented adjacent
               Givens transformations or row/column exchanges.
         ENDIF /*Outermost IF*/
      ii. FOR $col := sp + 1$ TO $n - b^c$ DO /*Eliminate below the split-point*/
         IF $A_{col+b^c,col} \neq 0$ THEN /*Zero $A_{col+b^c,col}$.*/
            IF $A_{col+b^c-1,col} = 0$ THEN
               Exchange rows/columns $(col + b^c)$ and $(col + b^c - 1)$ in A.
            ELSE
               $A := G(col + b^c, col + b^c - 1, \theta)^T \; A \; G(col + b^c, col + b^c - 1, \theta)$
            IF bandwidth(A) $> b^c$ THEN
               Chase bulges with additional row-oriented adjacent
               Givens transformations or row/column exchanges.
         ENDIF /*Outermost IF*/
      iii. $b^c = b^c - 1$
      iv. $sp =$*find_split-point*(A, b^c)
   (d) $b^t = b^c$ /*Record the transition bandwidth.*/
   (e) IF $b^c > 1$ THEN complete tridiagonalization with sparse R-S.

Figure 5.18: The Hybrid Split Bandwidth Contraction Tridiagonalization Algorithm

regulate HYBSBC's transition bandwidth and minimize computational requirements. After detailing the Δ-transition strategy, this section assesses its potential using a subset of the Harwell–Boeing test suite and symbolic reduction tools.

5.2.2.1 A Theoretical Basis

As demonstrated in the introduction to Section 5.2, the Rutishauser-Schwarz algorithm is clearly superior to bandwidth contraction schemes for the tridiagonalization of matrices with a dense band. When applying HYBSBC to a band with entries that are zero, however, Figure 5.16 demonstrates that continuing the SBC stage of the reduction is cost-effective while a split-point exists near enough to the midpoint of the outermost nonzero diagonal. Unfortunately, as a typical reduction proceeds the best split-points available for each diagonal's elimination generally exhibit larger and larger displacements. At a particular intermediate bandwidth, allowing SBC to continue one extra step using an off-centre split-point before switching to R-S may be more costly than switching to R-S immediately. The job of HYBSBC's transition strategy is to identify the optimal transition bandwidth b^t that minimizes the following cost function.

$$\text{Cost\_SBC}(b \rightarrow b^t) + \text{Cost\_R-S}(b^t \rightarrow 1) \tag{5.1}$$

Cost_SBC() and Cost_R-S() are problem dependent functions representing the computational requirements of SBC and R-S performing the indicated reductions. To permit the development of a generally applicable transition scheme, we simplify Equations 5.1's minimization by assuming that the displacement of split-points selected by an SBC reduction increase monotonically with contracting bandwidth. Although SBC violates this assumption for several sparse problems, decreases in split-point displacements are typically small local anomalies, overwhelmed by the dominant trend towards progressively larger displacements. (See Figure 5.17.)

If we assume monotonic increases in split-point displacement, it is also reasonable to assume that the flop requirements of eliminating each successive diagonal increase as the reduction progresses. A dominant factor governing the cost of SBC's elimination of a diagonal

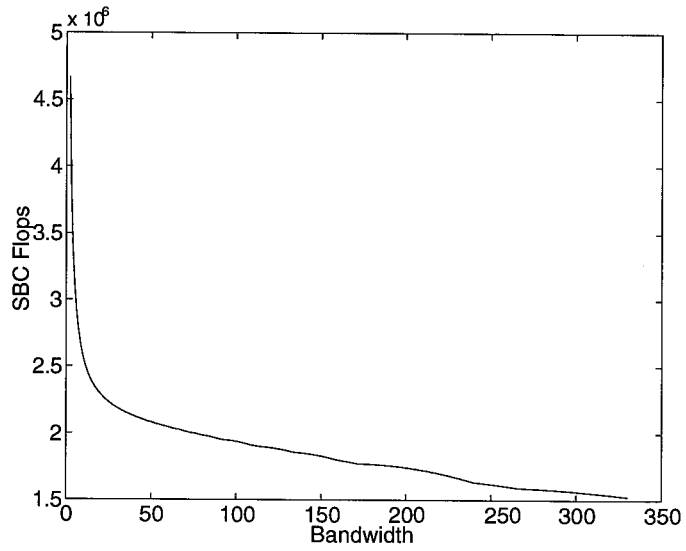


Figure 5.19: The Cost of Eliminating the Outermost Nonzero Diagonal of a Densely Banded Matrix (n=1000) Using a Split-Point of Displacement 100.

from a sparse band is the displacement of its best split-point. Increasing the split-point's displacement generally increases the cost of a diagonal's elimination. In addition, if split-point displacement is held constant as we lower the bandwidth of the intermediate matrix from which the diagonal is eliminated, elimination costs also generally rise. Figure 5.19 demonstrates this characteristic of SBC reductions for a family of densely banded matrices with a single split-point of displacement 100 in their outermost diagonal. Finally, we also assume the cost of completing a sparse tridiagonalization with R-S essentially decreases at a constant rate as the bandwidth contracts. This is a practical assumption, considering the speed with which R-S typically fills the band of a sparse matrix and that the dominant term of the densely banded flop analysis of R-S, $F_{\text{R-S}}^{\text{FG}}$, is $(4b+6)n^2$. Using the full $F_{\text{R-S}}^{\text{FG}}$ analysis, Figure 5.20 plots the flop requirements of R-S, as a function of b, for the tridiagonalization of densely banded matrices of order 1000.

Given these assumptions, the transition strategy can approximate the minimization of Equation 5.1's cost function by comparing the following costs before the elimination of each

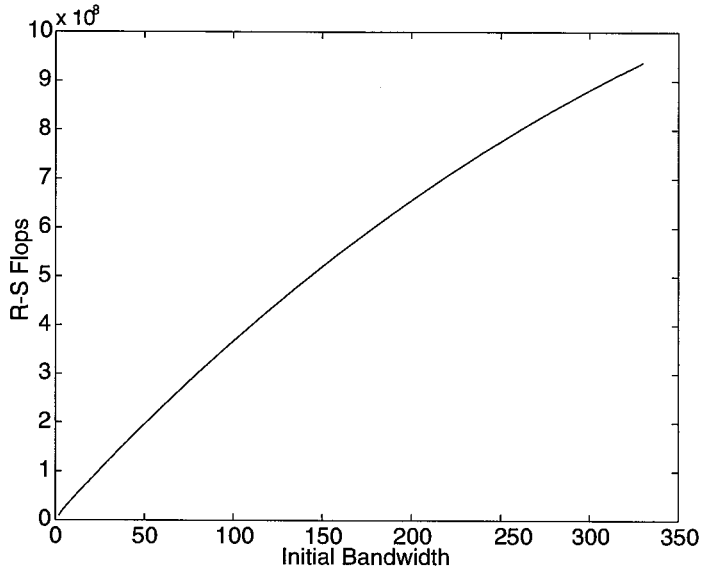


Figure 5.20: R-S Tridiagonalization Costs of Densely Banded Matrices. n=1000

diagonal. The bandwidth of the reduction's current intermediate matrix is given by b^c.

$$C3 = \text{Cost\_SBC}(b^c \rightarrow (b^c - 1)) + \text{Cost\_R-S}((b^c - 1) \rightarrow 1) \quad (5.2)$$

$$C4 = \text{Cost\_R-S}(b^c \rightarrow 1) \quad (5.3)$$

As long as $C3 < C4$ HYBSBC should continue eliminating diagonals with SBC. Once $C3$ grows larger than $C4$, however, the reduction switches to R-S to complete the tridiagonalization.

Although our assumptions have greatly simplified the optimization of Equations 5.1's cost function, predicting $C3$ and $C4$ exactly for general sparsity structures is difficult without unacceptably expensive modeling. The Δ-transition strategy avoids the difficulties of exact analysis by employing the results of two theoretical analyses, $F^{\text{FG}}_{\text{R-S}}$ and $F^{\text{FG}}_{\text{SBC1}}$ from Sections 3.4.1 and 5.1.3, to approximate components Cost_SBC() and Cost_RS() of $C3$ and $C4$. Throughout the following discussion we assume HYBSBC employs fast Givens transformations, but a completely analogous scheme could be developed for implementations based on standard Givens transformations.

$F^{\text{FG}}_{\text{R-S}}$ and $F^{\text{FG}}_{\text{SBC1}}$ estimate the flop requirements of specific densely banded reductions, but the Δ-transition strategy will be applied during the tridiagonalization of matrices with a wide variety of sparsely banded structures. A number of factors make the use of $F^{\text{FG}}_{\text{R-S}}$ and $F^{\text{FG}}_{\text{SBC1}}$ an acceptable compromise. First, without having detailed knowledge of a banded intermediate's sparsity structure both analyses provide an upper bound on the flop requirements of a corresponding sparsely banded reduction. For example, given the triple of values (n, b^c, m) and no additional knowledge of the band's sparsity structure, $F^{\text{FG}}_{\text{SBC1}}$ approximates the cost of eliminating the outermost nonzero diagonal by assuming the worst case in which all band entries except the split-point are nonzero. As the reduction proceeds the band suffers from fill and split-points are forced away from the midpoint. This makes the predicted flop counts more and more accurate, because information about the outermost diagonal's sparsity structure is implicitly represented by the minimum displacement split-point selected by SBC. We know, for example, there is a contiguous block of at least $2(|\lceil(\frac{N-b^c}{2})\rceil - m| + 1)$ nonzeros straddling the midpoint. Once split-points are sufficiently far from the midpoint to perhaps warrant a transition, this nonzero block typically comprises a large majority of the sparse diagonal's entries. As discussed in Section 4.3.2, band sparsity is best exploited at the transformation level. Thus during the critical period of HYBSBC's reduction, $F^{\text{FG}}_{\text{SBC1}}$ provides progressively more accurate approximations of the outermost diagonal's reduction costs. $F^{\text{FG}}_{\text{R-S}}$ also provides good approximations of the true cost of tridiagonalizing the contracted band of intermediate matrices, because of the speed with which R-S typically fills the band of a sparse band. Once again, the longer the reduction remains in the SBC stage of the algorithm producing fill within the band, the better $F^{\text{FG}}_{\text{R-S}}$'s approximation becomes.

Using the results of these analyses we define the Δ-transition function, $\Delta(n, b^c, m)$, as the difference between costs $C4$ and $C3$.

$$\Delta(n, b^c, m) = F^{\text{FG}}_{\text{R-S}}(n, b^c) - [F^{\text{FG}}_{\text{SBC1}}(n, b^c, m) + F^{\text{FG}}_{\text{R-S}}(n, b^c - 1)] \tag{5.4}$$

Each of the three flop counts in this formula incorporate the one time costs, given by Equation 2.16, of applying the transformation $D^{-1/2}TD^{-1/2}$ at the end of a reduction

employing fast Givens transformations. The Δ-transition strategy should not consider these costs, and they are nullified by adding $3n$ to the right hand side of Equation 5.4.

$$\Delta(n, b^c, m) = F^{\text{FG}}_{\text{R-S}}(n, b^c) - [F^{\text{FG}}_{\text{SBC1}}(n, b^c, m) + F^{\text{FG}}_{\text{R-S}}(n, b^c - 1)] + 3n \tag{5.5}$$

To employ the Δ-transition strategy, HYBSBC checks the value of $\Delta(n, b^c, m)$ before SBC eliminates each diagonal of the band. While $\Delta(n, b^c, m) > 0$, the transition strategy confirms the cost-effectiveness of continuing the band's contraction with bidirectional elimination techniques. When $\Delta(n, b^c, m)$ drops below zero, however, HYBSBC switches to R-S to complete the tridiagonalization. Because the storage costs of HYBSBC are independent of the transition bandwidth, the Δ-transition strategy is based solely upon the prediction of computational requirements.

The following subsection presents explicit formulas for $\Delta(n, b^c, m)$ and investigates their practical application.

5.2.2.2 A Practical Δ-Transition Strategy

We explicitly form the Δ-transition function $\Delta(n, b^c, m)$ by inserting the complexity results $F^{\text{FG}}_{\text{R-S}}$ and $F^{\text{FG}}_{\text{SBC1}}$ from Sections 3.4.1 and 5.1.3 into Equation 5.5. Operation count summation difficulties encountered during SBC's analysis forced $F^{\text{FG}}_{\text{SBC1}}$ to be separated into two independent results, each valid for a specific range of split-point displacements. Correspondingly, we construct the two formulas for $\Delta(n, b^c, m)$ shown in Figures 5.21 and 5.22.

Both formulas assume that the column index m of the split-point refers to an entry in the top half of the lower triangular diagonal under elimination. In those cases when the split-point lies in the diagonal's bottom half, m is simply assigned the column index of the entry in the diagonal's upper half with identical displacement. When $1 \leq m \leq b^c$, we use $\Delta_{b^c}(n, b^c, m)$ to compute the value of the Δ-transition function, while if $b^c < m \leq \lceil \frac{n-b^c}{2} \rceil$ we employ $\Delta_{mid}(n, b^c, m)$. Figure 5.23 plots the Δ-transition function for matrices with $n = 1000$ and bandwidths of $10, 50, 150$ and 300.

Each formula has been simplified to reduce its evaluation cost, but both remain relatively

$$
\begin{aligned}
\Delta_{b^c}(n, b^c, m) \;=\; & -1 + 2((b^c)^2 - b^c) - (8 + 6b^c + 10m)m + \frac{20(m - m^2)}{b^c} \\
& + \left(14 - 2\,b^c - \frac{20}{(b^c)^2 - b^c} + (8 + \frac{20}{b^c})m\right)\,\mathrm{n} + \left(\frac{20 - 10b^c}{(b^c)^2 - b^c}\right)(n^2 + 1) \\
& + 2\left[\left(\frac{b^c + 4}{b^c - 1}\right) Mod(n - 1, b^c - 1)(b^c - 1 - Mod(n - 1, b^c - 1))\right. \\
& \quad + \left(\frac{b^c + 5}{b^c}\right)[Mod(n - 1, b^c)(-b^c + Mod(n - 1, b^c)) \\
& \quad - b^c(Mod(m - 1, b^c) + Mod(n - m, b^c)) + Mod(m - 1, b^c)^2 \\
& \quad \left. + Mod(n - m, b^c)^2] - b^c Mod(m - 1, b^c) + Mod(m - 1, b^c)^2\ \right]
\end{aligned}
$$

Figure 5.21: The Δ-Transition Function for $1 \leq m \leq b^c$

$$
\begin{aligned}
\Delta_{mid}(n, b^c, m) \;=\; & 1 + 2\,(b^c)^2 - (12 + 8(b^c + m))m + \frac{20(m - m^2)}{b^c} \\
& + \left(14 - \frac{20}{(b^c)^2 - b^c} - 2\,b^c + \left(8 + \frac{20}{b^c}\right)m\right)\,\mathrm{n} + \left(\frac{20 - 10b^c}{(b^c)^2 - b^c}\right)(n^2 + 1) \\
& + 2\left[\left(\frac{b^c + 4}{b^c - 1}\right)(Mod(n - 1, b^c - 1)(b^c - 1 - Mod(n - 1, b^c - 1)))\right. \\
& \quad + \left(\frac{b^c + 5}{b^c}\right)[Mod(n - 1, b^c)(-b^c + Mod(n - 1, b^c)) \\
& \quad - b^c(Mod(m - 1, b^c) + Mod(n - m, b^c)) \\
& \quad \left. + Mod(m - 1, b^c)^2 + Mod(n - m, b^c)^2]\ \right]
\end{aligned}
$$

Figure 5.22: The Δ-Transition Function for $b^c < m \leq \left\lceil \frac{n - b^c}{2} \right\rceil$

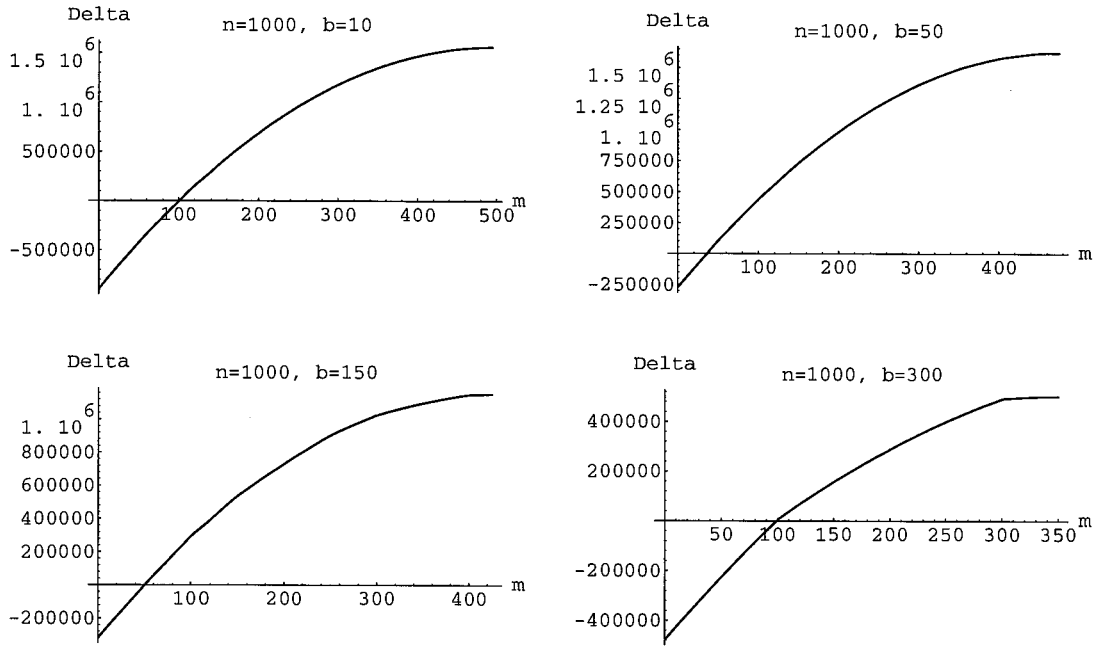


Figure 5.23: Variance of $\Delta(n, b^c, m)$ with Split-Point Positioning. $n = 1000, b^c = 10, 50, 150,$ and 300

complicated and lengthy. Many of the encumbering terms result from the nonanalytic correction terms involving $Mod()$. As discussed in Sections 3.4.1 and 5.1.3, these terms can be safely ignored under many circumstances. When the value of b^c is large relative to n, however, omitting these terms can shift the value of m at which $\Delta(n, b^c, m)$ is zero and significantly alter the behavior of the Δ-transition strategy. Fortunately, the cost of evaluating $\Delta(n, b^c, m)$ in its entirety before the reduction of each diagonal is minimal. Table 5.6 summarizes the total number of floating point and $Mod()$ operations required to compute $\Delta_{b^c}(n, b^c, m)$ or $\Delta_{mid}(n, b^c, m)$. Although many terms of the formulas could be evaluated using integer operations, we assume all calculations are conducted using floating point. Assuming a $Mod()$ operation is equivalent to approximately 3 flops, Δ_{b^c} and Δ_{mid} require 66 and 62 flops respectively. These costs are insignificant in comparison to the cost of reducing a typical diagonal from a moderately large practical problem. In fact, the cost

	Flops	$Mod()$s
Δ_{b^c}	54	4
Δ_{mid}	50	4

Table 5.6: $\Delta(n, b^c, m)$ Evaluation Cost Summary

Name	n	b^{GPS}
BCSSTK01	48	25
BCSSTK02	66	65
BCSSTK04	132	46
BCSSTK08	1074	475
CAN 73	73	27
CAN 256	256	86
CAN 268	268	101

Table 5.7: Sparse Problems with $b^{GPS} > n/3$.

of evaluating either formula is essentially equivalent to the cost of applying two fast Givens transformations to a penta-diagonal symmetric matrix.

The Δ-transition functions $\Delta_{b^c}(n, b^c, m)$ and $\Delta_{mid}(n, b^c, m)$ are only valid when $b^c \leq n/3$, because of assumptions underlying the SBC analysis. While the current bandwidth of a sparse matrix lies outside this range, the transition strategy must take an alternative approach. For several reasons, however, we do not feel that accommodating this special case warrants the expenditure of much effort. In our experience this class of problem is rare. Of the 115 problems in the Harwell–Boeing test suite, only 7 problems have $b^{GPS} > n/3$. (See Table 5.7.) In addition, we note that the reduction of a diagonal from a matrix with a relatively large bandwidth generally requires fewer bulge chasing transformations than a similar elimination from a matrix of lower bandwidth. In fact, when $b^c > n/2$ SBC creates no bulge entries while eliminating the outermost diagonal. As a result, the cost of a generic diagonal's reduction decreases as the current bandwidth gets larger. (For an example see Figure 5.19.) Thus if HYBSBC selects a poor transition bandwidth $b^t > n/3$, which is a few diagonals from the optimal transition, the penalty incurred is very small relative to the

```
FUNCTION transition(n, b^c, sp)
   IF (sp > ((n − b^c)/2)) THEN
      m = n − b^c − sp + 1
   ELSE
      m = sp

   IF (b^c > n/3) THEN
      IF ((no split-point) OR (n − b^c − 2m ≥ threshold ∗(n − b^c))) THEN
         return TRUE
      ELSE
         return FALSE
   ELSE IF ((split-point exists) AND (Δ(n, b^c, m) > 0)) THEN
      return FALSE
   ELSE
      return TRUE
END
```

Figure 5.24: HYBSBC's Δ-Transition Strategy

cost of HYBSBC's entire reduction.

The Δ-transition strategy can adequately regulate the reduction of sparse matrices with large bandwidths by simply adopting the density thresholding transition techniques of HYBBC while $b^c > n/3$. In this case if $(b^c > n/3)$ and $(nzcnt < (threshold * (n - b^c)))$ then HYBSBC eliminates the next diagonal with SBC. (*nzcnt* and *threshold* are defined in Section 4.2.2.) Rather than explicitly counting the number of nonzeros in the diagonal, however, we propose approximating *nzcnt* using the sparsity information implicit in the position of the chosen split-point. By definition all entries closer to the diagonal's midpoint than the selected split-point in column *sp* must be nonzero. Recall m is the column index of the split-point normalized to lie in the top half of the matrix's lower triangular diagonal. If we assume all entries in the first and last m columns of the diagonal are zero, *nzcnt*= $n - b^c - 2m$. Of course, in general $n - b^c - 2m$ is a lower bound on *nzcnt*. Using $n - b^c - 2m$ to approximate *nzcnt*, Figure 5.24 provides a pseudocode framework for the Δ-transition strategy by defining the boolean function *transition* used in Figure 5.18's description of

Name	n	b^{GPS}	*no-split* transition bandwidth	Δ-transition bandwidth	% flop reduction *no-split* $\rightarrow \Delta$
CAN 445	445	74	7	33	5.8%
CAN 634	634	100	14	57	6.7%
BCSSTK08	1074	475	1	159	8.1%
BCSSTK11	1473	62	1	33	13.0%
BCSSTK23	3134	351	10	66	2.0%
BCSSTK25	15439	238	1	162	5.8%

Table 5.8: Sparse Problems with Widely Varying *no-split* and Δ-Transition Bandwidths

the HYBSBC algorithm. We assume $\Delta(n, b^c, m)$ represents the form of the Δ-transition function, $\Delta_{b^c}(n, b^c, m)$ or $\Delta_{mid}(n, b^c, m)$, appropriate for the specific values of m and b^c.

5.2.2.3 Evaluation of the Δ-Transition Strategy

A general formal analysis of the optimality of the Δ-transition strategy, suitable for all sparse symmetric matrices, is not possible. To gauge the success of the Δ-transition strategy for practical problems, we conduct two sets of experiments with `Trisymb` symbolic implementations of HYBSBC and the Harwell–Boeing test suite. For these experiments the Δ-transition strategy uses a threshold of 1.0.

To benchmark the success of the Δ-transition strategy, our first set of experiments compares a symbolic implementation of HYBSBC to an almost identical implementation that replaces the Δ-transition strategy with a much simpler approach. This alternative transition strategy directs the hybrid algorithm to continue the SBC phase of the reduction until the matrix is in tridiagonal form or its outermost nonzero diagonal is void of split-points. For 46 problems in the Harwell–Boeing test suite, the Δ-transition strategy chooses a higher transition bandwidth than the *no-split* scheme. Table 5.8 presents examples of problems for which the Δ-transition strategy suggests a dramatically large increase in the transition bandwidth. For a majority of the problems, however, the difference between the two schemes' transition bandwidths is much smaller relative to the problem's initial permuted bandwidth.

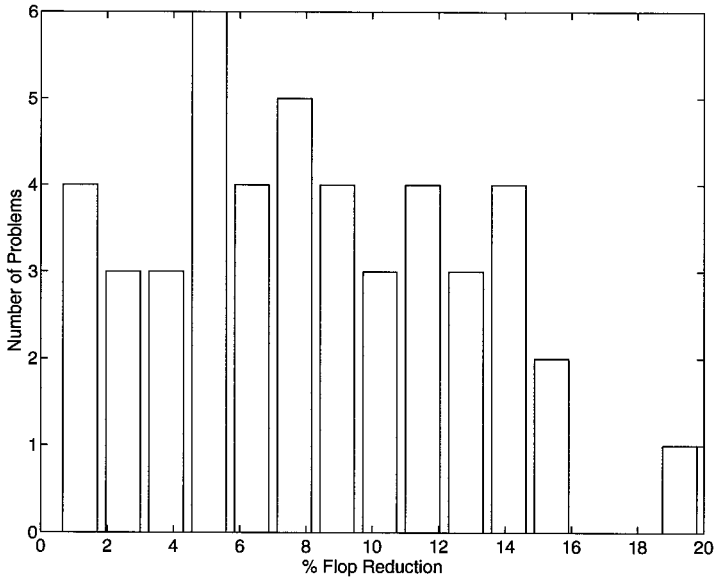


Figure 5.25: Distribution of the Δ-Transition Strategy's Flop Reductions

In comparison to the *no-split* approach, the Δ-transition strategy successfully reduces the flop requirements for each of the 46 problems with an earlier transition. Reductions in flop requirements range from 0.5 to 20%, but average 8.1% over all 46 problems. Figure 5.25 summarizes the distribution of flop reductions achieved by the Δ-transition strategy. As expected, the flop reductions attained by the Δ-transition strategy are comparable to the differences between the complexity of BC and R-S for densely banded problems observed in Section 4.2.1.

As predicted in Section 5.2.2.2, the cost of evaluating $\Delta(n, b^c, m)$ before SBC's reduction of each diagonal is an insignificant fraction of the entire tridiagonalization cost. For Harwell–Boeing problems with $n > 100$, the cost of evaluating the Δ-transition function is at most 0.2% of the total reduction costs. Typically, the fraction of the total reduction cost attributable to the Δ-transition strategy is at least an order of magnitude lower than this value.

The previous experiments clearly show that the Δ-transition strategy is superior to the rudimentary *no-split* transition bandwidth approach, but how well does it predict optimal

transition bandwidths? To determine the precision with which the Δ-transition strategy regulates the transition bandwidth, we investigate more closely the tridiagonalization costs of two representatives from each of the following general classes of sparse Harwell–Boeing problems.

1. Problems with very little difference between their *no-split* and Δ-transition bandwidths, but for which HYBSBC provides a very efficient tridiagonalization.
2. Problems for which there is a large difference between the *no-split* and Δ-transition bandwidths.

ERIS1176 and 1138 BUS are chosen to represent the first class of problems, while BCSSTK11 and CAN 445 are selected from the second class.

Using `Trisymb` we examine the change in tridiagonalization flop requirements of each problem as transition bandwidths are perturbed from the Δ-transition bandwidth b^t. In addition to the HYBSBC reduction, we attempt 16 tridiagonalizations with fixed transition bandwidths of $b^t+offset$, where $offset= -8, -7, \ldots, -1, +1, \ldots, +7, +8$. For each problem, Figure 5.26 plots the cost of these reductions normalized by the flop requirements of HYBSBC ($offset= 0$). Offsets $-8, -7, \ldots$, and -4 produce invalid transition bandwidths for ERIS1176 ($b^t = 4$) and are not included in its plot. In addition, it is not possible to select a transition bandwidth for 1138 BUS lower than 11, the Δ-transition bandwidth, because the outermost nonzero diagonal of the corresponding intermediate matrix is full. Although these plots only consider a small subset of possible transition bandwidths, expanding the number of reductions conducted for each problem does not change the exhibited general trends in tridiagonalization cost.

The Δ-transition strategy clearly chooses the optimal transition bandwidth for both BCSSTK11 and 1138 BUS. During the reduction of ERIS1176 HYBSBC transfers to R-S one diagonal prematurely, but HYBSBC's cost is within 0.63% of the optimal symbolic reduction's flop requirements. Technically, HYBSBC also transfers control of CAN 445's reduction to R-S one diagonal too soon. The difference between the cost of the HYBSBC and optimal reductions, however, is an insignificant 0.0015%. The general characteristics

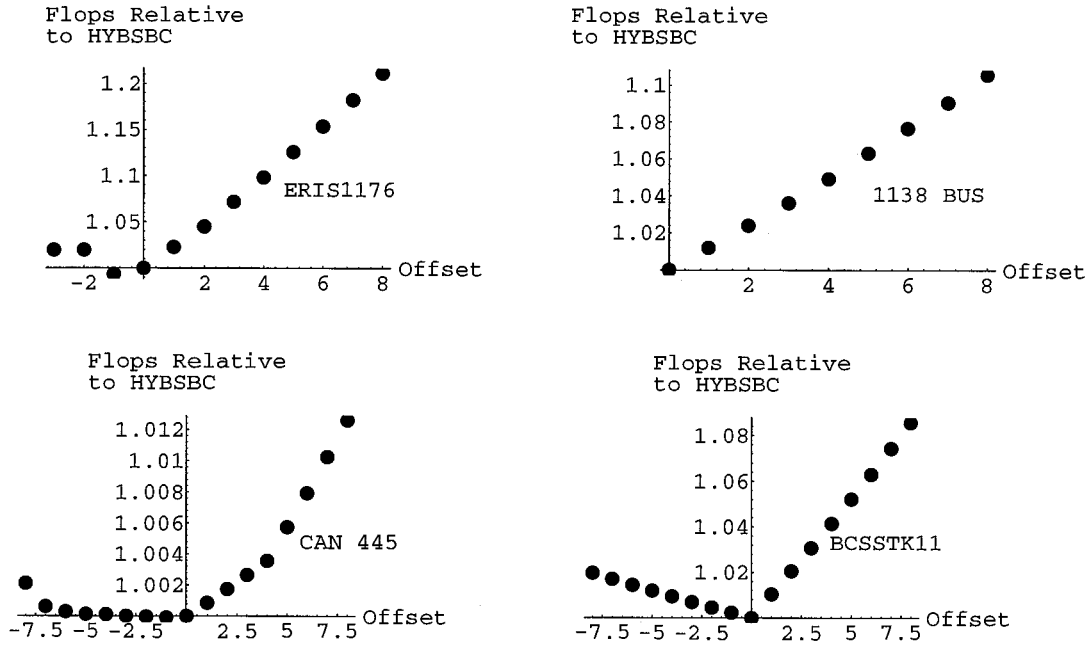


Figure 5.26: Variation in Tridiagonalization Flop Counts with the Transition Bandwidth Offset from the Δ-Transition Bandwidth, b^t.

of the Δ-transition strategy exhibited by the plots of these four problems' reduction costs are typical of the hybrid algorithm's reduction of other Harwell–Boeing problems. With respect to flop requirements, these results demonstrate that in practice the Δ-transition strategy selects optimal transition bandwidths for practical sparse problems. The optimality of the Δ-transition strategy will be further investigated in Section 5.4.3 using numerical implementations of the Hybrid Split Bandwidth Contraction algorithm.

5.3 Numerical Implementation of SBC and HYBSBC

This section provides an overview of implementations of the Split Bandwidth Contraction and Hybrid Split Bandwidth Contraction algorithms. The following discussion relies on Section 4.3's description of aspects of these implementations inherited from the BC and HYBBC implementations, while concentrating upon the implementation of features unique

to SBC and HYBSBC.

5.3.1 Implementation Basics

Once again, we rely upon existing preordering algorithm implementations to conduct the preordering phase of both SBC and HYBSBC, and direct our attention to the implementation of each reduction algorithm's second phase.

The numerical implementation of HYBSBC consists of two FORTRAN modules. The first module is a completely redesigned group of routines implementing SBC's bidirectional elimination techniques using fast Givens transformations. Within this module, the *find_split-point* routine identifies minimum displacement split-points by starting at the midpoint of each diagonal and searching in both directions simultaneously for the closest zero entry. When two split-points of equal minimal displacement are found, it uses a global variable indicating the region from which the last diagonal's split-point was selected to choose the best split-point with damped tiebreaking. The cost of conducting this sequential search before the elimination of each diagonal is insignificant relative to the cost of the entire reduction. To permit meaningful comparison of numeric and symbolic HYBSBC reductions, the *find_split-point* routine was carefully implemented to ensure the symbolic and numeric versions of SBC select the same split-point sequence for an identical series of sparsity structures. Of course, numerical cancellation is not modeled by `Trisymb` and for some practical problems the structure of the intermediate matrices encountered by the symbolic and numeric routines may differ despite equivalent split-point selection routines.

HYBSBC's Δ-transition strategy was implemented as a separate subroutine closely following Figure 5.24. It uses a *threshold* of 1 for problems whose permuted bandwidth is greater than $n/3$. As predicted in Sections 5.2.2.2 and 5.2.2.3, the fraction of the total tridiagonalization time spent in the Δ-transition routine is insignificant. As an example, ERIS1176 requires 96 calls to *transition*(n, b^c, sp), but the total number of CPU seconds used by these calls is below the precision of the built-in FORTRAN timing routine `etime`. Artificially increasing the number of calls by 500, *transition*(n, b^c, sp) requires a total of

0.01 second. This represents less than 0.02% of the total CPU requirements of ERIS1176's tridiagonalization.

Using its implementation of the Δ-transition strategy, HYBSBC regulates the transition to routines in the second module implementing the column-oriented phase of the reduction. The central routine in this module is a modified version of EISPACK's FORTRAN routine BANDR (an R-S code) employed by HYBBC. Once again, the speed with which a typical sparse band fills during an R-S reduction makes it impractical to use a sparsely banded R-S code for this portion of HYBSBC.

We also developed a separate implementation of SBC capable of partial bandwidth contractions and complete tridiagonalizations. It is essentially identical to HYBSBC's SBC module, but eliminates the switch to R-S using the Δ-transition strategy. Alternatively, it continues the contraction until reaching the desired final bandwidth. If it encounters a dense outermost diagonal during the reduction, control transfers to a special implementation of BC, which omits initializations, to complete the reduction.

As defined in Sections 5.1.3 and 5.2.1, SBC and HYBSBC incorporate the three techniques listed in Section 3.5.2 for exploiting band sparsity during the elimination of individual nonzero entries. The experimental study described in Section 4.3.2, however, clearly demonstrates that band sparsity is best exploited at the transformation level by identifying entries that are already zero or that the algorithm can eliminate with an adjacent row/column exchange. Performing sparse transformations, however, is shown to be without benefit for BC or HYBBC implementations, considering the storage and computational overhead required by a sparse data structure. This analysis applies equally well to both the SBC and HYBSBC algorithms, and neither algorithm's implementation pursues sparse transformations.

Without the need for a special data structure to accommodate sparse transformations, the SBC and HYBSBC implementations both use the densely banded data structure described in Section 4.3.3. As a result, the storage requirements of SBC and HYBSBC are essentially equivalent to the BANDR, BC, and HYBBC implementations, allowing the ex-

perimental analysis of Section 5.4 to concentrate upon the CPU requirements of SBC and HYBSBC.

5.3.2 Rescaling Techniques for SBC and HYBSBC

As discussed in Section 4.3.4, the use of fast Givens transformations necessitates periodic rescaling of the diagonal matrix, D, associated with the reduction. As for previous implementations, the SBC and HYBSBC implementations avoid difficulties with overflow by carefully restricting the worst case growth of D's entries between rescaling episodes and by selecting an appropriate rescaling constant for entries experiencing too much growth. Once again, for accounting purposes the rescaling schemes assume all entries above and below the split-point are nonzero.

SBC's implementation modifies the rescaling procedures in BC's implementation to reduce the number and scope of rescaling episodes by exploiting the decreased bulge chasing requirements of bidirectional elimination techniques. Specifically, SBC's rescaling procedures attempt to exploit the isolation of the shortened bulge chasing sequences above and below the split-point. For example, suppose SBC's procedures monitoring rescaling record that a diagonal's elimination modifies individual main diagonal entries above and below the split-point with at most abv and blw nontrivial transformations respectively. At the start of the next diagonal's elimination, the monitoring procedure assumes that in the worst case a single entry of D has been modified by $\max(abv, blw)$ transformations since the last rescaling. In contrast, during a BC reduction its monitoring procedure would be forced to assume that in the worst case an entry of D had been modified by approximately $abv+blw$ transformations. In addition, after the first rescaling during a diagonal's reduction by SBC, subsequent rescalings before the diagonal's elimination is complete are restricted to smaller and smaller regions above and/or below the split-point. In these cases SBC experiences shorter rescaling sequences during which fewer of D's entries are checked and possibly rescaled. Given these savings, the cost of SBC's rescaling procedures are never more than for BC and are typically substantially less.

The HYBSBC implementation inherits the rescaling techniques of its constituent algorithms with one modification. Just before transferring from SBC to R-S, the hybrid algorithm invokes a rescaling episode to ensure a successful transition between the rescaling procedures of the two modules.

5.4 Experimentation with Numerical Implementations

This section describes extensive testing of Section 5.3's implementations of the Split Bandwidth Contraction and Hybrid Split Bandwidth Contraction algorithms. Unless otherwise specified, each problem is preordered to reduce bandwidth using Lewis's implementation of GPS [Lew82], and tridiagonalized in the same testing environment described in Section 4.4.1. For future reference we also note that the Sun workstation used for testing has a 64 KByte external cache in addition to 16 MBytes of main memory. For these experiments the Δ-transition strategy uses a threshold of 1.0. After analyzing test results comparing our numerical implementations of SBC and HYBSBC to BC, HYBBC, and BANDR, we conclude by investigating the precision with which the Δ-transition strategy regulates HYBSBC's transition bandwidth for practical numerical reductions.

5.4.1 SBC Numerical Testing Results

The ability of SBC to efficiently execute a partial bandwidth contraction of a sparsely banded problem is crucial to the success of HYBSBC. As discussed previously, efficient partial SBC reductions can also play an important role as a preprocessing technique for other banded eigenvalue routines, such as BQR, or parallel implementations of Lang type tridiagonalizations, or sparse linear systems solvers. Consequently, we first analyze the partial bandwidth contraction performance of SBC relative to BC. The final bandwidth for each problem's contraction is the Δ-transition bandwidth chosen by the numerical implementation of HYBSBC.

Theoretical analysis predicts that SBC can at least halve the computational requirements of a single diagonal's elimination when split-points are well centred, but split-points

in the outermost nonzero diagonal have increasingly higher displacements as a typical reduction proceeds. Despite this tendency the numerical SBC implementation executes partial bandwidth contractions very efficiently. In fact, the numerical SBC routine performs partial contractions even more successfully relative to BC than the substantial improvements predicted by similar experiments with symbolic implementations of BC and SBC in Section 5.1.5.

For the 70 problems from the Harwell–Boeing test suite with $n > 400$, SBC requires on average 24% fewer CPU seconds than BC to perform the partial reductions. From this subset of problems SBC reduces the CPU requirements of 22 problems by more than 35% and reductions of 45 to 55% are common. Reductions ranged as high as 76% for the sparse problem DWT 1007.

Table 5.9 summarizes the transformation and CPU second requirements of 10 selected partial BC and SBC contractions. The table's columns provide similar information to Table 5.5's summaries of symbolic reductions, except the final three columns summarize CPU second requirements instead of flop counts. Comparison of Tables 5.5 and 5.9 reveals that the symbolic codes provide relatively accurate assessments of SBC's actual computational requirements relative to BC.

As the symbolic routines predict, however, SBC does not provide a faster partial contraction than BC for all test problems. For example, the numerical SBC and BC reductions of the "LSHP" problems are essentially identical. The symbolic analysis predicts that for 8 of the 115 test problems the computational requirements of SBC should be higher than BC. Using the numerical routines, however, SBC's performance degradation is lower than anticipated for these problems and fewer of them remain in this category. Although the symbolic implementations typically predict the actual computational requirements very closely, their flop counts do not encompass all reduction costs. For example, they do not include the cost of performing row/column exchanges. When these additional costs are taken into consideration by the timings of numerical routines, SBC is significantly slower than BC for only 3 problems, DWT 918, BCSSTK28 and BCSSTK09.

Name n, b^{GPS}, b^f	Transformation Totals (bulge chasing, row/col. exch.) BC	SBC	Time (sec) BC	SBC	%CPU Reduction
NOS5 468, 88, 49	21006 (14109,3319)	14285 (7663,3204)	4.29	2.77	35.4%
BCSSTK19 817, 18, 3	364569 (357001,1255)	195902 (188242,281)	13.3	6.41	51.8%
DWT 1007 1007, 34, 19	76908 (71200,2686)	21553 (17952,70)	7.1	1.7	76.1%
CAN 1072 1072, 156, 48	283858 (235492,16888)	178412 (126802,16194)	77.6	43.4	44.0%
1138 BUS 1138, 126, 10	770172 (715203,71691)	498302 (438850,60228)	80.5	49.3	38.8%
ERIS1176 1176, 100, 4	1295336 (1241984,19140)	620025 (569094,69630)	110.5	44.0	60.2%
BCSPWR06 1454, 100, 12	1299517 (1234992,90699)	636455 (573608,67932)	154.4	62.2	59.7%
BCSSTK11 1473, 62, 33	354157 (326321,1402)	184806 (157849,1078)	63.1	28.3	55.2%
PLAT 1919 1919, 80, 30	887007 (826386,31693)	537775 (478445,9523)	173.6	91.7	47.2%
BCSSTK26 1922, 245, 12	3983410 (3730049,151257)	3105826 (2871728,91367)	907.6	579.3	36.2%

Table 5.9: Selected Partial BC and SBC Reduction Timing Summaries

The symbolic routines predict that due to peculiar sparsity characteristics of DWT 918 SBC requires 13.8% more flops than BC. Although the symbolic analysis correctly predicts nontrivial transformations and row/column exchanges, SBC needs only 9.0% additional CPU seconds. Similarly, the symbolic routines correctly predict that special characteristics of BCSSTK28 result in higher transformation totals for SBC and a shift from row/column exchanges to nontrivial transformations, resulting in a 6.8% increase in CPU seconds for SBC relative to BC.

Finally, the symbolic analysis of BCSSTK09 predicts SBC requires more than 2.5 times as many flops as BC's partial contraction of the same problem. As discussed in Section 5.1.5, this large discrepancy in requirements is due to a moderate increase in transformation totals for SBC with a dramatically lower proportion of row/column exchanges. When BC's large number of row/column exchanges are no longer without cost in the numerical

implementations, the discrepancy in CPU seconds is reduced to 16.5%.

5.4.2 HYBSBC Numerical Testing Results

The Δ-transition strategy permits numerical implementations of HYBSBC to effectively exploit SBC's bidirectional elimination techniques and significantly improve upon both HYBBC and BANDR tridiagonalizations.

Relative to HYBBC, HYBSBC provides additional savings in CPU time, ranging as high as 52%, for 98 of the 115 test problems. HYBSBC achieves the largest timing reductions on problems which the SBC phase is especially efficient and comprises a significant proportion of the complete reduction cost. For the 70 problems with more than 400 nodes, HYBSBC requires on average 12.7% fewer CPU seconds than HYBBC. HYBSBC tridiagonalizations have significantly higher timings than HYBBC for only 5 sparse problems, which require between 2 and 10.6% additional CPU seconds. The worst offenders in this group are DWT 918, BCSSTK09 and BCSSTK28, which were previously identified as problems with peculiar sparsity structures resulting in sub-optimal partial SBC contractions. Thus, there remain problems for which HYBBC remains the optimal reduction. At present, however, we have not discovered easily identifiable characteristics common to the few problems for which HYBBC is superior and must analyze each problem's characteristics individually.

The additional gains enjoyed by HYBSBC relative to HYBBC make it an impressive alternative to EISPACK's BANDR. In fact, for every test problem of order greater than 100 HYBSBC's tridiagonalization requires fewer CPU seconds than BANDR. HYBSBC tridiagonalizes one problem, ERIS1176, in 1/5 of BANDR's time. For the 70 problems in the test suite with $n > 400$, HYBSBC requires on average 38.7% fewer CPU seconds than BANDR. In comparison, HYBBC's mean reduction is 31.1% for the same problem set. The histogram in Figure 5.27 illustrates the wide distribution of CPU reductions HYBSBC achieves for this group of 70 problems.

Table 4.4 in Chapter 4 summarizes the computational requirements of 20 test problems for which HYBBC is especially successful. For the same group of problems Table 5.10 sum-

	Transformation Total ($\times 10^3$) (bulge chasing, row/col. exch.)			Tridiagonalization Times (sec)			
Name n, b^{GPS}, b^t	BANDR	HYBBC	HYBSBC	BANDR	HYBBC (BC, R-S)	HYBSBC (SBC, R-S)	%CPU Reduct.
685 BUS 685, 78, 10	226.2 (176.7,0)	594.7 (570.3,23.51)	353.5 (326.9,21.82)	67.7	26.0 (22.5,3.5)	21.1 (11.9,9.2)	68.8%
GR 30 30 900, 49,20	392.1 (350.2,0)	425.5 (394.1,24.76)	556.5 (524.6,0.730)	77.9	57.9 (4.1,53.8)	53.5 (18.6,34.8)	31.4%
NOS3 960, 65, 40	449.6 (390.3,0)	486.9 (442.5,22.52)	480.9 (436.5,22.53)	128.6	83.1 (5.9,77.2)	81.9 (4.75,77.2)	36.3%
DWT 1005 1005, 106, 19	487.9 (388.4,0)	1352 (1299,24.93)	813.9 (759.3,18.69)	209.1	86.8 (75.1,11.7)	81.7 (40.1,41.6)	60.9%
CAN 1054 1054, 112, 34	547.5 (436.9,0)	1573 (1500,13.85)	742.3 (665.2,9.623)	237.5	147.6 (128.1,19.5)	114.4 (37.3,77.1)	51.8%
CAN 1072 1072, 156, 48	567.1 (413.5,0)	1744 (1647,17.76)	740.0 (639.2,16.19)	331.9	189.2 (169.0,20.2)	162.9 (43.5,119.4)	50.9%
BCSSTK09 1083, 95, 57	567.6 (479.4,0)	642.1 (568.2,38.86)	644.5 (567.8,6.710)	234.3	150.8 (15.4,135.4)	153.7 (18.0,135.7)	34.4%
1138 BUS 1138, 126, 10	633.0 (499.1,0)	1991 (1925,73.73)	1076 (1006,60.23)	305.3	112.2 (112.2,0)	77.1 (49.3,27.8)	74.8%
ERIS1176 1176, 100, 4	643.3 (534.5,0)	2007 (1950,19.47)	1136 (1082,69.63)	291.3	120.1 (120.1,0)	57.7 (44.0,13.7)	80.2%
DWT 1242 1242, 91, 31	752.4 (645.0,0)	1623 (1560,38.92)	940.2 (871.7,15.44)	278.6	142.6 (101.5,41.1)	128.0 (29.3,98.7)	54.0%
BCSPWR07 1612, 103, 5	1271 (1113,0)	3998 (3910,79.14)	2370 (2274,74.48)	604.4	232.8 (232.8,0)	151.0 (117.8,33.2)	75.0%
BCSPWR09 1723, 116, 9	1451 (1261,0)	4771 (4665,117.8)	2624 (2505,119.6)	730.4	313.1 (313.1,0)	207.1 (142.8,64.3)	71.6%
PLAT1919 1919, 80, 30	1801 (1653,0)	4399 (4283,34.79)	2316 (2201,9.523)	706.7	424.6 (335.6,89.0)	346.5 (91.7,254.8)	51.0%
BCSSTK26 1922, 245, 12	1827 (1388,0)	5008 (4734,151.2)	4797 (4542,91.37)	1864	1001 (839.5,161.6)	685.8 (579.3,106.5)	63.2%
DWT 2680 2680, 65, 28	3480 (3311,0)	9025 (8912,26.66)	4073 (3957,21.10)	1103	664.9 (548.3,116.6)	585.6 (93.5,492.1)	46.9%
ZENIOS 2873, 30, 1	5.283 (2.136,0)	5.532 (4.119,3.976)	5.530 (4.118,3.974)	1.25	0.86 (0.86,0)	0.75 (0.75,0)	40.0%
SSTMODEL 3345, 82, 1	3603 (3385,0)	11850 (11700,87.27)	10144 (10004,70.25)	1450	857.8 (857.8,0)	677.9 (677.9,0)	53.2%
LSHP3466 3466, 61, 29	5854 (5648,0)	19240 (19070,25.43)	8481 (8309,24.03)	1791	1442 (1346,95.8)	1405 (525.8,879.4)	21.5%
BCSSTK24 3562, 312, 102	6302 (5243,0)	14880 (14180,231.3)	9458 (8728,127.4)	8990	5329 (4000,1329)	5102 (1954,3148)	43.3%
BCSSTK28 4410, 323, 86	9646 (8280,0)	28100 (27080,603.3)	17716 (16618,350.9)	14643	9118 (7916,1202)	9712 (5534,4178)	33.7%

Table 5.10: Selected HYBSBC, BANDR and HYBBC Tridiagonalization Summaries

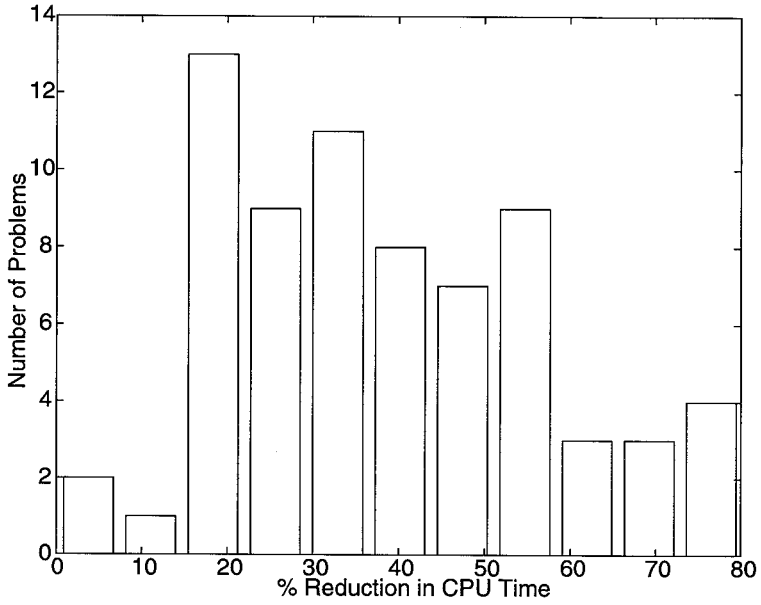


Figure 5.27: The Distribution of HYBSBC's Improved Reduction Performance Relative to BANDR for Problems with $n > 400$

marizes the tridiagonalization requirements of HYBSBC, as well as BANDR and HYBBC to facilitate comparison. The value of b^t listed for each problem is the bandwidth selected by HYBSBC's Δ-transition strategy. Table 4.4 provides HYBBC's transition bandwidth. The final column of Table 5.10 reports the reduction in CPU time HYBSBC achieves relative to BANDR. For this group of 20 problems HYBSBC exhibits an impressive mean reduction in CPU time of 52.2%, or on average is 2.09 times faster than BANDR.

As discussed during this chapter, the performance of SBC, and consequently HYBSBC, is dependent upon problem specific sparsity structures. As for BC, the class of sparsity structures particularly suited to the Split Bandwidth Contraction algorithm concentrates nonzeros near the main diagonal and exhibits increased sparsity towards the outermost diagonals. In addition, SBC ideally prefers nonzeros to be evenly distributed across the midpoint of each diagonal and concentrated towards its two ends.

As discussed in Section 4.4.4, GPS cannot always preorder a problem to simultaneously reduce bandwidth and produce desirable sparsity structure characteristics in the permuted

matrix. In addition to the GPS–HYBSBC experimentation previously discussed, we also conducted HYBSBC tridiagonalizations using RCM and GK preorderings. Although SBC efficiently exploits the increased peripheral sparsity often presented by these preorderings, the typically higher initial bandwidths they produce does not permit HYBSBC's overall performance to improve. As for the Bandwidth Contraction algorithm the primary preordering objective remains the reduction of bandwidth. As outlined in Section 4.4.4 for BC, investigating the ability of preorderings to produce small bandwidth sparsity structures conducive to SBC reductions is an interesting avenue of future research.

In conclusion we note one additional advantage of the HYBSBC algorithm. The Δ-transition strategy and bidirectional elimination techniques both contribute towards the lower transformation totals observed for HYBSBC relative to HYBBC. On average HYBSBC requires 1.4 times as many transformations as R-S, while HYBBC requires 2.25. Reducing transformation totals is of interest if we replace fast Givens transformations with standard Givens transformations, which have higher fixed costs associated with each transformation. In addition, it is desirable to reduce transformation totals if we want to accumulate transformations to find eigenvectors. Future research will investigate transition strategies that attempt to provide efficient reductions while minimizing transformation usage.

5.4.3 Δ-Transition Optimality

Using `Trisymb` symbolic implementations, Section 5.2.2.3 thoroughly investigated the precision with which the Δ-transition strategy regulates a reduction's transition bandwidth. In particular, Figure 5.26 examines the change in tridiagonalization flop requirements of four representative problems as their transition bandwidths are perturbed from the Δ-transition strategy's selection. These experiments rely completely upon the symbolic analysis of flop requirements to predict tridiagonalization performance. In this section we confirm this favorable analysis by conducting similar experiments with a special numerical implementation of the Hybrid Split Bandwidth Contraction algorithm that permits the user to specify the transition bandwidth.

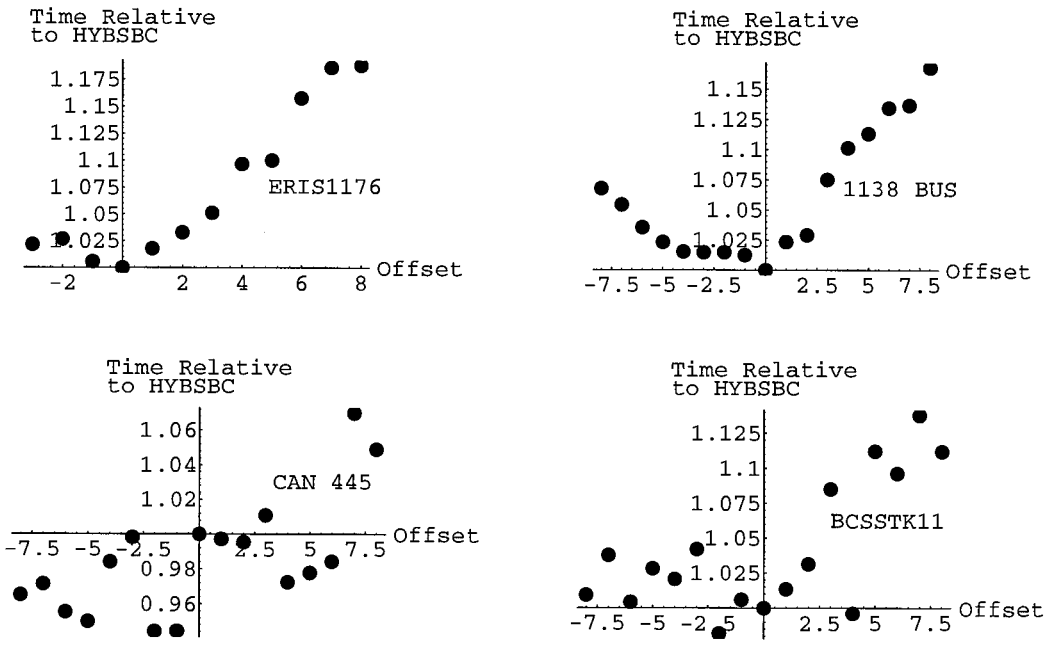


Figure 5.28: Variation in Tridiagonalization Time with the Transition Bandwidth Offset from the Δ-Transition Bandwidth

Once again we study the effects upon tridiagonalization timings of perturbing the transition bandwidth of the four Harwell–Boeing problems ERIS1176, 1138 BUS, CAN 445 and BCSSTK11. Figure 5.28 plots the cost of each problem's perturbed reductions normalized by the CPU second requirements of HYBSBC using the Δ-transition bandwidth (offset=0). The reduction time of each problem with a particular transition bandwidth is actually the mean of 3 or more reduction timings.

These plots clearly show that the Δ-transition strategy selects the optimal transition bandwidth for ERIS1176 and 1138 BUS. As expected, the plots of these problems are relatively smooth and follow the same general trends as their symbolic counterparts. The plots of the other two problems, however, are not smooth and tridiagonalization times often fluctuate wildly from one transition bandwidth to the next. Despite these irregularities the Δ-transition strategy chooses a near optimal transition bandwidth for BCSSTK11, while for

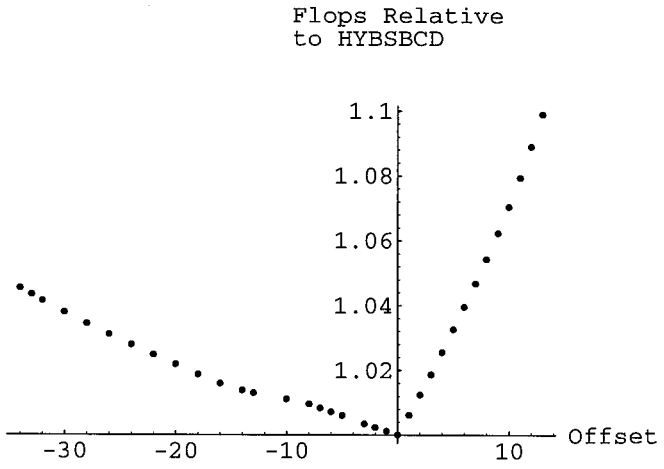


Figure 5.29: Tridiagonalization Flop Counts for CAN 634

CAN 445 the Δ-transition bandwidth is within 5.6% of the optimal reduction time. If SBC had been allowed to eliminate one additional diagonal from CAN 445, the optimal transition bandwidth would have been used. These results demonstrate that the Δ-transition strategy successfully picks optimal or near optimal transition bandwidths, but the irregular nature of CAN 445 and BCSSTK11's plots warrants further investigation.

To understand the source of the irregularities in Figure 5.28's plots, we chose to study more closely the tridiagonalization costs of a fifth sparse problem, CAN 634, as a function of transition bandwidth. Figure 5.29 plots the relative flop requirements of CAN 634's tridiagonalization against a large range of transition bandwidths offsets. As expected, the plot is smooth and the almost linear curves emanating from the origin indicate the Δ-transition bandwidth is computationally optimal. The nature of this plot is radically different from its numeric counterpart in Figure 5.30. Although this plot is smoother than the plot for CAN 445, the magnitude of the timing fluctuations is even larger. According to this plot the Δ-transition bandwidth results in a tridiagonalization whose reduction time is 10% higher than optimal. Although the Δ-transition strategy is foiled by the timing irregularities of CAN 634 and CAN 445, the Δ-transition strategy does not usually have this much difficulty picking the optimal transition bandwidth. These two pathological cases

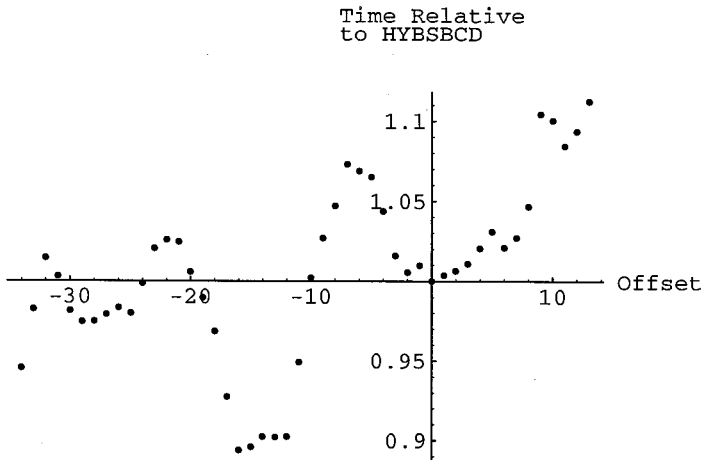


Figure 5.30: FORTRAN Tridiagonalization Timings for CAN 634 on a SPARCstation 2 with a 64KByte External Cache

are representative of less than 7 problems in the Harwell–Boeing test suite.

Fluctuations in integer and floating point operation counts do not change rapidly enough with transition bandwidth to account for these wild timing oscillations. We hypothesize that data access patterns are influencing reduction performance and producing the dramatic changes in CPU requirements. The densely banded data structure of HYBSBC stores each subdiagonal of the band's lower triangular portion in a separate column of a two dimensional double precision array. As we reduce a problem's transition bandwidth, the portion of the banded data structure referenced by the R-S phase shrinks. The elimination of one extra diagonal before switching to R-S means that R-S accesses $8n$ fewer bytes of memory. If eliminating one column of storage results in more of the data remaining in the cache during the R-S stage of the reduction, the performance of the tridiagonalization could dramatically increase as cache hits rise. As lower and lower transition bandwidths are considered data access patterns may change yet again, resulting in more cache misses and higher reduction times. In this fashion the effectiveness of the cache may change with different transition bandwidths, leading to the observed oscillatory nature of tridiagonalization timings.

To confirm this hypothesis we conducted different experiments with CAN 634 on ma-

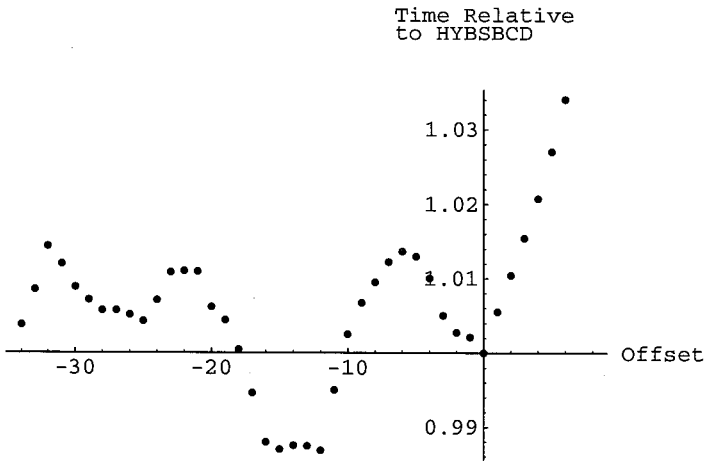


Figure 5.31: C Tridiagonalization Timings for CAN 634 on a SPARCstation 2 with a 64KByte External Cache

chines with different caching capabilities. For example, when the same reductions are performed on a SUN SPARCstation 10 with a 36 KByte on-chip cache and a 1 MByte external cache, the general characteristics of CAN 634's relative timings plot do not change, but the period of the oscillations is larger than for the smaller cache SPARCstation 2.

This result is consistent with our hypothesis, but to be certain of its validity we need to eliminate the effects of caching on timing altogether. Unfortunately, the SUN hardware does not permit users to turn off the cache. Alternatively, we conducted a series of experiments on a 50 MHz i486, with both an 8 KByte on-chip cache and a 256 KByte external cache, running release 2.5 of the Mach operating system. Due to the unavailability of a FORTRAN compiler on this machine, we created a C version of HYBSBC using the program *f2c* [FGMS93]. To ensure this conversion process did not change the general characteristics of tridiagonalization performance, we experimented with the C code version on a SUN SPARCstation 2, producing Figure 5.31. Although the magnitude of the oscillations have changed, the general trends of Figure 5.31's plot are identical to its FORTRAN counterpart.

Figure 5.32 plots normalized tridiagonalization times against transition bandwidth off-

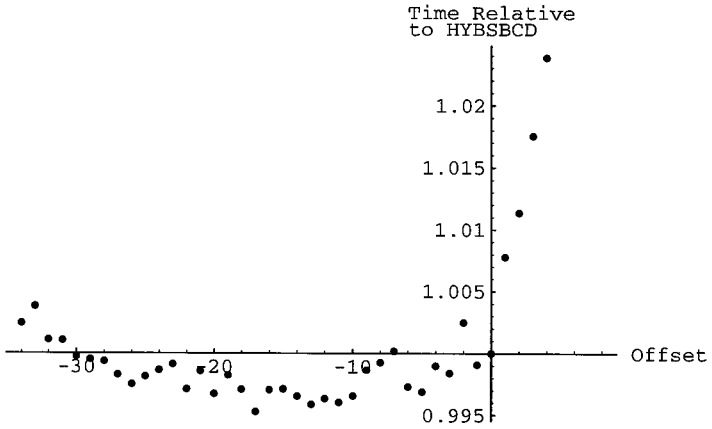


Figure 5.32: C Tridiagonalization Timings for CAN 634 on a 50 MHz i486 with a 8 KByte On-Chip Cache and a 256 KByte External Cache

sets for the C version of the code run on the i486 machine with caching enabled. The plot differs from Figure 5.31 significantly, but still exhibits the familiar erratic nature. Finally, Figure 5.33 plots normalized timings from the same machine with caching disabled. This plot is very close to the plot of normalized symbolic flop counts in Figure 5.29. It clearly shows that the erratic tridiagonalization performance is due to caching effects and that the Δ-transition strategy selects the computationally optimal transition bandwidth.

If a user desires to have the fastest possible tridiagonalization of a particular problem on a specific sequential architecture, the Δ-transition strategy may not always provide the optimal transition bandwidth. One might be able to fine-tune the Δ-transition strategy by creating a detailed cache model and analyzing the data access patterns of the Hybrid Split Bandwidth Contraction algorithm. Even if this difficult task is tractable, the approach is both machine and problem dependent. In addition, cache effects are sensitive to machine load. In fact, with more than one compute intensive process running during tridiagonalization testing, the erratic performance behavior will largely disappear as each plot approaches Figure 5.33. Thus, although fine-tuning the Δ-transition strategy is possible, it is not generally applicable. The Δ-transition strategy, however, provides a general scheme, which always selects an optimal or near optimal transition bandwidth for use in any sequential

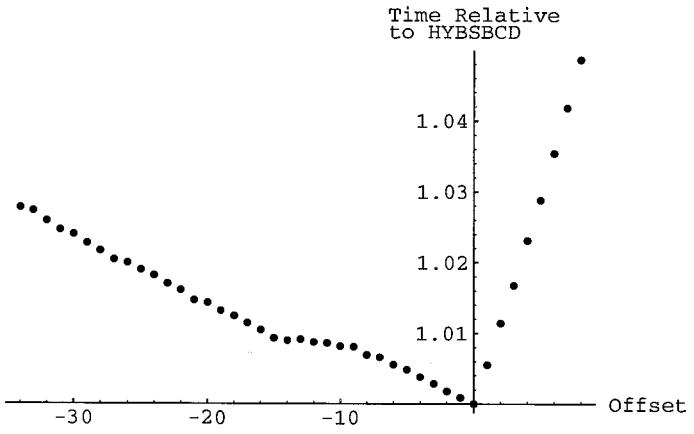


Figure 5.33: C Tridiagonalization Timings for CAN 634 on a 50 MHz i486 with Caching Disabled

computing environment. It is the best possible transition strategy with general applicability and, as the testing results of Section 5.4.2 demonstrate, it effectively contributes to HYBSBC's highly efficient tridiagonalizations.

Chapter 6

A Comparison of the Lanczos Algorithm with Direct Sparse Tridiagonalization

There continues to be widespread interest in the use of Lanczos-type algorithms [Lan50] for the solution of sparse eigenvalue problems. Lanczos algorithms are certainly well suited to finding a few eigenvalues from either end of the spectrum of a sparse symmetric matrix. This chapter, however, explores the ability of Lanczos algorithms to economically find moderately large subsets of eigenvalues or identify the complete spectrum, including eigenvalue multiplicities, of a sparse symmetric problem. To evaluate the relative success of applying Lanczos to these tasks we compare its resource requirements to the cost of isolating a sparse matrix's complete spectrum with HYBSBC and EISPACK's tridiagonal eigenvalue routine TQLRAT [SBD+76].

We begin the chapter with a brief overview of the mathematics underlying the basic Lanczos iteration and identify the difficulties associated with the implementation of Lanczos in this simple form. (For a rigorous discussion of the Lanczos method consult [GV89] or [Par80].) With a general understanding of the Lanczos iteration, we then survey the different techniques employed by practical Lanczos implementations. Although this survey is far from exhaustive, it attempts to touch upon the techniques of the dominant Lanczos methods. The presentation of each Lanczos approach includes a theoretical discussion

of its ability to compute efficiently the complete spectrum of a sparse symmetric problem. Finally, the chapter's last section summarizes experiments applying two well regarded Lanczos codes to practical sparse problems from the Harwell–Boeing test suite. We analyze the resource requirements of these codes to compute a sparse problem's complete spectrum, or moderately large subsets of its eigenvalues. For these sparse eigenvalue problems, direct methods based on HYBSBC tridiagonalizations compare very favorably to iterative Lanczos techniques.

6.1 Lanczos Basics

Using a three term recurrence, the Lanczos algorithm directly computes tridiagonal matrices whose eigenvalues approximate some of the original matrix's eigenvalues. Essentially following the notation of Parlett [Par80], the simple form of the Lanczos algorithm for an $n \times n$ matrix A, and arbitrary starting vector r_0, is described by the following iteration. Let q_0 be the zero vector and $\beta_0 = \| r_0 \|$. All vectors are of length n.

For $j = 1, 2, \ldots$

1. $q_j = r_{j-1}/\beta_{j-1}$
2. $u_j = Aq_j - \beta_{j-1}q_{j-1}$
3. $\alpha_j = q_j^T u_j$
4. $r_j = u_j - \alpha_j q_j$
5. $\beta_j = \| r_j \|$

Each iteration of the Lanczos algorithm produces a new Lanczos vector q_j and scalars β_j and α_j. It is not difficult to understand why the Lanczos iteration is attractive to sparse matrix researchers. Lanczos does not modify the original matrix and sparsity can be easily exploited during the formation of the matrix-vector products Aq_j.

Assume that all Lanczos vectors are collected into a single matrix

$$Q_j = \{q_1, q_2, \ldots, q_j\} \tag{6.1}$$

and that the scalars α_j and β_j are used to form the tridiagonal matrix T_j.

$$T_j = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & & \\ & & \ddots & \beta_{j-1} \\ & & \beta_{j-1} & \alpha_j \end{bmatrix}$$

It can be shown that

$$AQ_j - Q_j T_j = r_j e_j^{\mathrm{T}}, \tag{6.2}$$

where $e_j^{\mathrm{T}} = (0, \ldots, 0, 1)$. If exact arithmetic is used to perform the computations of each Lanczos iteration, the Lanczos vectors are by definition orthogonal. Thus $Q_j^T Q_j = I_j$ and $Q_j^T r_j = 0$, and it can be shown that

$$T_j = Q_j^T A Q_j. \tag{6.3}$$

Using the Rayleigh-Ritz procedure, the tridiagonal matrix T_j can be used to find approximations to the eigenvalue-eigenvector pairs of A from the subspace $\{q_1, \ldots, q_j\}$. Suppose that eigenvalue-eigenvector pairs of T_j are $(\theta_i, s_i), i = 1, \ldots, j$. The Ritz vectors $y_i = Q_j s_i$ and corresponding Ritz values θ_i approximate eigenpairs of A. T_j's eigenvalues tend to exhibit the quickest convergence to the extremal eigenvalues of A, making the Lanczos algorithm particularly well suited to finding eigenvalues at either end of A's spectrum.

With exact arithmetic, the Lanczos algorithm continues until for some $k \leq n$ $\beta_k = 0$ and the orthogonal Lanczos vectors $\{q_1, \ldots, q_k\}$ span the invariant subspace of A containing the starting vector. Equivalently, the iteration continues until $k = \mathrm{rank}(K(A, q_1, n))$*. At this stage the Ritz pairs $(\theta_i, y_i), i = 1, \ldots, k$, form an exact subset of A's eigenpairs. If $k < n$, one can restart the iterative process by choosing a starting vector orthogonal to all previous Lanczos vectors.

In practice, when using finite precision arithmetic Lanczos encounters difficulties related to the loss of orthogonality between Lanczos vectors, resulting from numerical cancellation during the computation of r_j. This phenomenon complicates the algorithm's termination and the relationship between Ritz pairs and the eigenvalue-eigenvector pairs of A. For

*$K(A, q_1, n)$ is the Krylov subspace spanned by $\{q_1, Aq_1, \ldots, A^{n-1}q_1\}$.

example, in practice a straightforward implementation of the basic Lanczos algorithm does not terminate. It keeps producing duplicates of previously converged Ritz values indefinitely. In fact, the algorithm may even duplicate eigenvalues before all other eigenvalues of A have been identified by a converged Ritz value. In addition, without modifying the simple iteration, Lanczos is theoretically able to find at most a single eigenvector for each distinct eigenvalue. The eigenvector computed by Lanczos is the projection of the starting vector onto the invariant subspace of the corresponding eigenvalue. As a result, in its simplest form Lanczos is unable to determine the multiplicity of eigenvalues.

Many researchers have suggested modifications of the Lanczos algorithm to overcome these difficulties. The following subsections provide brief overviews of the most popular of these techniques. We evaluate each variant of the Lanczos algorithm in terms of its ability to isolate efficiently the complete spectrum of a sparse problem.

6.2 Lanczos With Full Reorthogonalization

The first technique resolving the difficulties of the simple Lanczos algorithm actively maintains strict orthogonality among the Lanczos vectors. As each new Lanczos vector is constructed it is orthogonalized with respect to all previously computed Lanczos vectors.

$$r_j = r_j - \sum_{i=1}^{j} (q_i^T r_j) q_i \tag{6.4}$$

Using n iterations of this modified Lanczos method creates an $n \times n$ tridiagonal matrix whose n Ritz values provide good approximations to all of A's eigenvalues with correct multiplicity. Of course, to successfully complete n iterations the Lanczos algorithm must be restarted with an orthogonal starting vector whenever $\beta_k = 0$.

Unfortunately, the costs of employing full reorthogonalization are very high. Independent of the sparsity of A, the reorthogonalization of Equation 6.4 alone requires $2n^3 + O(n^2)$ flops over n iterations and $O(n^2)$ storage. Although in practice full reorthogonalization is not often implemented precisely as shown in Equation 6.4 (see [GV89]), the economies of these alternative schemes do not change the leading term of the operation count. Clearly

this variant of the Lanczos algorithm cannot isolate the complete spectrum of a sparse problem with the efficiency of a HYBSBC+TQLRAT combination.

6.3 Selective or Partial Reorthogonalization Techniques

In response to the overwhelming cost of full reorthogonalization, researchers have proposed augmenting the simple Lanczos iteration with *selective reorthogonalization* [PS79] or *partial reorthogonalization* [Sim84]. Rather than insisting upon strict preservation of orthogonality, these schemes attempt to maintain semi-orthogonality amongst the Lanczos vectors. Both the selective and partial reorthogonalization schemes use clever techniques to monitor the loss of orthogonality and when necessary reorthogonalize Lanczos vectors against either converged Ritz vectors or a subset of the Lanczos vectors produced by previous iterations. Parlett and Scott [PS79] claim that an eigenvalue's multiplicity can be found by Lanczos schemes with selective reorthogonalization, but they appear to be relying upon rounding errors to sufficiently perturb an eigenvalue problem.

Either reorthogonalization technique allows the Lanczos iteration to successfully isolate a few of a problem's eigenvalues without incurring the excessive computational requirements of a complete reorthogonalization approach. If we use these variants of the Lanczos algorithm to isolate all the eigenvalues of a sparse symmetric matrix, however, the reorthogonalization costs of either scheme could approach those of full reorthogonalization. As an example, we consider the potential costs of selective reorthogonalization in more detail.

The selective orthogonalization scheme reorthogonalizes each new Lanczos vector against all Ritz vectors that have converged to an eigenvector of A, because the loss of orthogonality between Lanczos vectors goes hand in hand with convergence [PS79]. As a result, once this Lanczos algorithm detects the convergence of a Ritz vector, it must compute and store the Ritz vector even if the eigenvectors of A are not desired. During the algorithm's identification of a few eigenvalues, there are usually far fewer converged Ritz vectors than Lanczos vectors. This allows the algorithm to enjoy substantial savings in comparison to full reorthogonalization approaches. If all of A's eigenvalues are sought

using n Lanczos iterations, however, the number of converged Ritz vectors may eventually approach n. Suppose on average a newly converged Ritz vector must be computed after each Lanczos iteration. During the j^{th} iteration, the cost of computing the matrix-vector product $Q_j s_i$ to form Ritz vector y_i is $2jn$ flops. Thus the calculation of all n converged Ritz vectors requires

$$\sum_{j=1}^{n} 2jn = \frac{2n(n+1)n}{2} = (n^3 + n^2) \text{ flops.} \tag{6.5}$$

This result is a lower bound on the cost of computing the n Ritz vectors. Typically, the convergence of many Ritz vectors is delayed until much later in the Lanczos process than assumed by this analysis, forcing their computation from matrix-vector products of higher dimension.

Independent of A's sparsity structure, just the cost of calculating the Ritz vectors is potentially very high, but this operation count does not even include the significant computational requirements of the following portions of the Lanczos algorithm.

1. The cost of computing T_j's eigensystem when the selective reorthogonalization variant of the Lanczos algorithm pauses to assess the convergence of Ritz vectors.

2. The cost of the actual reorthogonalization process.

3. The cost of forming the matrix-vector products for the basic Lanczos iteration.

Even though the partial reorthogonalization approach does not require the computation of Ritz vectors, Simon [Sim84] reports that the costs of selective and partial reorthogonalization are comparable. Thus, it is unlikely that either of these Lanczos algorithms can compete with HYBSBC+TQLRAT when computing the complete spectrum of sparse matrices of moderate bandwidth. In addition to reorthogonalization that may approach $O(n^3)$, both methods potentially require $O(n^2)$ storage to record Lanczos vectors and converged Ritz vectors in the case of selective orthogonalization.

Because the rate of convergence of Ritz vectors is problem dependent, this section's analysis could not be as precise as the discussion of full reorthogonalization. Section 6.6.2

reports on experiments with a Lanczos code employing variants of both reorthogonalization techniques while identifying the complete spectrum of sparse problems.

6.4 Lanczos With No Reorthogonalization

As mentioned in Section 6.1's discussion of the simple Lanczos iteration, without the reorthogonalization of Lanczos vectors, T_j may have duplicate Ritz values corresponding to many of A's simple eigenvalues. At any particular stage of the Lanczos process duplicate Ritz values have converged with varying degrees of success to one of A's eigenvalues. Much effort has gone into the development of Lanczos type algorithms using no reorthogonalization, which can identify these *ghost* or *spurious* eigenvalues and deal with their existence. This section briefly outlines two spurious eigenvalue approaches by Cullum and Willoughby [CW79, CW80] and Parlett and Reid [PR81]. These algorithms are intended for the isolation of many or all distinct eigenvalues of a large sparse matrix. When A's eigenvectors are not required, both algorithms enjoy the distinct advantage of being able to discard Lanczos vectors once they are no longer required by the three term recurrence of the simple Lanczos iteration.

The two algorithms use different techniques to identify and manage spurious eigenvalues. Cullum and Willoughby's approach executes the simple Lanczos algorithm a user specified number of iterations m ($> n$), creating a large tridiagonal matrix T_m. They identify spurious or unconverged Ritz values by comparing the eigenvalues of T_m to a second tridiagonal matrix of order $(m-1)$, produced by deleting the first row and column of T_m. If the number and accuracy of the good eigenvalues identified is satisfactory the routine exits. Otherwise, a preset increment is added to m and the Lanczos iteration continues until the larger tridiagonal matrix T_m is found. Once again, the algorithm compares the eigenvalues of T_m and T_{m-1} to identify eigenvalues of A. The algorithm continues in this fashion until satisfying the eigenvalue request or reaching the maximum number of Lanczos iterations preset by the user. Cullum and Willoughby report that some problems may require more than $10n$ iterations to find all distinct eigenvalues.

Parlett and Reid's algorithm approaches the isolation of nonspurious eigenvalues using different techniques, which do not require the direct computation of the eigenvalues of large tridiagonal matrices. Once again, the algorithm executes the simple Lanczos iteration, but it uses a heuristic test for termination. Parlett and Reid's algorithm monitors the convergence of eigenvalues using recurrences on sets of points within and near the requested portion of the spectrum. To determine the actual positions of A's eigenvalues it employs rational interpolation procedures at these points. When all distinct eigenvalues are sought this technique typically requires far more than n simple Lanczos iterations. In the limited experimentation presented by Parlett and Reid, one problem required more than $5n$ iterations to identify all eigenvalues.

A number of difficulties are represented by both algorithms. First, there is no way to predict the final number of iterations required to find a problem's complete spectrum without executing the algorithms. In both cases, the computational requirements depend upon the distribution of eigenvalues within the spectrum of a particular sparse problem. In general, without the use of reorthogonalization both algorithms have difficulty with regions of a problem's spectrum in which the eigenvalues are not well separated. Secondly, neither algorithm can guarantee to find all distinct eigenvalues, because the algorithms may terminate before all eigenvalues are identified. In the Cullum and Willoughby approach there is no mechanism for deciding the number of iterations to use unless n nonspurious eigenvalues are identified. Despite encouraging practical experience, the heuristic termination techniques of Parlett and Reid cannot be guaranteed to succeed. In addition, both algorithms rely upon the assumption that the user-selected starting vector has a nontrivial projection onto the invariant subspace of all requested eigenvalues. If one uses a starting vector deficient in a particular eigenvalue's invariant subspace, the algorithms may never find this eigenvalue independent of the number of iterations taken.

Finally, both algorithms are designed to find eigenvalues without regard to multiplicity. Augmenting these methods with multiplicity checking would be unrealistic. For example, in the Cullum and Willoughby algorithm the multiplicity of eigenvalues could be determined by looking at the dimension of the subspace spanned by the eigenvectors corresponding to

converged eigenvalues. If the approach uses m Lanczos iterations it would require $O(mn)$ storage for the Lanczos vectors and at least $O(n^3)$ flops to compute the Ritz vectors. Alternatively, the multiplicity of each eigenvalue could be checked by using an LDL^T factorization of $A - \lambda I$ [BK77]. The cost of these factorizations are prohibitive considering the fill levels experienced during a typical sparse LDL^T factorization. Even if all $A - \lambda I$ matrices were banded and positive definite, each Cholesky factorization would require $O(b^2 n)$ flops, which translates into $O(b^2 n^2)$ flops for all factorizations in the worst case.

In summary, these spurious eigenvalue approaches cannot economically determine eigenvalue multiplicity and guarantee to find all distinct eigenvalues, and only very rough estimates of their resource requirements are possible prior to execution. Despite these difficulties, many researchers suggest the use of these algorithms for the practical computation of a problem's complete set of distinct eigenvalues. Section 6.6.1 reports on experiments comparing the efficiency of HYBSBC+TQLRAT to an implementation of Parlett and Reid's algorithm, which we selected to represent the spurious eigenvalue Lanczos codes.

6.5 Block Shift and Invert Lanczos

The spurious eigenvalue approaches discussed in the previous section are unable to provide the multiplicity of eigenvalues. The block Lanczos algorithm is a generalization of the single vector Lanczos iteration that overcomes these limitations. Rather than computing a single Lanczos vector, each iteration of block Lanczos produces a block of Lanczos vectors. Similarly, in place of tridiagonal matrices, the block Lanczos algorithm computes a sequence of block tridiagonal matrices whose eigenvalues approximate eigenvalues of A. The block Lanczos algorithm is able to compute the multiplicity of eigenvalues, as long as the *block size* is larger than the dimension of the invariant subspace of each requested eigenvalue. Without prior knowledge of a matrix's spectrum, however, the selection of an appropriate block size is difficult. The efficiency of block Lanczos algorithms may suffer from the selection of an overly large block size, because the algorithm's overhead is quadratic in the block size [Sco83, GLS94].

Another difficulty with algorithms based on the simple Lanczos iteration is their potentially slow convergence for poorly separated eigenvalues. For many eigenvalue applications the simple Lanczos iteration isolates eigenvalues efficiently from only one end of the matrix's spectrum. To overcome this difficulty Grimes, Lewis, and Simon [GLS94] combine the block Lanczos algorithm with the spectral transformation Lanczos method of Ericsson and Ruhe [ER80]. The goal of this sophisticated block shift and invert Lanczos algorithm is to provide a "black box" extraction routine for the generalized symmetric eigenvalue problem,

$$Hx = \lambda Mx. \tag{6.6}$$

Spectral transformations permit the block Lanczos algorithm to quickly isolate eigenvalues in any part of the spectrum of a generalized eigenvalue problem. Of course, this approach applies equally well to the ordinary eigenvalue problem, $Ax = \lambda x$, examined by this dissertation. In terms of this simpler problem, the spectral transformation works in the following manner. Suppose we want to find a group of A's eigenvalues near σ, which are poorly separated. If we invert the shifted eigenvalue problem $(A - \sigma I)$ using a sparse factorization, then the eigenvalues of interest are transformed into the largest and most well-separated eigenvalues of the following eigenvalue problem.

$$(A - \sigma I)^{-1}x = \mu x \tag{6.7}$$

The shift and invert transformation does not change the eigenvectors, but the eigenvalues μ associated with Equation 6.7 are related to those of A by $\mu = \frac{1}{\lambda - \sigma}$.

When more than a few eigenvalues close to a single shift are sought, a sequence of shifts can be used to speed up the convergence of the isolation process. Grimes et al provide a robust strategy to regulate the selection and use of shifts. With the selection of each new shift σ, the algorithm performs a sparse factorization of $(A - \sigma I)$ and initiates a new block Lanczos iteration.

In general, finite precision arithmetic implementations of block Lanczos algorithms suffer from the same difficulties as the simple Lanczos iteration, resulting from the loss of orthogonality amongst Lanczos vectors. To cope with the loss of orthogonality, Grimes,

Lewis and Simon's algorithm employs a novel combination of three reorthogonalization schemes.

1. *Local Reorthogonalization*
   At each iteration of the block shift and invert Lanczos algorithm, a local reorthogonalization is performed between the newest block of Lanczos vectors and the set of Lanczos vectors created by the previous step of Lanczos.

2. *Partial Reorthogonalization*
   To correct the global loss of orthogonality amongst Lanczos vectors, the algorithm uses a block version of Simon's *partial reorthogonalization* [Sim84].

3. *External Selective Reorthogonalization*
   External selective reorthogonalization keeps the current sequence of Lanczos vectors orthogonal to eigenvectors computed with previous shifts, preventing the recomputation of eigenvalues already identified. This novel scheme is motivated by the standard *selective orthogonalization* approach of Parlett and Scott [PS79].

Unlike *local reorthogonalization*, the second and third reorthogonalization techniques are used only when the algorithm's models of orthogonality loss indicate that their application is warranted. As discussed in Section 6.3, when seeking moderately large numbers of a problem's eigenvalues the introduction of reorthogonalization schemes may degrade algorithm efficiency to unacceptable levels.

The techniques outlined in this section form the basis of a robust Lanczos algorithm, which is capable of isolating groups of eigenvalues from any portion of a matrix's spectrum or computing a problem's complete spectrum, including eigenvalue multiplicities. Section 6.6.2 reports on experiments comparing the efficiency of HYBSBC+TQLRAT to an implementation of Grimes, Lewis and Simon's shift and invert block Lanczos algorithm.

6.6 Experimentation With Lanczos Codes

To complement the previous discussion, this section presents experimentation with the two well-known Lanczos codes. EA15D is a double precision implementation of Parlett and Reid's spurious eigenvalue Lanczos algorithm [PR81] from the Harwell Subroutine

Library [Har90]. Boeing Computer Services provided access to an implementation of the block shift and invert Lanczos algorithm [GLS94] from the Boeing Extended Mathematical Subprogram Library (BCSLIB-EXT) [Boe89]. We selected these implementations, because their capabilities appropriately match the eigenvalue problems addressed by this thesis and the codes use contrasting techniques to cope with the difficulties resulting from the loss of orthogonality amongst Lanczos vectors.

Applying these codes to practical sparse symmetric problems from the Harwell–Boeing test suite, we conduct two types of experiments. First, we investigate the costs of using Lanczos to isolate the complete spectrum of sparse problems. In addition, we explore the resource requirements of identifying subsets of eigenvalues, ranging in size from 12.5 to 100% of a sparse problem's complete spectrum. In all cases, we compare the resource requirements of the Lanczos codes to the costs of using HYBSBC and TQLRAT to compute a problem's entire spectrum, including eigenvalue multiplicities.

As for the majority of experimentation in Chapters 4 and 5, testing was conducted on a Sun SPARCstation 2 with 16 MBytes of main memory. All routines were compiled by the standard Sun FORTRAN compiler with full object code optimization requested. Prior to Hybrid Split Bandwidth Contraction tridiagonalizations, each problem was preordered to reduce bandwidth using Lewis's implementation of GPS [Lew82]. The CPU second timings of HYBSBC+TQLRAT and EA15D were produced using the built-in system routine `etime`, while the reported timings of the block shift and invert Lanczos code are provided by BCSLIB-EXT's own statistic gathering routines. The timings of HYBSBC+TQLRAT do not include the requirements of GPS, which are typically insignificant relative to CPU second totals for the entire process.

6.6.1 Harwell's EA15

EA15D provides the user with several variable parameters to control the Lanczos code's execution. Using these parameters we requested EA15D to provide the starting vector for the Lanczos iteration and to skip eigenvector computations. EA15D's parameter ACC

permits the user to request the precision, relative to the matrix's largest eigenvalue, to which the routine computes all eigenvalues. The user can assign ACC a specific relative accuracy or a negative value, which directs EA15D to find all requested eigenvalues to as much accuracy as the working precision reasonably allows. We chose to perform all experiments with EA15D twice, once requesting full accuracy and a second time with the relatively low accuracy of 10^{-5}.

The user must provide EA15D with a routine for the computation of $u + Av$, where A is the sparse matrix under analysis and u and v are vectors. To maximize the efficiency of computing the matrix-vector product Av, we chose to store A's nonzero entries in a row-oriented data structure using full adjacency lists. During a typical application of EA15D, we found that on average the computation of $u + Av$ at each iteration makes up 10–15% of the total CPU requirements, but ranges as high as 40% for specific eigenvalue problems and levels of accuracy. If storage is a paramount concern, we could easily exploit the symmetry of A by representing it with a lower adjacency data structure, requiring approximately half the storage. Consequently, when computing the storage requirements of EA15D we assume a lower adjacency structure is in use.

Unfortunately, the user must predict the total storage requirements of EA15D prior to its execution of an eigenvalue task. If allocations are too small, EA15D exits without reaching its goal and must be restarted with higher allocations of storage. In practice we found it very difficult to assess EA15D's storage requirements. In contrast, HYBSBC has the distinct advantage of knowing its storage requirements prior to commencing a reduction. In subsequent discussion of EA15D experimentation, we report the minimum storage needed by EA15D to successfully complete an eigenvalue task.

The EA15D code is designed for the isolation of a symmetric problem's complete spectrum, without regard to multiplicity. For 11 sparse problems from the Harwell–Boeing collection, Table 6.1 summarizes the costs of complete spectrum determination using HYBSBC+TQLRAT and EA15D with full accuracy requested. The table's first column records the problem's name, order and preordered bandwidth, as well as indicating if the Harwell–

	HYBSBC + TQLRAT		EA15D			
Name n,b^{GPS},val/noval	Time (CPU sec)	Storage (KBytes)	Time (CPU sec)	Storage (KBytes)	Eigenvalues Found	Lanczos Iterations
ERIS1176 1176,100,noval	65.9	969.02	4335.1	647.52	1176	23071
BCSPWR06 1454,100,noval	130.9	1198.1	6413.9	676.94	1455	27412
BCSSTK05 153,23,val	0.74	31.824	17.4	47.472	153	887
BCSSTK09 1083,95,val	159.9	849.07	>8181.6	>677.68	812	>25825
CAN.1054 1054,112,noval	121.3	969.68	1134.2	339.41	1054	8598
DWT.1005 1005,106,noval	88.8	876.36	1278.2	329.00	1005	9662
NOS3 960,65,val	87.6	522.24	>3039.5	>538.68	695	>19201
BCSSTK19 817,18,val	14.7	137.26	>3101.2	>271.60	56	>8171
1138.BUS 1138,126,val	84	1174.4	>12234	>429.98	69	>15249
BCSSTK26 1922,245,val	705.0	3813.3	>109022	>1070.1	15	>38441
PLAT1919 1919,80,val	361.9	1274.2	>21571.6	>889.16	10	>28786

Table 6.1: The Computational Cost of Complete Spectrum Determination with HYBSBC+TQLRAT and EA15D (Full Accuracy)

Boeing collection supplies the problem's nonzero values. As discussed in Section 2.2, if only the matrix's sparsity pattern is available, we assign a random value in the range $(0.0, 1.0]$ to each nonzero entry. During our testing we did not observe a correlation between EA15D performance and the use of randomly assigned or provided values. The next two columns of the table summarize the CPU seconds and storage requirements ($\sim 8n(b^{GPS}+3)$ bytes) of HYBSBC+TQLRAT. The table's final columns report EA15D's CPU and storage usage, the number of eigenvalues found, and the total number of Lanczos iterations executed. The convergence of EA15D is very slow for many problems. In fact, for some problems we gave up restarting EA15D with more storage before the isolation of all eigenvalues was complete. For these problems EA15D's resource requirements are preceded by the ">" symbol, indi-

	HYBSBC + TQLRAT		EA15D			
Name n,b^{GPS},val/noval	Time (CPU sec)	Storage (KBytes)	Time (CPU sec)	Storage (KBytes)	Eigenvalues Found	Lanczos Iterations
ERIS1176 1176,100,noval	65.9	969.02	589.6	397.72	1080	7459
BCSPWR06 1454,100,noval	130.9	1198.1	444.1	333.34	1446	5937
BCSSTK05 153,23,val	0.74	31.824	3.8	39.376	152	381
BCSSTK09 1083,95,val	159.9	849.07	417.1	401.94	711	8591
CAN.1054 1054,112,noval	121.3	969.68	251.9	258.8	1054	3560
DWT.1005 1005,106,noval	88.8	876.36	193.1	222.7	1004	3018
NOS3 960,65,val	87.6	522.24	247.6	299.92	953	4279
BCSSTK19 817,18,val	14.7	137.26	71.9	192.92	155	3253
1138.BUS 1138,126,val	84	1174.4	1505.5	442.75	452	16047
BCSSTK26 1922,245,val	705.0	3813.3	4532.6	1054.8	492	37487
PLAT1919 1919,80,val	361.9	1274.2	146.3	461.71	684	2070

Table 6.2: The Computational Cost of Complete Spectrum Determination with HYBSBC+TQLRAT and EA15D ($ACC = 10^{-5}$)

cating that additional Lanczos iterations are required to successfully solve the eigenvalue problem.

EA15D successfully isolates all the distinct eigenvalues of 5 problems, but in doing so requires dramatically higher CPU second totals than HYBSBC+TQLRAT. For these problems the CPU requirements of EA15D range from 9.4 to 66 times those of HYBSBC+TQLRAT, while the storage requirements of the two approaches are comparable. For the remaining 6 problems EA15D requires between 34 and 212 times the CPU seconds of HYBSBC+TQLRAT and substantially higher levels of storage, without successfully completing the eigenvalue tasks. Even with these extraordinary timings, EA15D isolates very

few eigenvalues for some of these problems.

Table 6.2 summarizes similar experiments with ACC set to 10^{-5}. In this case, EA15D successfully isolates all the distinct eigenvalues of each problem to an accuracy of 10^{-5} relative to their largest eigenvalue. Despite dramatically reducing the requested accuracy, for 10 of the 11 problems EA15D still uses substantially more CPU seconds than HYBSBC+TQLRAT, which finds all eigenvalues to full accuracy and identifies their multiplicities as well. For PLAT1919 EA15D's running time actually drops below that of HYBSBC+TQLRAT. The efficiency of EA15D in this case is a consequence of the low accuracy requested. Except for a single eigenvalue at zero, all of PLAT1919's eigenvalues have a multiplicity of 2 and the magnitude of the smallest and largest eigenvalues differ by a factor of 10^{13}. Consequently, to satisfy the eigenvalue request EA15D needs to isolate only a small number of eigenvalues relative to the matrix's order. Finally, the storage requirements of EA15D and HYBSBC+TQLRAT for the table's 11 problems are comparable. For some problems EA15D requires significantly less storage, while for others its requirements exceed those of HYBSBC+TQLRAT.

We are also interested in the potential use of HYBSBC based methods for the isolation of moderately large subsets of eigenvalues. To investigate the efficiency of this approach relative to EA15D, we requested EA15D to find all the eigenvalues of BCSSTK05 and ERIS1176 in smaller and smaller intervals including the right end point of their spectra. The fraction of each problem's spectrum EA15D isolates ranges from 0.25 to 1.0. The graphs in Figure 6.1 plot EA15D timings, normalized by the time for complete spectrum determination with HYBSBC+TQLRAT, against the fraction of the spectrum isolated by EA15D. HYBSBC+TQLRAT costs could be reduced if TQLRAT sought the subset of eigenvalues found by EA15D, but HYBSBC dominates the computational requirements and the timings would only change marginally. Once again, for both problems we conduct EA15D experiments with full accuracy and $ACC = 10^{-5}$. As shown by these plots, EA15D has dramatically higher CPU requirements than HYBSBC+TQLRAT for the identification of all but the smallest subsets of eigenvalues. Even when the requested accuracy is significantly lowered, EA15D still requires much higher CPU requirements for the isolation

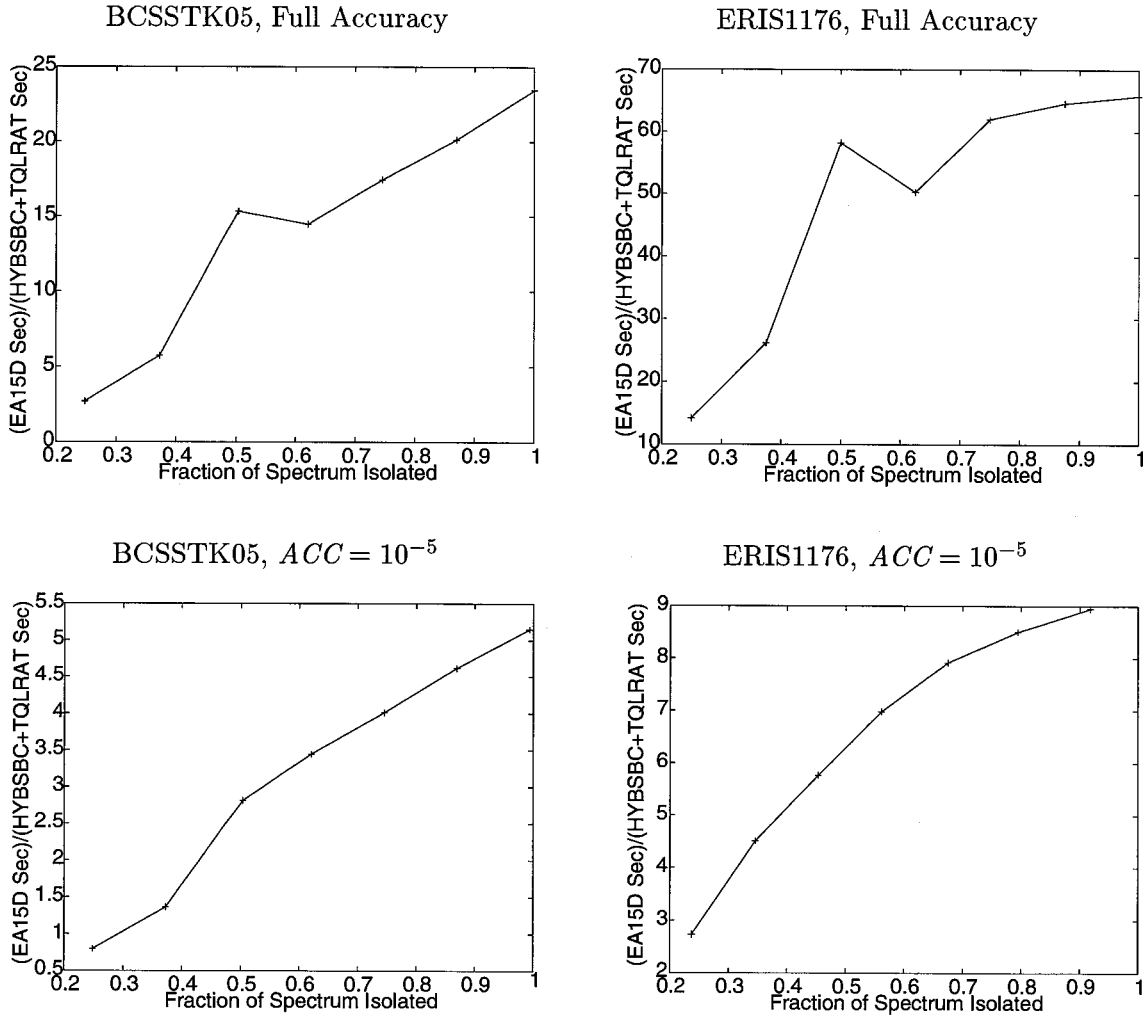


Figure 6.1: CPU Requirements of EA15D Relative to HYBSBC+TQLRAT

of all subsets of ERIS1176's eigenvalues. For BCSSTK05, EA15D timings only manage to drop below those of HYBSBC+TQLRAT during the isolation of the smallest subset of eigenvalues with $ACC = 10^{-5}$.

These experiments clearly demonstrate the efficiency of our novel hybrid tridiagonalization techniques. Although it may be possible to improve the efficiency of EA15D, it is unlikely to significantly change the dramatic advantage enjoyed by HYBSBC+TQLRAT.

For many sparse problems, EA15D's difficulties appear to arise from the isolation of eigenvalues in regions of the spectrum that are poorly separated. The shift and invert Lanczos code of the following subsection is especially suited to isolating eigenvalues from such poorly separated regions of the spectrum. A distinct advantage of our efficient hybrid tridiagonalization approach, however, is that it can isolate blocks of eigenvalues from well and poorly separated regions equally well.

6.6.2 BCS's Block Shift and Invert Lanczos

We conducted all experimentation with the block shift and invert Lanczos algorithm using a test program, HLANCZOS_TEST, provided by John Lewis of Boeing Computer Services. This program uses BCSLIB-EXT routines to input the sparse matrix's structure and values, preorder it with the minimum degree algorithm [TW67, GL89] and perform a symbolic factorization, before calling the double precision eigenvalue extraction routine HDSEEX to compute the requested eigenvalues.

As for Harwell's EA15D, the user must predict the maximum level of storage required by HDSEEX, and its accompanying BCSLIB-EXT routines, before the eigenvalue extraction begins. When isolating the eigenvalues of an ordinary eigenvalue problem, $Ax = \lambda x$, with $NZLA$ nonzeros in the strictly lower triangular portion of A, BCSLIB-EXT documentation suggests allocating the main working storage as a double precision array of length $(200 + (3NZLA + 20n)/2)$. If storage levels are underestimated, the code halts with an error message and must be restarted with a longer array. To reduce the in-memory storage requirements of intermediate results such as eigenvectors, the BCSLIB-EXT code also uses

input/output files for additional storage. The BCSLIB-EXT code may use many MBytes of file space for large eigenvalue problems.

HDSEEX provides a variety of mechanisms for the user to specify which eigenvalues are desired. For each experiment we chose to provide an interval within which the routine is requested to extract all eigenvalues. The user must also provide HDSEEX with the maximum block size for its Lanczos iterations. The block size used for our experments will be specified shortly. Finally, the user must specify an accuracy tolerance for HDSEEX's eigenvalue computations. This value is provided to HDSEEX by HLANCZOS_TEST and evaluates to 2.31×10^{-11} in our testing environment.

All reported timings of the BCSLIB-EXT Lanczos code include the requirements of symbolic factorization and HDSEEX's eigenvalue extraction. To be consistent with the HYBSBC experiments, however, they do not include the time to conduct the preordering, to input the sparse problem, and to check the computed eigenvalues. The CPU second timing of each extraction is actually the average timing for 2 identical extractions. HLANCZOS_TEST performs the timings using the BCSLIB routines HDSLT1 and HDSLT2. Based on comparisons with shell level timing routines, it appears these BCSLIB routines only report user time, and do not account for the system time or any idle time. Consequently, the high levels of file I/O HDSEEX often requires may cause the reported timings to differ significantly from elapsed time. The system time and I/O latency are highly dependent upon the computing environment. In our testing environment, the system time for some extractions comprises more than 10% additional CPU seconds and the elapsed time may differ from user time by more than a factor of 1.4. In contrast, HYBSBC+TQLRAT does not require high levels of file I/O, and timings include both the user and system time.

The focus of this thesis is the computation of eigenvalues, but HDSEEX always computes the eigenvector(s) corresponding to each requested eigenvalue. The cost of constructing eigenvectors cannot be removed completely from our timings, because the code's *external selective reorthogonalization* techniques reorthogonalize Lanczos vectors against subsets of the converged eigenvectors. By subtracting the time for computing eigenvectors, however,

we get a range into which the performance of a block shift and invert Lanczos code tuned for eigenvalue extraction might fall.

To investigate the relative efficiency of the block shift and invert Lanczos code we experimented with three Harwell–Boeing problems: PLAT362, BCSSTK19 and NOS3. The Harwell–Boeing collection provides both the sparsity pattern and values of each problem. The eigenvalues of these problems have a maximum multiplicity of 2 and we chose to use a block size of 3 for all HDSEEX extractions. Because the eigenvalues of each problem are all positive, the eigenvalues at the left end of each spectrum are generally more poorly separated, relative to the difference between the largest and smallest eigenvalues, than those at the right end. We performed two sets of experiments to explore the ability of the BCSLIB-EXT code's shift and invert techniques to isolate eigenvalues from both regions of the spectrum. These experiments request HDSEEX to extract contiguous subsets of eigenvalues from either the left or the right extreme of each problem's spectrum. The fraction of a problem's eigenvalues isolated by each extraction ranges from 0.125 to 1.0. The graphs in Figure 6.2 plot Lanczos timings, normalized by the time for complete spectrum determination with HYBSBC+TQLRAT, against the fraction of the eigenvalues isolated. The solid and broken lines in each plot connect data points whose Lanczos timings include or exclude the computation of eigenvectors respectively. As previously discussed, the performance curve of a shift and invert Lanczos code tuned for eigenvalue extraction would lie between this pair of lines. Table 6.2 of Section 6.6.1 summarizes the cost of determining the complete spectrum of BCSSTK19 and NOS3 with HYBSBC+TQLRAT, while Table 6.3 provides similar information for PLAT362.

n	b^{GPS}	Time (CPU sec.)
362	54	8.96

Table 6.3: HYBSBC+TQLRAT Summary for PLAT362

As shown by the three pairs of plots in Figure 6.2, the Lanczos code exhibits very similar performance characteristics while extracting eigenvalues from opposite ends of a

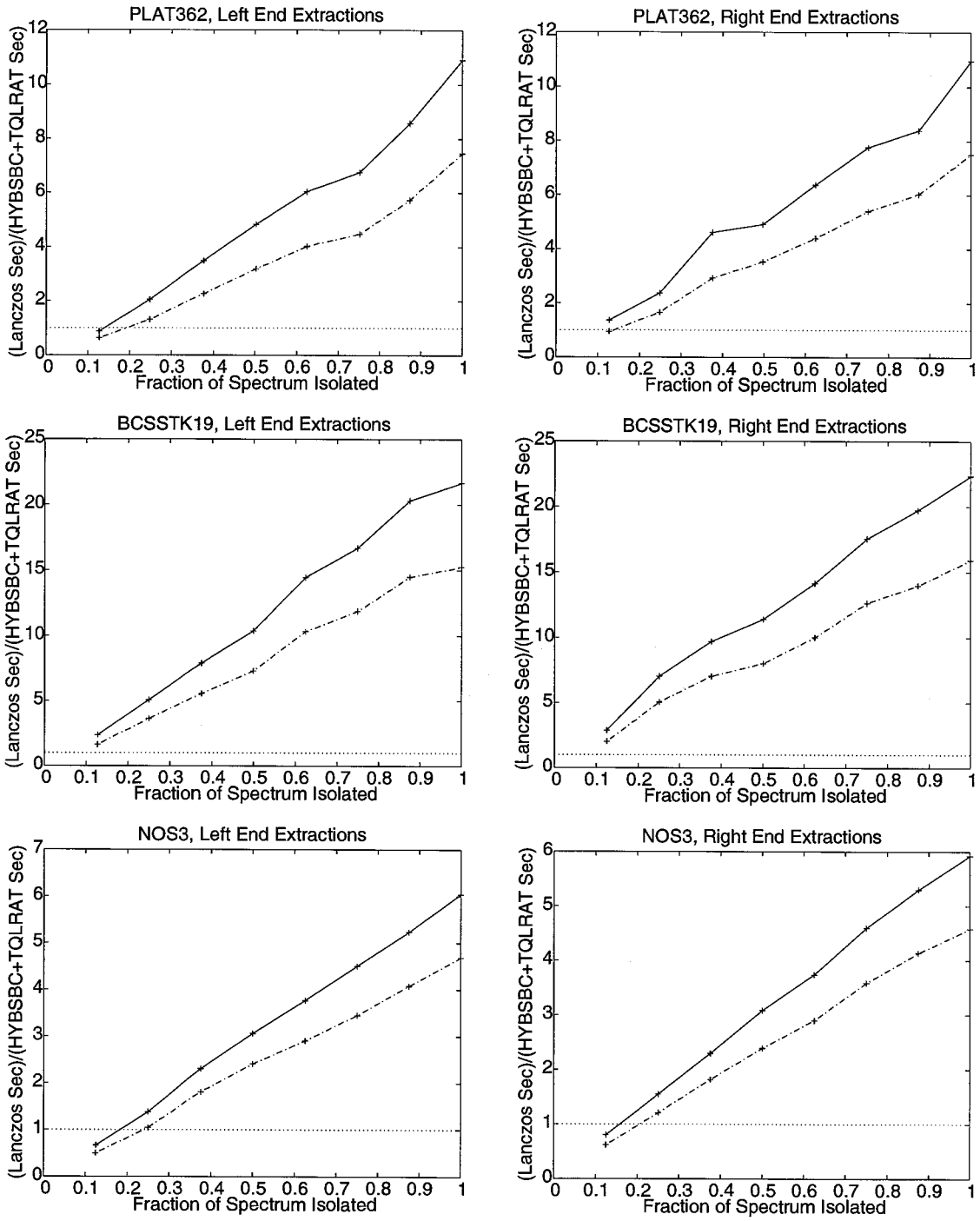


Figure 6.2: The CPU requirements of BCSLIB-EXT Lanczos, relative to HYB-SBC+TQLRAT, extracting subsets of eigenvalues from both the left and right ends of a problem's spectrum.

problem's spectrum. The shift and invert Lanczos code, however, has substantially higher CPU requirements than HYBSBC+TQLRAT for the identification of all but the smallest subsets of eigenvalues. Because the Lanczos timings do not account for the potentially substantial cost of performing file I/O, it is difficult to pinpoint the size of the eigenvalue subset for which computing the complete spectrum with HYBSBC+TQLRAT becomes slower than with the Lanczos code. For PLAT362 the crossover point is close to 10 or 12%, while for BCSSTK19 it appears to be significantly lower. For NOS3 the Lanczos code is relatively more efficient, moving the crossover point slightly higher. Finally, we note that the magnitude of the relative advantage enjoyed by HYBSBC+TQLRAT in these experiments is comparable to the EA15D experiments with $ACC = 10^{-5}$.

This chapter's experiments with BCSLIB-EXT Lanczos and EA15D clearly demonstrate the efficiency and versatility of eigenvalue algorithms based on our novel hybrid tridiagonalization techniques relative to Lanczos-type algorithms. HYBSBC+TQLRAT significantly out performs either Lanczos code when all eigenvalues are requested, and maintains its advantage until we reduce the requested number of eigenvalues to surprisingly relatively small subsets of the entire spectrum.

Chapter 7

Conclusions and Future Research

This dissertation has studied a restricted form of the fundamental algebraic eigenvalue problem, focusing upon generally applicable methods for determining a significant fraction of the eigenvalues of a large sparse symmetric matrix or its complete spectrum. The approach taken by this research was to reexamine techniques for reducing symmetric matrices to tridiagonal form with carefully selected sequences of orthogonal similarity transformations. Using a combination of theoretical analysis and extensive testing with symbolic and numerical implementations, we have developed novel sequential eigenvalue methods that more fully exploit the sparsity of symmetric matrices than previously published methods.

This chapter concludes the thesis by providing a summary of its major results and conclusions, and briefly outlining directions of future research.

7.1 Summary and Conclusions

Without sparse model problems that are both conducive to formal analysis and representative of the characteristics of a broad spectrum of practical sparse problems, we chose to support the development and evaluation of sparse reduction algorithms using a combination of theoretical and experimental analysis. This successful approach was greatly assisted by the group of novel sparse reduction tools described in Chapter 2. A key role in the research this thesis was played by our general framework for analyzing the complexity of algorithms applying sequences of Givens similarity transformations to sparse symmetric matrices. This

framework was used by our research to facilitate the precise analysis of the BC, SBC and R-S algorithms for special model problems. The new symbolic reduction tools `Xmatrix` and `Trisymb` also use this analysis framework to predict the computational requirements of their simulated reductions. `Xmatrix` provides an interactive graphical X Windows interface for modeling the reduction of small sparse matrices with sequences of Givens transformations. In contrast, `Trisymb` provides a platform for the symbolic implementation of sparse reduction algorithms using sequences of Givens transformations to reduce large sparsely or densely banded symmetric matrices. While conducting the research of this thesis both symbolic reduction tools were indispensable, but without the encumbering influence of the computing environment `Trisymb` was especially important for the accurate analysis of algorithms and the confirmation of conclusions drawn from experiments with numerical codes. Finally, we developed a bipartite graph model for the application of orthogonal transformations to sparse symmetric matrices, which is useful for the description and analysis of sparse reduction algorithms.

Using these tools we explored the deficiences and limitations of previously published tridiagonalization methods and considered their potential extension to improve sparsity exploitation. By characterizing the high levels of fill associated with standard and customized Givens reduction algorithms using formal and experimental analysis, we demonstrated that it is essential to restrict the accumulation of fill entries to some maintainable substructure of the matrix to effectively utilize matrix sparsity. Alternatively, we extended the band-preserving tridiagonalization algorithm of Rutishauser and Schwarz for application to general sparse symmetric matrices by introducing an initial bandwidth reducing preordering stage and techniques to exploit band sparsity during the application of transformations. Although the sparse R-S algorithm is a significant improvement, we demonstrated that it suffers from instances of fill cascading that quickly fill the band of a permuted matrix, placing almost complete reliance on the preordering to exploit matrix sparsity. The densely banded algorithm of Lang can be similarly extended for use with general sparse matrices, but its Householder transformations exhibit more prolific fill characteristics.

In response to the limitations of these algorithms we proposed the Bandwidth Con-

traction algorithm. It also employs bandwidth reducing preorderings, band-preserving reduction techniques, and adjacent Givens transformations, but reorders the elimination sequence to prolong and more fully exploit internal band sparsity. Several contributing factors permit the diagonally-oriented reduction of BC to perform a partial reduction that significantly contracts the bandwidth of a permuted sparse matrix at low cost before producing a densely banded intermediate matrix. This allows BC to reduce the tridiagonalization costs of many practical sparse problems relative to R-S. Using detailed complexity analyses of the Rutishauser-Schwarz and Bandwidth Contraction algorithms, however, we showed that the tridiagonalization costs of BC are typically 10–25% more than those of R-S for a densely banded matrix. These observations led to the development of the Hybrid Bandwidth Contraction algorithm. Using a transition strategy thresholding the density of the outermost diagonal of the intermediate band, HYBBC combines a partial bandwidth contraction and R-S to exploit their specific reduction characteristics and improve reduction efficiency.

Extensive experiments were conducted with implementations of the Bandwidth Contraction and Hybrid Bandwidth Contraction algorithms, and a large test suite of practical sparse problems. Both implementations were shown to dramatically reduce the computational requirements of tridiagonalization relative to the EISPACK BANDR routine, an R-S implementation. In fact, for a wide range of 70 practical sparse problems HYBBC reduces CPU requirements by an average of 31%, with reduction as high as 63% for certain problems, but unlike BC it is always comparable to BANDR in the worst case. The improved efficiency of these algorithms was accomplished without increasing storage requirements.

With additional experimentation we also investigated the relationship between preorderings, sparsity structures and HYBBC performance. The class of sparsity structures ideally suited to bandwidth contraction techniques concentrates nonzeros near the main diagonal and exhibits increased sparsity towards the outermost diagonals of the band. These experiments clearly demonstrated, however, that the primary objective of a preordering algorithm must be to reduce bandwidth.

Typically, bulge chasing requirements dominate the costs of applying the Bandwidth Contraction algorithm to most sparse problems. During a sparse tridiagonalization with BC, more than 96% of the transformations may eliminate bulge entries. In response to this observation we developed the diagonally-oriented Split Bandwidth Contraction algorithm, which inherits all the advantages enjoyed by its predecessor for a sparse band. The bidirectional elimination techniques of SBC, however, take additional advantage of band sparsity to dramatically shorten bulge chasing transformation sequences and potentially halve the computational requirements of partial bandwidth contractions.

The success of SBC is dependent upon the position of the split-point at which a sparse diagonal's bidirectional elimination commences. We demonstrated that for many sparse problems the split-point selected for a diagonal's elimination may significantly influence the efficiency of the remainder of the reduction. Using the symbolic reduction tools `Xmatrix` and `Trisymb`, we investigated four global split-point selection strategies. Experimental evidence demonstrated that the minimum displacement split-point selection strategy with damped tiebreaking is preferable.

To evaluate the efficiency of performing partial bandwidth contractions with SBC, we conducted an extensive experimental comparison of BC and SBC using both symbolic and numeric implementations. SBC was found to substantially reduce the computational requirements of partial bandwidth contractions of most problems. Reductions of 45–55% were common, but ranged as high as 76% for one sparse problem. SBC's efficient partial contractions make it an ideal replacement for HYBBC's Bandwidth Contraction stage in a second generation hybrid tridiagonalization algorithm, and a valuable preprocessing technique for other banded eigenvalue routines and sparse linear systems solvers.

Chapter 5's novel Hybrid Split Bandwidth Contraction algorithm incorporates many aspects of HYBBC, but replaces its BC stage with SBC and improves upon the techniques for transition bandwidth selection. Using formal analyses of SBC and BC, we showed that as long as SBC finds a well centred split-point for each diagonal's reduction, it provides tridiagonalizations with significantly lower flop requirements than R-S. If split-points are

forced sufficiently far from the midpoint of a diagonal, however, the costs of completing a tridiagonalization with SBC may rise above those of R-S. To select the transition bandwidth at which HYBSBC switches between its SBC and R-S stages, we developed the Δ-transition strategy. By exploiting characteristics of a typical SBC reduction, the Δ-transition strategy is able to use complexity analyses of SBC and R-S to precisely regulate HYBSBC's transition, selecting close to optimal transition bandwidths that minimize computational requirements for practical problems.

Once again, extensive experimentation was conducted with a numeric implementation of the Hybrid Split Bandwidth Contraction algorithm. The Δ-transition strategy permits HYBSBC to effectively exploit SBC's bidirectional elimination techniques and significantly improve upon both HYBBC and BANDR tridiagonalizations. Relative to HYBBC, HYBSBC substantially reduces transformation totals and provides additional savings in CPU time of 20–40% for many sparse problems. Savings of more than 50% were observed for select problems. These additional gains make HYBSBC a very impressive alternative to BANDR, with one problem tridiagonalized in 1/5 of BANDR's time. For the 70 test problems with $n > 400$, HYBSBC requires on average 39% fewer CPU seconds than BANDR.

To demonstrate the relative efficiency of sparse tridiagonalization based eigenvalue methods, we compared variants of the Lanczos algorithm with the Hybrid Split Bandwidth Contraction algorithm. We used both theoretical and experimental analysis to explore the ability of Lanczos algorithms to economically extract moderately large subsets of eigenvalues or identify the complete spectrum of sparse symmetric problems. Experiments with the BCSLIB-EXT Lanczos code and Harwell's EA15D clearly demonstrate the efficiency of HYBSBC based eigenvalue methods relative to Lanczos-type algorithms. In conjunction with TQLRAT, HYBSBC significantly out performs either Lanczos code when all eigenvalues are requested, and maintains its advantage for the typical sparse problem until the requested number of eigenvalues is reduced to a surprisingly small subset consisting of 10–20% of the entire spectrum's eigenvalues.

From these results we conclude that eigenvalue methods based on the novel hybrid tridi-

agonalization algorithms introduced in this dissertation provide the most reliable, versatile and efficient techniques for isolating subsets of a sparse symmetric matrix's eigenvalues ranging in size from moderate fractions of the entire spectrum to all eigenvalues, including their multiplicities. Although HYBSBC is slower than HYBBC for a small number of our test matrices, HYBSBC significantly outperforms HYBBC for the majority of problems and is recommended as the best algorithm for general implementation.

7.2 Future Research

This section briefly outlines some areas of possible future research resulting from the work of this thesis. Each topic belongs to one or more general categories of research including symbolic reduction tools, the enhancement of sparse bandwidth contraction techniques, the generalization of sparse tridiagonalization techniques, and parallel sparse tridiagonalization algorithms.

- Continue the development of `Xmatrix` to create an interactive teaching tool, permitting the symbolic modeling, manipulation and demonstration of a wide variety of sparse matrix techniques and algorithms.

- Extend our sparse reduction methods with new techniques facilitating eigenvector computation. If one wants to compute eigenvectors by accumulating the transformations applied during a sparse tridiagonalization, it is important to carefully regulate transformation totals to minimize flop requirements. The development of new transition strategies founded in complexity analysis are needed to optimize HYBSBC's reduction in this situation. Alternatively, when only a few eigenvectors are desired, we propose investigating the trade-offs associated with applying a banded inverse iteration algorithm to intermediate matrices from the SBC stage of HYBSBC tridiagonalizations.

- Continue to investigate the relationship between preorderings and the performance of sparse bandwidth contraction techniques. One approach of particular interest is the

possibility of using a local reordering scheme to refocus spectral preordering methods [BPS93] for bandwidth reduction without significantly altering the characteristics of spectral preorderings beneficial to BC and SBC reductions.

- For the special case in which a partial bandwidth contraction is the end goal of a reduction, develop a sparsely banded data structure and reevaluate the implementation of SBC with sparse band transformations.

- Explore the potential role of nonadjacent transformations in special variants of our sparse bandwidth contraction algorithms designed for problems with specific sparsity pattern characteristics.

- Investigate the trade-offs associated with the use of rank-1 tearing and modification techniques [Gol73, BNS78] to create split-points and prolong the efficient Split Bandwidth Contraction stage of HYBSBC's reduction.

- Extend our successful sparse tridiagonalization techniques to develop sparse bidiagonalization algorithms for both symmetric and unsymmetric sparse matrices.

- Explore the potential of creating a sparse, symmetric generalized eigenvalue method by extending Crawford's banded algorithm [Cra73, Kau84] with our sparse reduction techniques.

- Use the knowledge of sparse tridiagonalization gained during the investigations of this thesis to guide the development of novel parallel sparse eigenvalue algorithms. A primary focus of this research would be parallel hybrid tridiagonalization algorithms combining the bidirectional elimination of SBC with densely banded Lang-type algorithms [Lan92, BS92], which employ multiple elimination and delayed bulge chasing techniques to provide efficient parallel implementations. Although SBC's sparse bidirectional elimination enhances the parallelism of sparse bandwidth contractions, one could also consider the introduction of other techniques to improve the parallel implementation of SBC including split-point creation with rank-1 tearings and modifications, delayed bulge chasing, and modified split-point selection methods permitting

the simultaneous elimination of multiple sparse diagonals in a pipelined fashion.

Bibliography

[ABB+92] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, 1992.

[AP94] Andrew A. Anda and Haesun Park. Fast plane rotations with dynamic scaling. *SIAM Journal on Matrix Analysis and Applications*, 15(1):162–174, 1994.

[BK77] James R. Bunch and Linda Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, 31(137):163–179, 1977.

[BNS78] James R. Bunch, Christopher P. Nielsen, and Danny C. Sorensen. Rank-one modification of the symmetric eigenproblem. *Numerische Mathematik*, 31:31–48, 1978.

[Boe89] Boeing Computer Services, Seattle, WA. *The Boeing Extended Mathematical Subprogram Library*, 1989.

[BPS93] Stephen T. Barnard, Alex Pothen, and Horst D. Simon. A spectral algorithm for envelope reduction of sparse matrices. Technical Report CS-93-49, Department of Computer Science, University of Waterloo, Waterloo, Ontario, 1993.

[BS92] Christian H. Bischof and Xiaobai Sun. A framework for symmetric band reduction and tridiagonalization. Preprint MCS-P298-0392, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, July 1992.

[Cav93] Ian A. Cavers. Tridiagonalization costs of the Bandwidth Contraction and Rutishauser-Schwarz algorithms. Technical Report 93–39, Department of Computer Science, University of British Columbia, Vancouver, B.C., November 1993.

[Cav94] Ian A. Cavers. A hybrid tridiagonalization algorithm for symmetric sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 15(4), October 1994. To appear.

[CCDG82] P. Z. Chinn, J. Chvatalova, A. K. Dewdney, and N. E. Gibbs. The bandwidth problem for graphs and matrices – a survey. *Journal of Graph Theory*, 6:223–254, 1982.

[Cra73] C. R. Crawford. Reduction of a band-symmetric generalized eigenvalue problem. *Communications of the ACM*, 16(1):41–44, 1973.

[Cup81] J. J. M. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numerische Mathematik*, 36:177–195, 1981.

[Cut72] Elizabeth Cuthill. Several strategies for reducing the bandwidth of matrices. In Donald J. Rose and Ralph A. Willoughby, editors, *Sparse Matrices and Their Applications*, pages 157–166. Plenum Press, New York, 1972.

[CW79] Jane Cullum and Ralph A. Willoughby. Lanczos and the computation in specified intervals of the spectrum of large, sparse real symmetric matrices. In Iain S. Duff and G. W. Stewart, editors, *Sparse Matrix Proceedings 1978*, pages 220–255. Society for Industrial and Applied Mathematics, 1979.

[CW80] Jane Cullum and Ralph A. Willoughby. Computing eigenvectors (and eigenvalues) of large, symmetric matrices using Lanczos tridiagonalization. In G. A. Watson, editor, *Numerical Analysis, Proceedings of the 8th Biennial Conference, Dundee, June 26-29, 1979*, pages 46–63. Springer-Verlag, 1980.

[DC92] J. Du Croz, Personal communication, October 1992.

[DDHD90] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.

[DDHH88] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–17, 1988.

[DGLP82] I. S. Duff, Roger Grimes, John Lewis, and Bill Poole. Sparse matrix test problems. *ACM SIGNUM Newsletter*, 17:22, June 1982.

[DR75] I. S. Duff and J. K. Reid. On the reduction of sparse matrices to condensed forms by similarity transformations. *Journal of the Institute of Mathematics and Its Applications*, 15:217–224, 1975.

[DS87] Jack J. Dongarra and D. C. Sorensen. A fully parallel algorithm for the symmetric eigenvalue problem. *SIAM Journal on Scientific and Statistical Computing*, 8(2):s139–s154, 1987.

[ELT79] J. T. Edwards, D. C. Licciardello, and D. J. Thouless. Use of the lanczos method for finding complete sets of eigenvalues of large sparse symmetric matrices. *Journal of the Institute of Mathematics and Its Applications*, 23:277–283, 1979.

[ER80] Thomas Ericsson and Axel Ruhe. The spectral transformation Lanczos method for the numerical solution of large sparse generalized symmetric eigenvalue problems. *Mathematics of Computation*, 35(152):1251–1268, October 1980.

[FGMS93] S. I. Feldman, David M. Gay, Mark W. Maimone, and N. L. Schryer. A FORTRAN–to–C converter. Computing Science Technical Report 149, AT&T Bell Laboratories, 1993.

[GBDM77] B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler. *Matrix Eigensystem Routines - EISPACK Guide Extension*, volume 51 of *Lecture Notes in Computer Science*. Springer-Verlag, New York, 1977.

[Gen73] W. Morven Gentleman. Least squares computations by Givens transformations without square roots. *Journal of the Institute of Mathematics and Its Applications*, 12:329–336, 1973.

[GGJK78] M. R. Garey, R. L Graham, D. S. Johnson, and D. E. Knuth. Complexity results for bandwidth minimization. *SIAM Journal on Applied Mathematics*, 34(3):477–495, 1978.

[Gib76] N. E. Gibbs. A hybrid profile reduction algorithm. *ACM Transactions on Mathematical Software*, 2(4):378–387, 1976.

[GL78a] Alan George and Joseph W. H. Liu. An automatic nested dissection algorithm for irregular finite element problems. *SIAM Journal on Numerical Analysis*, 15(5):1053–1069, October 1978.

[GL78b] J. A. George and J. W. H. Liu. User guide for SPARSPAK: Waterloo sparse linear equations package. Research Report CS-78-30, Department of Computer Science, University of Waterloo, Waterloo, Ontario, 1978.

[GL89] Alan George and Joseph W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989.

[GLD92] Roger G. Grimes, John G. Lewis, and Iain S. Duff. Users' guide for the Harwell-Boeing sparse matrix collection (release I). Technical Report MEA-TR-202 R1, Mathematics and Engineering Analysis, Boeing Computer Services, Seattle, WA, October 1992.

[GLS94] Roger G. Grimes, John G. Lewis, and Horst D. Simon. A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems. *Siam Journal on Matrix Analysis and Applications*, 1(15):228–272, January 1994.

[GMS92] John R. Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in MATLAB: Design and implementation. *SIAM Journal on Matrix Analysis and Applications*, 13(1):333–356, January 1992.

[Gol73] Gene H. Golub. Some modified matrix eigenvalue problems. *SIAM Review*, 15(2):318–334, 1973.

[GPS76a] Norman E. Gibbs, William G. Poole Jr., and Paul K. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis*, 13(2):236–250, 1976.

[GPS76b] Norman E. Gibbs, William G. Poole Jr., and Paul K. Stockmeyer. A comparison of several bandwidth and profile reduction algorithms. *ACM Transactions on Mathematical Software*, 2(4):322–330, 1976.

[GV89] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, second edition, 1989.

[Har69] Frank Harary. *Graph Theory.* Addison-Wesley Publishing Company, Inc., 1969.

[Har90] Harwell Laboratory, Oxfordshire, England. *Harwell Subroutine Library, Release 10*, 1990.

[Jes89] Elizabeth R. Jessup. *Parallel Solution of the Symmetric Tridiagonal Eigenproblem.* PhD thesis, Department of Computer Science, Yale University, October 1989.

[Kau84] Linda Kaufman. Banded eigenvalue solvers on vector machines. *ACM Transactions on Mathematical Software*, 10(1):73–86, 1984.

[Lan50] Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 45(4):255–282, October 1950.

[Lan92] Bruno Lang. Reducing symmetric banded matrices to tridiagonal form – A comparison of a new parallel algorithm with two serial algorithms on the iPSC/860. Technical report, Universität Karlsruhe, January 1992.

[Lew82] John G. Lewis. Implementation of the Gibbs–Poole–Stockmeyer and Gibbs–King algorithms. *Transactions on Mathematical Software*, 8(2):180–189, 1982.

[LHKK79] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for FORTRAN usage. *ACM Transactions on Mathematical Software*, 5(3):308–323, 1979.

[Mat92] The MathWorks, Natick, MA. *MATLAB User's Guide*, 1992.

[MRW71] R. S. Martin, C. Reinsch, and J. H. Wilkinson. The QR algorithm for band symmetric matrices. In J. H. Wilkinson and C. Reinsch, editors, *Linear Algebra*, volume II of *Handbook for Automatic Computation*, pages 266–272. Springer-Verlag, 1971.

[Pap76] C. H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*, 16:263–27, 1976.

[Par80] Beresford N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice–Hall, 1980.

[Pot93] Alex Pothen, Personal communication, May 1993.

[PR81] B. N. Parlett and J. K. Reid. Tracking the progress of the Lanczos algorithm for large symmetric eigenproblems. *IMA Journal of Numerical Analysis*, 1:135–155, 1981.

[PS79] B. N. Parlett and D. S. Scott. The Lanczos algorithm with selective orthogonalization. *Mathematics of Computation*, 33(145):217–238, January 1979.

[Rut63] H. Rutishauser. On Jacobi rotation patterns. In *Experimental Arithmetic, High Speed Computing and Mathematics*, volume 15 of *Proceedings of Symposia in Applied Mathematics*, pages 219–239, Providence, RI, April 1963. American Mathematical Society.

[SBD+76] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. *Matrix Eigensystem Routines - EISPACK Guide*, volume 6 of

Lecture Notes in Computer Science. Springer-Verlag, New York, second edition, 1976.

[Sch63] H. R. Schwarz. Reduction of a symmetric bandmatrix to triple diagonal form. *Communications of the ACM*, 6(6):315–316, June 1963.

[Sch71] H. R. Schwarz. Tridiagonalization of a symmetric band matrix. In J. H. Wilkinson and C. Reinsch, editors, *Linear Algebra*, volume II of *Handbook for Automatic Computation*, pages 273–283. Springer-Verlag, New York, 1971.

[Sco83] David S. Scott. *LASO2 Documentation.* Computer Science Department, University of Texas at Austin, 1983.

[Sim84] Horst D. Simon. The Lanczos algorithm with partial reorthogonalization. *Mathematics of Computation*, 42(165):115–142, 1984.

[TW67] W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proceedings of the IEEE*, 55(11):1801–1809, 1967.

[Wil65] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Claredon Press, Oxford, second edition, 1965.