Solution of Unassembled Linear Systems Using Block Stretching: Preliminary Experiments.

Michel J. Daydé[1], Jérôme P. Décamps[1], Nicholas I. M. Gould[2]
ENSEEIHT-IRIT Technical Report, **RT/APO/97/3**

December 3, 1997

Abstract

We consider so-called "matrix stretching" technique that make structured unassembled linear systems larger, but sparser. Our solution technique combines a direct factorization of the leading block diagonal submatrix of the stretched system, with a preconditioned conjugate gradient solution of the Schur complement system which results from the factorization of the diagonal blocks.

We show that matrix stretching is an effective technique, particularly for ill-conditioned systems. The Schur complement is often considerably better conditioned than the whole system. The main challenge is to find a suitable preconditioner for this matrix. We consider a range of preconditioners, including those proposed by Chan, and band approximations. We also study the use of some Element-by-Element preconditioners such as EBE and the recently introduced Subspace-by-Subspace preconditioner.

We report on experiments using structured problems and examples from the Harwell-Boeing sparse matrix collection. We also report on some preliminary parallel experiments that show how stretching improves the parallelisation of the linear solver.

1 Introduction

Stretching, a sparse matrix preprocessing technique that makes matrices sparser but, at the same time, larger was first introduced by Grcar (1990). He proposed two general stretching techniques, simple row and column stretching, and studied their efficacy on the factorization of arrow-head matrices. Alvarado (1997) showed that such techniques are an effective way of treating matrices with dense rows or columns before forming their LDU or QR factorizations. Similar ideas have been developed by Vanderbei (1991) and Andersen (1996) for solving the Schur complements arising from interior point methods in the context of linear programming.

In this paper, we apply matrix stretching to the solution of a linear systems of "finite-element" type without assembling the coefficient matrices from its "elements". This requires that we introduce extra variables to recouple the "element" blocks of the matrix. The resulting enlarged system is a sparse "augmented system", which can be solved using a range of direct and iterative methods.

To be precise, we consider the solution of the symmetric system of linear equations of the form

$$\boldsymbol{B}\boldsymbol{x} = \boldsymbol{b}, \tag{1.1}$$

[1]ENSEEIHT-IRIT, 2 rue Camichel, 31071 Toulouse CEDEX, France
[2]Central Computing Department, Rutherford Appleton Laboratory, Oxfordshire, OX11 0QX, England.

where $\boldsymbol{B}$ can be expressed as

$$\boldsymbol{B} = \sum_{i=1}^{n_e} \boldsymbol{B}_i, \tag{1.2}$$

and where each symmetric elementary matrix $\boldsymbol{B}_i$ only involves a small subset of the variables $\boldsymbol{x}$. We let $\boldsymbol{B}^i$ be the matrix obtained from by $\boldsymbol{B}_i$ by removing its zero rows and columns. Note that we do not assume that each $\boldsymbol{B}_i$ is positive definite, merely that it is non-singular.

Linear systems of the form (1.1)–(1.2) arise when solving the Newton equations for the unconstrained minimization of a partially separable function (Griewank and Toint, 1982*a*, 1982*b*), and from finite-element methods for the solution of partial-differential equations (see, for instance, Zienkiewicz, 1977).

The stretched system we obtain is an augmented system of the form

$$\begin{pmatrix} \boldsymbol{B}^{\mathrm{S}} & \boldsymbol{A} \\ \boldsymbol{A}^T & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{\mathrm{S}} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}^{\mathrm{S}} \\ \boldsymbol{0} \end{pmatrix} \tag{1.3}$$

where $\boldsymbol{B}^{\mathrm{S}}$ is block diagonal and $\boldsymbol{A}$ is very sparse. The diagonal blocks of $\boldsymbol{B}^{\mathrm{S}}$ are the $\boldsymbol{B}^i$. The matrix $\boldsymbol{A}$ is used to recouple the $\boldsymbol{B}_i$ when they share variables. As we shall shortly show, the required solution $\boldsymbol{x}$ may easily recovered from $\boldsymbol{x}^{\mathrm{S}}$.

The solution $\boldsymbol{x}^{\mathrm{S}}$ to (1.3) also solves the quadratic programming problem

$$\underset{\boldsymbol{x}^{\mathrm{S}} \in R^{n_b}}{\text{minimize}} \quad q(\boldsymbol{x}^{\mathrm{S}}) = \tfrac{1}{2}\langle \boldsymbol{x}^{\mathrm{S}}, \boldsymbol{B}^{\mathrm{S}}\boldsymbol{x}^{\mathrm{S}}\rangle - \langle \boldsymbol{b}^{\mathrm{S}}, \boldsymbol{x}^{\mathrm{S}}\rangle \ \text{ subject to } \ \boldsymbol{A}^T\boldsymbol{x}^{\mathrm{S}} = \boldsymbol{0}, \tag{1.4}$$

where n_b is the dimension of B^{S} and $\langle \cdot, \cdot \rangle$ denotes the Euclidean inner-product (see for instance Gill, Murray and Wright, 1981). The augmented system (1.3) or the quadratic program (1.4) may be solved in many ways (see for instance Gill et al., 1981), and we choose the well-known Schur complement approach. That is, we perform a block decomposition of the coefficient matrix

$$\begin{pmatrix} \boldsymbol{B}^{\mathrm{S}} & \boldsymbol{A} \\ \boldsymbol{A}^T & \boldsymbol{0} \end{pmatrix} \tag{1.5}$$

by factorizing each of the the diagonal blocks of $\boldsymbol{B}^{\mathrm{S}}$, and use the method of preconditioned conjugate gradients (PCG) to solve the remaining linear system which results following elimination of these leading blocks.

2 Description of the stretching algorithm

2.1 Introduction: a simple example

We consider the solution of $\boldsymbol{B}\boldsymbol{x} = \boldsymbol{b}$ where $\boldsymbol{B}$ is structured as

$$\begin{pmatrix} 8 & 1 & 1 & & \\ 1 & 8 & 1 & & \\ 1 & 1 & 8 & 1 & 1 \\ & & 1 & 8 & 1 \\ & & 1 & 1 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}. \tag{2.6}$$

Additionally, we assume that $\boldsymbol{B}$ is composed of two elements $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$ — $\boldsymbol{B}_1$ involves variables x_1, x_2, x_3, while $\boldsymbol{B}_2$ involves x_3, x_4, x_5. Thus

$$\boldsymbol{B}^1 = \begin{pmatrix} 8 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 4 \end{pmatrix} \quad \text{and} \quad \boldsymbol{B}^2 = \begin{pmatrix} 4 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 8 \end{pmatrix}$$

(Such an example might arise, for instance, if the linear system is the Newton equation arising from the unconstrained minimization of the partially separable cost function $f(x) = f^1(x) + f^2(x)$ with $f^1(x) = f^1(x_1, x_2, x_3)$ and $f^2(x) = f^2(x_3, x_4, x_5)$. In this case, $\boldsymbol{b}$ is the negative of the gradient of f, $\boldsymbol{B}$ is the Hessian of f, and $\boldsymbol{B}^1$ and $\boldsymbol{B}^2$ are respectively the projected Hessians of f^1 and f^2 onto the variables used by f^1 and f^2.) The "overlap" in this simple example only involves the variable x_3.

Since we do not want to assemble B, we expand—or stretch—the system $Bx = b$ as follows. Let $\boldsymbol{B}^{\mathrm{S}}$ be the matrix obtained by expanding $\boldsymbol{B}$ without assembling the overlapping parts, ie,

$$\boldsymbol{B}^{\mathrm{S}} = \begin{pmatrix} 8 & 1 & 1 & & & \\ 1 & 8 & 1 & & & \\ 1 & 1 & 4 & & & \\ & & & 4 & 1 & 1 \\ & & & 1 & 8 & 1 \\ & & & 1 & 1 & 8 \end{pmatrix}. \tag{2.7}$$

Our aim is to replace (2.6) by a system in which (2.7) is a submatrix. We look for a new solution vector $({x_1}^{\mathrm{S}}, \cdots, {x_6}^{\mathrm{S}})^T$, where we identify

$$x_1 = {x_1}^{\mathrm{S}}, \quad x_2 = {x_2}^{\mathrm{S}}, \quad x_4 = {x_5}^{\mathrm{S}} \quad \text{and} \quad x_5 = {x_6}^{\mathrm{S}} \tag{2.8}$$

since these unknowns do not occur in the overlap. We also require that

$$x_3 = x_3^{\mathrm{S}} = x_4^{\mathrm{S}}. \tag{2.9}$$

In addition, if we let

$${x_1}^{\mathrm{S}} + {x_2}^{\mathrm{S}} + 4{x_3}^{\mathrm{S}} = \omega_1 \quad \text{and} \quad 4{x_4}^{\mathrm{S}} + {x_5}^{\mathrm{S}} + {x_6}^{\mathrm{S}} = \lambda_1, \tag{2.10}$$

then the third equation in (2.6) requires that

$$\omega_1 + \lambda_1 = b_3. \tag{2.11}$$

Combining (2.6)–(2.10) and eliminating ω_1 using (2.11) then yields the symmetric augmented system

$$\begin{pmatrix} 8 & 1 & 1 & & & & & \\ 1 & 8 & 1 & & & & & \\ 1 & 1 & 4 & & & & +1 & \\ & & & 4 & 1 & 1 & -1 & \\ & & & 1 & 8 & 1 & & \\ & & & 1 & 1 & 8 & & \\ & & +1 & -1 & & & & \end{pmatrix} \begin{pmatrix} {x_1}^{\mathrm{S}} \\ {x_2}^{\mathrm{S}} \\ {x_3}^{\mathrm{S}} \\ {x_4}^{\mathrm{S}} \\ {x_5}^{\mathrm{S}} \\ {x_6}^{\mathrm{S}} \\ \lambda_1 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ 0 \\ b_4 \\ b_5 \\ 0 \end{pmatrix}. \tag{2.12}$$

2.2 The stretching algorithm

The general algorithm for building the required stretched system is simple. Suppose $\boldsymbol{B}$ is of order n. We then construct a system of the form

$$\begin{pmatrix} {\boldsymbol{B}_1}^{\mathrm{S}} & & & & \boldsymbol{A}_1 \\ & {\boldsymbol{B}_2}^{\mathrm{S}} & & & \boldsymbol{A}_2 \\ & & \ddots & & \vdots \\ & & & \boldsymbol{B}_{n_e}^{\mathrm{S}} & \boldsymbol{A}_{n_e} \\ \boldsymbol{A}_1^T & \boldsymbol{A}_2^T & \cdots & \boldsymbol{A}_{n_e}^T & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_1^{\mathrm{S}} \\ \boldsymbol{x}_2^{\mathrm{S}} \\ \vdots \\ \boldsymbol{x}_{n_e}^{\mathrm{S}} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}_1^{\mathrm{S}} \\ \boldsymbol{b}_2^{\mathrm{S}} \\ \vdots \\ \boldsymbol{b}_{n_e}^{\mathrm{S}} \\ \boldsymbol{0} \end{pmatrix} \tag{2.13}$$

using the algorithm described in Algorithm 2.1.

Algorithm 2.1: The stretching algorithm

Initialization of $\boldsymbol{B}^{\mathrm{S}}$
  For each element $i = 1, \cdots, n_e$
    $\boldsymbol{B}^{\mathrm{S}}_i$ is the matrix $\boldsymbol{B}^i$
Initialization of $\boldsymbol{A}$ and $\boldsymbol{b}^{\mathrm{S}}$
  For each variable $i = 1, \cdots, n$
    Compute $list$, the list of elements possessing i.
    If i is shared by $\deg(i)$ elements,
    create ($\deg(i)$-1) columns for each $\boldsymbol{A}_j$ for $j = 1, n_e$.
    $list(1)$ is the first element possessing i
    Let k be the row of $\boldsymbol{B}^{\mathrm{S}}_{list(1)}$ associated with variable i.
    Set the $(deg(i) - 1)$ columns of row k of $\boldsymbol{A}_{list(1)}$ to 1.
    Set the corresponding value in $\boldsymbol{b}^{\mathrm{S}}$ to b_i.
    For $j = 2, \cdots, deg(i)$
      Let k be the row of $\boldsymbol{B}^{\mathrm{S}}_{list(j)}$ associated with variable i.
      Set column $j - 1$ of row k of $A_{list(j)}$ to -1
      Set the corresponding value in $\boldsymbol{b}^{\mathrm{S}}$ to 0.

Having solved the stretched system, we simply recover $\boldsymbol{x}$ from $\boldsymbol{x}^{\mathrm{S}}$, by setting $\boldsymbol{x}_i$ to its value in one of the blocks involving it. A number of comments are in order.

1. We start the stretching process for variable i with the first element in which it is involved. Other strategies may prove more effective, and further experiments are required.

2. It is easy to compute the number of variables involved in the augmented system. Let n_o be the number of variables shared by several elements, $deg(i)$ the number of elements in which the i^{th} variable appears, and $n_v(i)$ the number of variables involved in element i. Then the augmented system involves $n + \sum_{i=1}^{n_o} 2(deg(i) - 1)$ variables, of which $\sum_{i=1}^{n_o}(deg(i) - 1)$ make up $\boldsymbol{\lambda}$. The matrix $\boldsymbol{A}$ has $n_s \stackrel{def}{=} \sum_{i_e=1}^{n_e} n_v(i_e)$ rows and $\sum_{i=1}^{n_o}(deg(i) - 1)$ columns, while $\boldsymbol{B}^{\mathrm{S}}$ is of order $\sum_{i_e=1}^{n_e} n_v(i_l)$; $\boldsymbol{A}$ has $2\ \sum_{i=1}^{n_o}(deg(i) - 1)$ nonzeros (two nonzeros per column).

3. If we are solving a sequence of problems of the same form (1.2), we only need to construct $\boldsymbol{A}$ for the first such problem. For subsequent problems, we merely update the nonzeros of $\boldsymbol{b}^{\mathrm{S}}$ and $\boldsymbol{B}^{\mathrm{S}}$.

2.3 A further example

In Figure 2.1, we consider a 6×6 matrix composed of 4 elements. Here, element 1 uses variables 1, 2, 3, 4, 5 and 6, element 2 uses 2, 3 and 4, element 3 uses 3, 4, 5 and 6, and element 4 uses variables 4 and 5.

When using the method described, we obtain the augmented system illustrated in Figure 2.2. Here, $\boldsymbol{B}^{\mathrm{S}}$ is a 15×15 matrix composed of the 4 unassembled elements, $\boldsymbol{A}$ is a 15×9 matrix with 18 nonzeros, and $\boldsymbol{\lambda}$ is a vector of 9 elements.

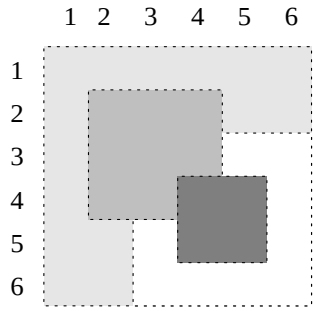

Figure 2.1: The initial matrix and its elements.

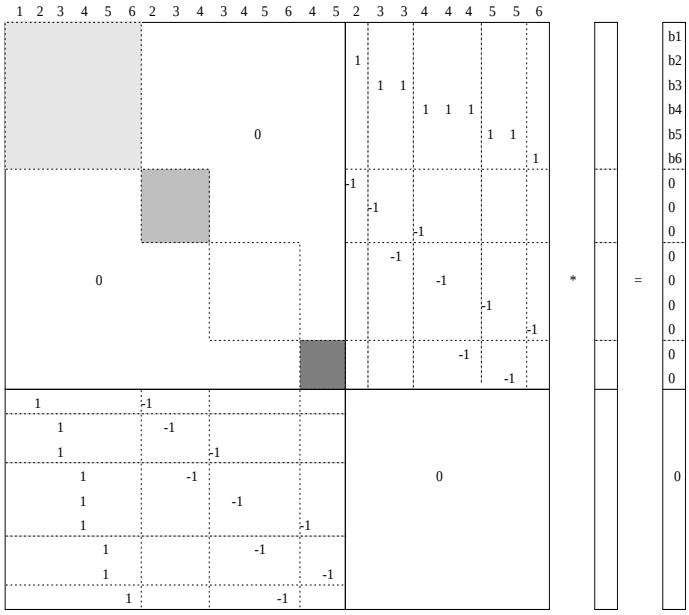

Figure 2.2: The stretched system.

3 The solution of stretched linear systems

The Schur complement method (see Haynsworth, 1968 and Axelsson, 1996) takes advantage of the structure of the system (2.13). The method is formalized as Algorithm 3.1.

Algorithm 3.1: The Schur complement algorithm on the stretched system

1. $\forall i \in 1,..,n_e$, solve $\boldsymbol{B}_i{}^{\mathrm{S}} \boldsymbol{z}_i = \boldsymbol{b}_i{}^{\mathrm{S}}$ (using the LDL^T factorization from LAPACK, see Anderson, Bai, Bischof, Demmel, Dongarra, Croz, Greenbaum, Hammarling, McKenney and Sorensen, 1990)
2. Use the PCG method to solve $\boldsymbol{S}\boldsymbol{\lambda} = \boldsymbol{s}$, where the n_s by n_s Schur complement $\boldsymbol{S} = \sum_{i=1}^{n_e} \boldsymbol{A}_i^T (\boldsymbol{B}_i{}^{\mathrm{S}})^{-1} \boldsymbol{A}_i$ and $\boldsymbol{s} = \sum_{i=1}^{n_e} \boldsymbol{A}_i^T \boldsymbol{z}_i$
3. Solve $\boldsymbol{B}_i{}^{\mathrm{S}} \boldsymbol{x}_i{}^{\mathrm{S}} = \boldsymbol{b}^{\mathrm{S}} + \boldsymbol{A}_i \boldsymbol{\lambda}$ (using LAPACK)
4. $\forall k \in 1,..,n$, recover x_k from its value in $\boldsymbol{x}_i{}^{\mathrm{S}}$, where (e.g.) $\boldsymbol{B}_i{}^{\mathrm{S}}$ is the last element involving x_k.

This approach offers great opportunities for parallelisation since Steps 1 and 3 can be computed element-wise in parallel. Moreover, one hopes, in Step 2, to exploit parallelism within the PCG iteration — conjugate gradients (CG) are chosen since we may not want to form $\boldsymbol{S}$. We use the LDL^T factorization of the elements formed during the step 1 to compute the Schur complement. We then obtain

$$\boldsymbol{S} = \sum_{i=1}^{n_e} \boldsymbol{A}_i^T (\boldsymbol{B}_i{}^{\mathrm{S}})^{-1} \boldsymbol{A}_i = \sum_{i=1}^{n_b} \boldsymbol{s}_i \boldsymbol{s}_i^T, \tag{3.14}$$

where n_b is the dimension of $\boldsymbol{B}^{\mathrm{S}}$ in (1.3). This decomposition proves to be useful during the construction of all but the probing-vector preconditioner we consider in the next section. The remaining ingredient is, of course, the choice of preconditioner.

4 Preconditioning the Schur complement

We have used a number of different preconditioners. These include

- probing-vector preconditioners — the Chan diagonal and band preconditioners,
- traditional band preconditioners, and
- Element-by-Element (EBE) and Subspace-by-Subspace (SBS) preconditioners.

We consider each in some detail.

4.1 Probing-vector preconditioners

The aim here is to estimate values of $\boldsymbol{S}$ merely by forming products of $\boldsymbol{S}$ with appropriate vectors. This avoids the need to explicitly form $\boldsymbol{S}$.

4.1.1 The Chan diagonal preconditioner

The simplest preconditioner is simply the diagonal of $\boldsymbol{S}$. The exact diagonal $\Delta(\boldsymbol{S})$ could be obtained from n_s matrix-vector products with columns of the identity matrix. However, the cost of this strategy is usually prohibitive. As an alternative, Chan and Goovaerts (1988) propose approximating this diagonal by the product of the Schur complement with p ($p < n_s$) so-called *probing* vectors. The i^{th} probing vector, $\boldsymbol{v}_i$, has ones in positions $i + kp$, $k = 0, 1, \cdots$, and zeros elsewhere. The resulting diagonal preconditioner $\boldsymbol{M}$ is obtained as

$$\begin{array}{ccccc} \Delta(\boldsymbol{S})_{i,i} & \simeq & (\boldsymbol{S}\boldsymbol{v}_i)_i & = & \boldsymbol{M}_{i,i} \\ \Delta(\boldsymbol{S})_{i+p,i+p} & \simeq & (\boldsymbol{S}\boldsymbol{v}_i)_{i+p} & = & \boldsymbol{M}_{i+p,i+p} \\ \Delta(\boldsymbol{S})_{i+2p,i+2p} & \simeq & (\boldsymbol{S}\boldsymbol{v}_i)_{i+2p} & = & \boldsymbol{M}_{i+2p,i+2p} \end{array}$$

for $i = 1, \ldots, p$. Any negative diagonal elements so formed are subsequently replaced by $\epsilon_m^{1/3}$ — ϵ_m is the of the machine precision — which appears, empirically, to be a satisfactory value.

4.1.2 The Chan band preconditioner

The idea behind the Chan diagonal preconditioner is easily generalized to finding band approximations (see, for instance, Chan and Mathew, 1992). To this end, suppose $\boldsymbol{S}$ is a symmetric band matrix. We then compute $\boldsymbol{S}$ using values obtained from the products of $\boldsymbol{S}$ with vectors $\{\boldsymbol{v}_i\}_{i \in \mathcal{I}}$. Although computing the minimum number of vectors required is an NP hard problem for general matrices (Coleman and Moré, 1984), a number of satisfactory heuristics have been proposed (see for instance, Curtis, Powell and Reid, 1974, Powell and Toint, 1979, and Coleman and Moré, 1984). In particular, if $\boldsymbol{S}$ has a semi-bandwith of q, only $2q + 1$ probing vectors are required; the example

$$\begin{pmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & & \\ & a_{32} & a_{33} & a_{34} & & \\ & & a_{43} & a_{44} & a_{45} & \\ & & & a_{54} & a_{55} & \ddots \\ & & & & \ddots & \ddots \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ \vdots & \vdots & \vdots \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & a_{23} \\ a_{34} & a_{32} & a_{33} \\ a_{44} & a_{45} & a_{43} \\ a_{54} & a_{55} & a_{56} \\ \vdots & \vdots & \vdots \end{pmatrix}$$

shows how this is achieved for tridiagonal matrices.

We shall denote the matrix computed by this strategy as $P(\boldsymbol{S}, q)$. Since, our matrix $\boldsymbol{S}$ is generally not a band matrix, we only get an approximation of the band of $\boldsymbol{S}$. This approximation may only be useful if $\boldsymbol{S}$ is diagonal dominant, but the approximation generally improves as q is increases. As the matrix we require is also is symmetric, we need to symmetrize $P(\boldsymbol{S}, q)$. Several strategies have been proposed for imposing symmetry of the computed approximation (see Chan and Mathew, 1992 for an exhaustive list). That by Keyes and Gropp (1987, 1989) requires only $q + 1$ probing vectors. However, we prefer to use the alternative
$P_{sym}(\boldsymbol{S}, q)_{ij} =$

$$\begin{cases} P(\boldsymbol{S}, q)_{ij} \text{ if } |P(\boldsymbol{S}, q)_{ij}| = \min\{|P(\boldsymbol{S}, q)_{ij}|, |P(\boldsymbol{S}, q)_{ji}|\}, \\ P(\boldsymbol{S}, q)_{ji} \text{ if } |P(\boldsymbol{S}, q)_{ji}| = \min\{|P(\boldsymbol{S}, q)_{ij}|, |P(\boldsymbol{S}, q)_{ji}|\}. \end{cases}$$

where $P(\boldsymbol{S}, q)$ is still the initial version described above. Although this strategy may require more probing vectors, it has the advantage of preserving the diagonal dominance of $\boldsymbol{S}$. Once, we have calculated this approximation to the band of $\boldsymbol{S}$, we use the LDL^T (modified) band factorization method from the LANCELOT package (see Conn, Gould and Toint, 1992) to form its factors.

4.2 Traditional band preconditioners

We might equally assemble a band submatrix of the Schur complement, and factorize it using the LANCELOT band factorization. Unlike the probing vector preconditioners, the band submatrix computed in this way is the exact band restriction of the Schur complement $\boldsymbol{S}$.

4.3 Element-by-Element preconditioners

We can construct the elements of the Schur complement as the sum $\sum_{i=1}^{n_e} \boldsymbol{S}_i$ where $\boldsymbol{S}_i = \boldsymbol{A}_i^T \boldsymbol{B}_i^{-1} \boldsymbol{A}_i$ and then form an EBE preconditioner based upon the unassembled structure of the Schur complement. Alternatively, we can use the decomposition

$$\boldsymbol{S} = \sum_{i=1}^{n_e} \boldsymbol{S}_i = \sum_{i=1}^{n_b} \boldsymbol{s}_i \boldsymbol{s}_i^T, \tag{4.15}$$

where the $\boldsymbol{s}_i \boldsymbol{s}_i^T$ are rank-one contributions to the elements $\boldsymbol{S}_i$.

Element-By-Element (EBE) preconditioners were introduced by Hughes, Levit and Winget (1983) and Ortiz, Pinsky and Taylor (1983) and have been successfully applied in a number of applications in engineering and physics (see, for example, Hughes, Ferencz and Hallquits, 1987, and Erhel, Traynard and Vidrascu, 1991), and in the solution of partially separable linear systems (see Daydé, L'Excellent and Gould, 1997*b*). A detailed analysis of this technique for certain problems arising from partial-differential equations is given by Wathen (1989).

Let $\Delta(\boldsymbol{S})$ and $\Delta(\boldsymbol{S}_i)$ be the diagonal parts of $\boldsymbol{S}$ and $\boldsymbol{S}_i$ respectively. Then we obtain the EBE preconditioner

$$\boldsymbol{P}_{EBE} \;=\; \Delta(\boldsymbol{S})^{1/2} \left\{ \prod_{i=1}^{n_e} \boldsymbol{L}_i \prod_{i=1}^{n_e} \boldsymbol{D}_i \prod_{i=n_e}^{1} \boldsymbol{L}_i^T \right\} \Delta(\boldsymbol{S})^{1/2},$$

where $\boldsymbol{L}_i$ and $\boldsymbol{D}_i$ are the LDL^T factors of the Winget terms

$$\boldsymbol{L}_i \boldsymbol{D}_i \boldsymbol{L}_i^T = \boldsymbol{W}_i \equiv \boldsymbol{I} + \Delta(\boldsymbol{S})^{-1/2} (\boldsymbol{S}_i - \Delta(\boldsymbol{S}_i)) \Delta(\boldsymbol{S})^{-1/2}$$

4.4 Subspace-by-Subspace preconditioners

More recently, Daydé, Décamps and Gould (1997*a*) have introduced a class of so-called Subspace-by-Subspace (SBS) preconditioners. These techniques consist of regrouping the rank-one terms in (4.15), and forming and applying a QR factorization of the each of the resulting low-rank matrices. If we use the decomposition of the Schur complement into n_b rank-one terms as indicated by (3.14), we may rewrite $\boldsymbol{S}$ as

$$\boldsymbol{S} = \Delta(\boldsymbol{S})^{1/2} \left(\boldsymbol{I} + \sum_{i=1}^{n_b} \boldsymbol{E}_i \right) \Delta(\boldsymbol{S})^{1/2}, \tag{4.16}$$

where $\boldsymbol{E}_i = \boldsymbol{W}_i - \boldsymbol{I}$, $\boldsymbol{W}_i = \boldsymbol{D}_i + \boldsymbol{h}_i \boldsymbol{h}_i^T$, $\boldsymbol{D}_i = \boldsymbol{I} - \Delta(\boldsymbol{S})^{-1/2} \Delta(\boldsymbol{s}_i \boldsymbol{s}_i^T) \Delta(\boldsymbol{S})^{-1/2}$, and $\boldsymbol{h}_i = \Delta(\boldsymbol{S})^{-1/2} \boldsymbol{s}_i$. If, instead, we group together the rank-one terms into r_n sets, we obtain

$$\boldsymbol{S} = \Delta(\boldsymbol{S})^{1/2} \left(\boldsymbol{I} + \sum_{j=1}^{r_n} \boldsymbol{E}_j \right) \Delta(\boldsymbol{S})^{1/2} \tag{4.17}$$

where $\boldsymbol{E}_j = \boldsymbol{W}_j - \boldsymbol{I}$, $\boldsymbol{W}_j = \boldsymbol{D}_j + \boldsymbol{H}_j \boldsymbol{H}_j^T$, $\boldsymbol{H}_j = [\boldsymbol{h}_{j_1}, \ldots, \boldsymbol{h}_{j_n}]$ and $\boldsymbol{D}_j = \boldsymbol{I} - \Delta(\boldsymbol{S})^{-1/2}$ $\sum_{i=i_1}^{i_n} [\Delta(\boldsymbol{s}_i \boldsymbol{s}_i^T)] \Delta(\boldsymbol{S})^{-1/2}$. Replacing the summation in (4.17) by a product, an approximation

of $\boldsymbol{S}$ is

$$\boldsymbol{P}_1 = \Delta(\boldsymbol{S})^{1/2} \prod_{j=1}^{r_n} \boldsymbol{W}_j \Delta(\boldsymbol{S})^{1/2}$$

To use this as a preconditioner, we merely need to be able to invert the matrices $\boldsymbol{W}_j$. To this end, we can use the factorization of the diagonal matrix $\boldsymbol{D}_j$ to obtain

$$\boldsymbol{W}_j = \boldsymbol{D}_j^{1/2} \left\{ \boldsymbol{I} + \boldsymbol{C}_j \boldsymbol{C}_j^T \right\} \boldsymbol{D}_j^{1/2},$$

where $\boldsymbol{C}_j = \boldsymbol{D}_j^{-1/2} \boldsymbol{H}_j$. Thus we must factorize matrices – omitting the indices – of the form $\boldsymbol{I} + \boldsymbol{C}\boldsymbol{C}^T$, where $\boldsymbol{I}$ is a $n_s \times n_s$ matrix and $\boldsymbol{C}$ a $n_s \times l$ matrix (n_s is the dimension of the Schur complement $\boldsymbol{S}$, while l is the number of rank-one terms grouped together). One approach is to assemble the whole n_s by n_s matrix and use a LDL^T factorization — this is essentially what the EBE methods mentioned earlier do. However, since this ignores the inherent structure, we might instead use the QR factorization of the rank-l matrix

$$\boldsymbol{C} = \boldsymbol{Q} \begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix}, \quad \text{where} \quad \boldsymbol{Q} = \begin{pmatrix} \boldsymbol{Y} & \boldsymbol{Z} \end{pmatrix},$$

$\boldsymbol{Q}$ and $\boldsymbol{R}$ are, respectively, $n_s \times n_s$ orthonormal and $k \times k$ triangular matrices, and $\boldsymbol{Y}$ is the leading $n_s \times k$ submatrix of $\boldsymbol{Q}$. We then have that

$$\boldsymbol{I} + \boldsymbol{C}\boldsymbol{C}^T = \boldsymbol{Q} \begin{pmatrix} \boldsymbol{L} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \boldsymbol{Q}^T \boldsymbol{Q} \begin{pmatrix} \boldsymbol{L}^T & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \boldsymbol{Q}^T = \boldsymbol{M}\boldsymbol{M}^T$$

where $\boldsymbol{L}\boldsymbol{L}^T = \boldsymbol{I} + \boldsymbol{R}\boldsymbol{R}^T$ and

$$\boldsymbol{M} = \boldsymbol{Q} \begin{pmatrix} \boldsymbol{L} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \boldsymbol{Q}^T$$

Using the orthonormality of the columns of $\boldsymbol{Q}$, it immediately follows that the action of the inverses of the matrices $\boldsymbol{M}$ and $\boldsymbol{M}^T$ on the vector $\boldsymbol{v}$ can be expressed as

$$\begin{aligned} \boldsymbol{M}^{-1}\boldsymbol{v} &= \boldsymbol{Q} \begin{pmatrix} \boldsymbol{L}^{-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \boldsymbol{Q}^T \boldsymbol{v} = \begin{pmatrix} \boldsymbol{Y} & \boldsymbol{Z} \end{pmatrix} \begin{pmatrix} \boldsymbol{L}^{-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{Y}^T \\ \boldsymbol{Z}^T \end{pmatrix} \boldsymbol{v} \\ &= \left(\boldsymbol{I} + \boldsymbol{Y}(\boldsymbol{L}^{-1} - \boldsymbol{I})\boldsymbol{Y}^T \right) \boldsymbol{v} \quad \text{and} \\ \boldsymbol{M}^{-T}\boldsymbol{v} &= \left(\boldsymbol{I} + \boldsymbol{Y}(\boldsymbol{L}^{-T} - \boldsymbol{I})\boldsymbol{Y}^T \right) \boldsymbol{v}. \end{aligned}$$

As $(\boldsymbol{I} + \boldsymbol{C}\boldsymbol{C}^T)^{-1} = \boldsymbol{M}^{-T}\boldsymbol{M}^{-1}$, there is no need to assemble the matrix $\boldsymbol{M}$ to solve $\boldsymbol{M}^{-1}\boldsymbol{v}$. A possible preconditioner is then

$$\boldsymbol{P}_1 = \Delta(\boldsymbol{S})^{1/2} \prod_{j=1}^{r_n} \left[\boldsymbol{D}_j^{1/2} \boldsymbol{M}_j \boldsymbol{M}_j^T \boldsymbol{D}_j^{1/2} \right] \Delta(\boldsymbol{S})^{1/2}$$

but, to preserve symmetry, we prefer the Subspace-by-Subspace preconditioner

$$\boldsymbol{P}_{SBS} = \Delta(\boldsymbol{S})^{1/2} \prod_{j=1}^{r_n} \left[\boldsymbol{D}_j^{1/2} \boldsymbol{M}_j \right] \prod_{j=r_n}^{1} \left[\boldsymbol{M}_j^T \boldsymbol{D}_j^{1/2} \right] \Delta(\boldsymbol{S})^{1/2}, \tag{4.18}$$

which results from a simple reordering of the product in $\boldsymbol{P}_1$.

To find $\boldsymbol{M}^{-1}\boldsymbol{v}$ and $\boldsymbol{M}^{-T}\boldsymbol{v}$, it is sufficient to compute the $\boldsymbol{Y}$ part of $\boldsymbol{Q}$ in the QR factorization of $\boldsymbol{C}$. Thus we use a QR factorization based on the modified Gram-Schmidt algorithm (see for instance Golub and Loan, 1989), rather than a factorization based on Givens rotations or Householder transformations.

A simple calculation shows that the QR factorization is less expensive than assembling the total matrix $\boldsymbol{I} + \boldsymbol{C}\boldsymbol{C}^T$ and using a LDL^T factorization so long as k is less than $(\sqrt{5}-2) \times n_s$.

5 Experiments with matrix stretching

In this section, we investigate whether matrix stretching is an effective means of solving systems of the form (1.1)–(1.2). We first consider an artificial problem, which allows us to investigate isolate different aspects of the technique. We then consider its effect on practical problems.

5.1 Experiments on structured matrices with specified values

We experiment on block diagonal matrices with overlapping between two consecutive blocks. The blocks are constructed using random values so that each block has eigenvalues between 10^{-1} and 10^3. The percentage of overlap (number of variables shared by two elements over the total number of variables) is set to 10%, 20%, and 30%. Characteristics of the test examples are given in Table 5.1.

	10% overlap			20% overlap			30% overlap		
n_e	10	50	100	10	50	100	10	50	100
n	91	451	901	82	402	802	73	353	703
n_s	9	49	99	18	98	198	27	147	297

Table 5.1: Test problem characteristics. Key: n_e is the number of elements used, and n, the number of variables of the initial problem, n_s the number of variables in the Schur complement.

We compare the following methods:

1. CG on the initial system without preconditioning (**cg none**);
2. the same method with a diagonal preconditioner (**cg diag**);
3. the Schur complement method on the stretched system (**Schur none**); and
4. the Schur complement method with the Chan diagonal preconditioner (**Schur diag**).

The CG and its preconditioners are available with the PAREBE package (Daydé, L'Excellent and Gould, 1997*c*). The stopping criteria for all the methods is chosen so that $\|\boldsymbol{B}\,\boldsymbol{x} - \boldsymbol{b}\| \leq 10^{-9}\ \|\boldsymbol{b}\|$. For our experiments, the number of probing vectors used with the Chan diagonal preconditioner is 10% of the dimension of the Schur complement.

We give our results In Table 5.2. We report the number of iterations, the CPU time, the residual error ($\|\boldsymbol{B}\boldsymbol{x} - \boldsymbol{b}\|_\infty / \|\boldsymbol{B}\|_\infty \|\boldsymbol{b}\|_\infty$), and the component-wise error ($\max_{i=1,n} |\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i| / |\hat{\boldsymbol{x}}_i|$ where $\hat{\boldsymbol{x}}$ is the theoretical solution and $\boldsymbol{x}$ the computed one).

The time and number of iterations of the CG method increase as the number of elements increases. This is less the case when using the Schur complement method with and without preconditioning. The residual and component-wise errors are slightly better in the case of the CG

10% of overlap					
n_e	Method	#its	Time (s)	Residual	CW error
10	CG NONE	237	1.00	0.8×10^{-10}	0.7×10^{-6}
	Schur NONE	11	0.05	0.2×10^{-9}	0.3×10^{-7}
	CG DIAG	225	0.69	0.1×10^{-9}	0.7×10^{-6}
	Schur Chan	18	0.08	0.1×10^{-6}	0.5×10^{-4}
50	CG NONE	602	9.00	0.6×10^{-9}	0.3×10^{-5}
	Schur NONE	175	2.10	0.1×10^{-5}	0.4×10^{-4}
	CG DIAG	561	8.50	0.4×10^{-9}	0.1×10^{-5}
	Schur Chan	19	0.37	0.8×10^{-7}	0.2×10^{-5}
100	CG NONE	696	21.0	0.5×10^{-9}	0.2×10^{-5}
	Schur NONE	472	11.0	0.3×10^{-6}	0.5×10^{-4}
	CG DIAG	687	21.0	0.4×10^{-9}	0.3×10^{-5}
	Schur Chan	22	0.93	0.5×10^{-6}	0.5×10^{-5}
20% of overlap					
10	CG NONE	215	0.63	0.8×10^{-10}	0.1×10^{-5}
	Schur NONE	24	0.09	0.3×10^{-7}	0.8×10^{-6}
	CG DIAG	188	0.58	0.1×10^{-9}	0.9×10^{-7}
	Schur Chan	30	0.12	0.3×10^{-13}	0.6×10^{-12}
50	CG NONE	530	7.70	0.6×10^{-9}	0.4×10^{-5}
	Schur NONE	582	7.20	0.2×10^{-5}	0.7×10^{-3}
	CG DIAG	481	7.10	0.3×10^{-9}	0.8×10^{-5}
	Schur Chan	42	0.73	0.3×10^{-6}	0.6×10^{-4}
100	CG NONE	602	16.0	0.3×10^{-9}	0.5×10^{-5}
	Schur NONE	1464	36.0	0.5×10^{-6}	0.7×10^{-4}
	CG DIAG	517	15.0	0.4×10^{-9}	0.9×10^{-5}
	Schur Chan	53	1.90	0.2×10^{-6}	0.3×10^{-4}
30% of overlap					
10	CG NONE	179	0.56	0.1×10^{-9}	0.5×10^{-6}
	Schur NONE	50	0.14	0.6×10^{-8}	0.6×10^{-5}
	CG DIAG	153	0.44	0.1×10^{-9}	0.1×10^{-7}
	Schur Chan	32	0.13	0.2×10^{-8}	0.4×10^{-7}
50	CG NONE	428	5.70	0.3×10^{-9}	0.3×10^{-5}
	Schur NONE	1146	16.0	0.2×10^{-5}	0.8×10^{-4}
	CG DIAG	324	4.30	0.3×10^{-8}	0.3×10^{-5}
	Schur Chan	69	1.20	0.4×10^{-6}	0.1×10^{-3}
100	CG NONE	499	13.0	0.4×10^{-9}	0.7×10^{-5}
	Schur NONE	3384	90.0	0.2×10^{-5}	0.3×10^{-3}
	CG DIAG	401	11.0	0.4×10^{-9}	0.4×10^{-5}
	Schur Chan	75	2.80	0.1×10^{-5}	0.1×10^{-3}

Table 5.2: Number of iterations, execution time, residual, and component-wise error of CG applied to the initial system and on the stretched system relatively to the number of elements and the degree of overlap.

method. As the percentage of overlap increases to 30%, the execution times for the preconditioned Schur complement method and PCG applied to the initial system are very similar, while the Schur complement approach without preconditioning performs poorly. As one might expect, as the percentage of overlap increases, the Schur complement approach becomes less and less effective because of its increased size.

5.1.1 Preliminary parallel experiments

In Table 5.3, we study the parallel potential of the stretching method on our artificial examples without preconditioning. We give the execution time in seconds for solving the problem with n_e equal to 10 using one processor (1 Proc) and eight processors (8 Proc) of an Alliant FX/80. The **cg none** method is parallelised using a coloring algorithm (see Daydé et al. (1997*c*)). The speed-ups achieved using matrix stretching are almost double that achieved by CG on the initial system, and are close to 6 on the 8 processors of the ALLIANT FX/80, a very good result for such a computer. This clearly demonstrates the potential for parallelisation of matrix stretching techniques.

		Execution time (s)		
Solver	Percent. Over.	1 Proc	8 Proc	Speed Up
cg none	10 %	17.0	5.4	3.2
on	20 %	14.0	4.1	3.4
initial system	30 %	11.0	3.1	3.5
Schur none	10 %	2.6	0.5	5.7
on	20 %	5.8	1.0	5.9
stretched system	30 %	8.2	1.4	5.9

Table 5.3: Preliminary parallel experiments on the Alliant FX/80.

5.2 Experiments on matrices from the Harwell-Boeing collection

5.2.1 Description of the test problems

We have also experimented on some of the unassembled matrices available from the Harwell-Boeing collection (see Duff, Grimes and Lewis, 1989, 1992). Since only the structure of these problems is available, the numerical values of the elemental matrices have been set to random values, in the same way as in the previous section.

We have generated, for the three structures `CEGB2802`, `LOCK3491`, and `MAN5976`, four test problems with increasing condition numbers. These problems are termed `CEGB2802-1`, `CEGB2802-2`, `CEGB2802-3`, and `CEGB2802-4` for `CEGB2802`, and similarly for the other problems.

In order to have an efficient Schur complement method, the size of the Schur complement should ideally be much smaller than the size of the initial problem. For this reason, we have used element amalgamation in our test problems. This is achieved with the merge algorithm introduced by Daydé et al. (1997*b*, 1997*c*). In this algorithm, elements are amalgamated until a certain threshold is reached. The lower the threshold, the smaller is the Schur complement, as we illustrate in Figure 5.3 for our three classes of test problems. For our experiments, the threshold used by the amalgamation algorithm is -10^5.
The characteristics of the problems used in our tests are given in Table 5.4.

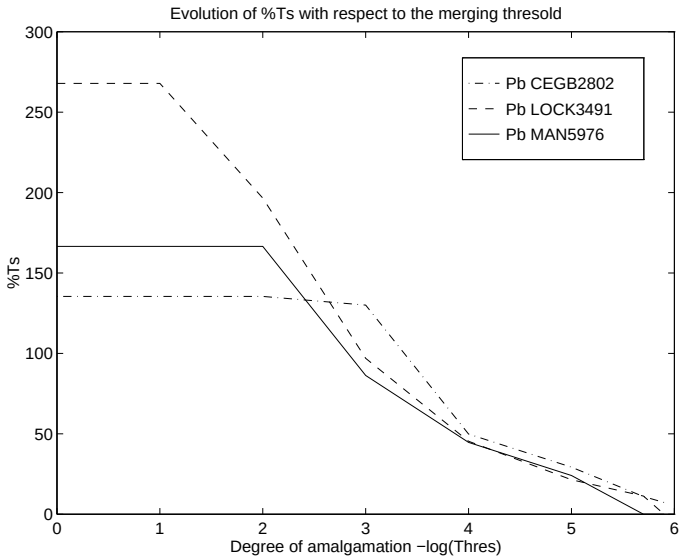

Figure 5.3: Size of the Schur complement with respect to the amalgamation threshold on the test problems (expressed as $100 n_s/n$).

Problem	n	n_e	min	max	$aver$	κ	n_s	$\%T_s$	κ_S
CEGB2802-1	2694	12	267	432	290.3	2.1×10^2	789	29.3	9.4×10^1
CEGB2802-2	2694	12	267	432	290.3	1.1×10^4	789	29.3	2.7×10^3
CEGB2802-3	2694	12	267	432	290.3	5.2×10^5	789	29.3	8.6×10^4
CEGB2802-4	2694	12	267	432	290.3	2.3×10^7	789	29.3	3.4×10^6
LOCK3491-1	3416	11	281	468	377.1	3.8×10^2	732	21.4	1.2×10^2
LOCK3491-2	3416	11	281	468	377.1	1.2×10^4	732	21.4	1.2×10^3
LOCK3491-3	3416	11	281	468	377.1	4.3×10^5	732	21.4	5.4×10^4
LOCK3491-4	3416	11	281	468	377.1	1.5×10^7	732	21.4	1.3×10^6
MAN5976-1	5882	22	226	445	331.7	1.7×10^2	1416	24.1	8.0×10^1
MAN5976-2	5882	22	226	445	331.7	8.8×10^3	1416	24.1	1.9×10^3
MAN5976-3	5882	22	226	445	331.7	4.6×10^5	1416	24.1	5.7×10^4
MAN5976-4	5882	22	226	445	331.7	1.8×10^7	1416	24.1	1.9×10^6

Table 5.4: Characteristics of the test problems. Key: n is the number of variables used in the initial problem; n_e is the number of blocks in the initial system; min, max and $aver$ are respectivly the minimum, maximum and average size of the blocks in the initial system; κ is the condition number of the problems; n_s is the number of variables used for the Schur complement; $\%T_s$ is the percentage 100 n_s / n; and κ_S is the condition number of the Schur complement

5.2.2 Numerical experiments

We compare the following solution techniques:

- Solution of the initial system $\boldsymbol{B}\boldsymbol{x} = \boldsymbol{b}$:

  CG Diag — PCG with a diagonal preconditioner, and
  CG EBE — PCG with the Element-by-Element preconditioner EBE.

- Solution of the stretched system (1.3):

  NONE — unpreconditioned CG applied to $\boldsymbol{S}\boldsymbol{\lambda} = \boldsymbol{s}$,
  CHANd — PCG with the Chan diagonal preconditioner applied to $\boldsymbol{S}\boldsymbol{\lambda} = \boldsymbol{s}$,
  CHANb — PCG with the CHAN band preconditioner applied to $\boldsymbol{S}\boldsymbol{\lambda} = \boldsymbol{s}$,
  EBE — PCG with an EBE preconditioner applied to $\boldsymbol{S}\boldsymbol{\lambda} = \boldsymbol{s}$,
  SBS — PCG with a Subspace-by-Subspace preconditioner applied to $\boldsymbol{S}\boldsymbol{\lambda} = \boldsymbol{s}$,
  Band — PCG with a band preconditioner applied to $\boldsymbol{S}\boldsymbol{\lambda} = \boldsymbol{s}$, and
  Dir — a direct solver applied to $\boldsymbol{S}\boldsymbol{\lambda} = \boldsymbol{s}$.

When solving the initial system (1.1) by an iterative method, we stop as soon as

$$\|\boldsymbol{B}\boldsymbol{x} - \boldsymbol{b}\| \leq 10^{-9} \|\boldsymbol{b}\|;$$

when we solve the stretched system (1.3) using Algorithm 3.1, Step 2 is terminated as soon as

$$\|\boldsymbol{S}\boldsymbol{\lambda} - \boldsymbol{s}\| \leq 10^{-10} \|\boldsymbol{s}\|.$$

In our experiments, the number of probing vectors used for the Chan diagonal preconditioner is $0.1n_s$, and the semi-bandwidth of the approximation of the Schur complement for the Chan band preconditioner is $0.2n_s$. For the band preconditioner, we report results using a matrix of semi-bandwidth of $0.2n_s$, while the number of rank-one terms regrouped in the Subspace-by-Subspace preconditioner is also $0.2n_s$. Other values were tried for the various preconditioners but those reported here achieved the best compromise over all the problems considered in our experiments.

In the Tables 5.5, 5.6, and 5.7, we report the results of the tests performed on our test set. *Ttotal* gives the total time required to solve the linear system, *Tconv* gives the time to solve the system involving the Schur complement, *Tcons* is the construction time for the Schur complement preconditioner and *Iter* gives the number of iterations required to solve the Schur complement system. The experiments were performed on a SUN workstation with a 125 MHZ HyperSPARC processor.

We make the following observations. Firstly, solving the systems using CG without preconditioning is the least effective method of all. Unfortunately, the Chan diagonal preconditioner also appears to be both rather costly and ineffective for these general matrices. Its banded sister is rather more effective, but is in general even more costly. The Band 20% method performs a comparable number of iterations, but is slightly cheaper than the Chan variant.

Secondly, the EBE preconditioner appears to be reliable and efficient with all our test problems, for both the initial and stretched systems. However, the cost of constructing the preconditioner may be high, the method only pays off overall for the more ill conditioned problems. (Daydé et al. 1997*b*) observed a similar behaviour under more general circumstances. The main drawback when using this preconditioner on the Schur complement is that its memory requirements are sometimes prohibitive, since it is necessary to compute the unassembled structure of the Schur complement

CEGB2802, Threshold = -1e-5, n = 2694, n_s = 789						
System	Preconditioner		CEGB2802-1	CEGB2802-2	CEGB2802-3	CEGB2802-4
Stretched	NONE	Ttotal	17.5	66.4	282.4	609.6
		Tconv	14.3	63.3	279.3	606.5
		Tcons	0.0	0.0	0.0	0.0
		Iter	92	408	1822	3945*
	CHANd	Ttotal	23.0	40.5	91.7	169.2
		Tconv	9.2	26.5	76.8	165.2
		Tcons	10.7	10.8	11.4	10.9
		Iter	59	169	446	969
	CHANb 20%	Ttotal	41.3	42.6	47.0	134.0
		Tconv	0.9	1.6	2.4	92.3
		Tcons	37.1	37.8	40.9	38.5
		Iter	4	7	11	448
	EBE	Ttotal	**10.9**	**13.4**	17.5	**21.0**
		Tconv	2.7	4.8	8.1	12.1
		Tcons	0.5	0.5	0.5	0.5
		Iter	14	24	38	62
	SBS 20%	Ttotal	17.0	20.6	28.5	39.9
		Tconv	5.5	8.9	16.0	28.0
		Tcons	4.8	4.8	5.0	4.9
		Iter	14	24	40	76
	Band 20%	Ttotal	13.2	**13.4**	**14.8**	107.6
		Tconv	1.0	1.3	1.6	94.7
		Tcons	5.2	5.3	5.6	5.6
		Iter	4	6	7	451
	Dir	Ttotal	24.7	24.9	24.6	23.7
		Tfact	6.2	6.5	6.3	6.4
		Tcons	15.3	15.3	15.1	14.2
		Iter	1	1	1	1
Initial	CG Diag	Ttotal	**5.5**	25.3	132.9	668.3
		Iter	47	234	1245	6150
	CG EBE	Ttotal	6.6	**15.7**	**58.1**	**241.7**
		Iter	13	48	211	915

Table 5.5: A comparison of the preconditioners on variants of CEGB2802.

from the rank-one contribution terms. An extreme example is when one element involves all the variables, as then the EBE method will require more memory than a direct solver.

When this occurs a Subspace-by-Subspace preconditioner may be particularly appropriate since its storage requirements will then be significantly lower. It is very close to EBE in terms of the number of iterations required for convergence, but, for the examples here, rather more expensive to use. We do not report here variations in the number of rank-one terms, n_g, which are grouped together for SBS — n_g is fixed to $0.2n_s$ in table 5.7. Empirically, this value has proved to be generally effective, but for MAN5976, the construction time crucially depends on n_g, and a value of $n_g = 0.05n_s$ proves to be better for the three first condition numbers (*Ttotal* decreases from 178.2 to 34.6 seconds for the problem MAN5976-1 with $n_g = 0.05n_s$). Since this preconditioner has only recently been proposed, it is clear that further experimentation is needed really to assess its effectiveness. For the cases considered here EBE appears to have an edge on SBS.

Thirdly, it is clear here that the direct factorization of the Schur complement is a very appealing alternative when it is feasible. In our experiments it is often the best option, but EBE proves to be almost as competitive. We would not expect that, for larger problems, the method will not be as attractive, as the Schur complement, while having considerable hidden structure, is quite

LOCK3491, Thres = -1e-5, n = 3416, n_s = 732						
System	Preconditioner		LOCK3491-1	LOCK3491-2	LOCK3491-3	LOCK3491-4
Stretched	NONE	Ttotal	27.1	74.0	204.6	582.0
		Tconv	22.7	69.6	200.2	577.1
		Tcons	0.0	0.0	0.0	0.0
		Iter	91	280	806	2105
	CHANd	Ttotal	30.5	46.5	104.3	351.9
		Tconv	11.4	26.2	83.8	331.4
		Tcons	14.6	15.6	15.9	15.7
		Iter	43	97	314	1287
	CHANb 20%	Ttotal	57.5	64.2	98.9	125.0
		Tconv	2.2	4.5	41.1	69.5
		Tcons	50.8	55.1	53.0	51.0
		Iter	7	14	133	232
	EBE	Ttotal	**16.1**	**20.4**	**24.5**	**31.6**
		Tconv	4.1	7.5	12.0	19.9
		Tcons	0.5	0.5	0.5	0.5
		Iter	14	24	40	65
	SBS 20%	Ttotal	24.3	33.2	40.0	56.5
		Tconv	7.1	14.0	22.5	39.7
		Tcons	7.0	7.9	7.0	6.7
		Iter	14	25	43	80
	Band 20%	Ttotal	18.5	20.9	57.4	86.5
		Tconv	2.2	3.8	40.1	69.0
		Tcons	6.2	6.3	6.4	6.2
		Iter	7	12	133	229
	Dir	Ttotal	29.8	29.8	29.9	31.9
		Tfact	5.0	5.0	5.1	5.4
		Tcons	20.4	20.4	20.5	22.0
		Iter	1	1	1	1
Initial	CG Diag	Ttotal	**6.7**	26.3	92.4	270.2
		Iter	40	161	569	1678
	CG Ebe	Ttotal	8.7	**13.6**	**23.0**	**40.8**
		Iter	11	23	47	92

Table 5.6: A comparison of the preconditioners on variants of LOCK3491.

often dense. The limitation of these "direct" preconditioners lies in the fact that they do not take advantage of the structure of the Schur complement.

Finally, from a numerical point of view, the matrix stretching approach is highly interesting because of the observation we made in Table 5.4 that the Schur complement is, for the examples we considered, always better conditioned than the initial matrix. We noted that by amalgamating more and more of the elements from the problems CEGB2802 and LOCK3491, the condition number of the Schur complement also improved. Once the condition number of the initial problem is large enough, matrix stretching appears to be more efficient than directly attacking the initial system.

6 Conclusions

In this paper, we have demonstrated the advantages of using a block stretching method in combination with a Schur complement solution approach. Such a method can be applied on the symmetric unassembled matrices arising from finite element problems or from partially separable optimization. The main benefit of matrix stretching appears to be that on ill-conditioned systems, the condition number of the Schur complement is lower than that of the initial system, and thus

MAN5976, Thres = -1e-5, n = 5882, n_s = 1416						
System	Preconditioner		MAN5976-1	MAN5976-2	MAN5976-3	MAN5976-4
Stretched	NONE	Ttotal	38.2	135.6	501.6	1695.8
		Tconv	32.7	130.1	496.1	1690.2
		Tcons	0.0	0.0	0.0	0.0
		Iter	84	337	1287	4368
	CHANd	Ttotal	72.6	121.7	264.5	1177.3
		Tconv	24.5	74.3	217.1	1132.0
		Tcons	42.1	41.5	41.5	39.7
		Iter	59	181	543	2913
	CHANb 20%	Ttotal	174.5	179.7	293.6	480.4
		Tconv	3.7	9.6	125.0	314.0
		Tcons	165.0	164.3	162.7	160.9
		Iter	6	17	231	588
	EBE	Ttotal	**25.4**	**33.3**	**49.4**	80.6
		Tconv	8.3	16.7	32.8	63.9
		Tcons	0.9	0.8	0.9	0.8
		Iter	17	36	73	136
	SBS 20%	Ttotal	178.2	181.4	212.7	279.1
		Tconv	19.7	38.9	80.8	112.2
		Tcons	143.9	127.6	118.0	152.7
		Iter	17	38	83	161
	Band 20%	Ttotal	40.0	42.6	45.2	368.9
		Tconv	3.7	5.9	9.5	331.0
		Tcons	22.2	22.4	21.7	22.8
		Iter	6	10	17	573
	Dir	Ttotal	67.6	67.6	68.2	**71.6**
		Tfact	31.6	31.7	32.3	34.4
		Tcons	30.4	30.5	30.4	31.6
		Iter	1	1	1	1
Initial	CG Diag	Ttotal	**12.2**	51.0	196.4	632.0
		Iter	44	187	720	2482
	CG EBE	Ttotal	14.3	**28.1**	**59.8**	**124.6**
		Iter	13	36	86	190

Table 5.7: A comparison of the preconditioners on variants of MAN5976.

less sophisticated preconditioned may be required. The other main benefit of matrix stretching is obviously its potential for parallelisation, since the diagonal blocks within the stretched system can be factorized independently. The challenge is to make sure that the iterative solution step on the Schur complement is also effectively parallelised, and EBE and SBS preconditioners seem ideal in this respect.

Naturally, we do not expect such an approach to be efficient on all kinds of partially separable linear systems, but believe that we have demonstrated that, at least in some cases, there is considerable benefit to be gained from stretching. Amalgamation algorithms such as the ones described in (Daydé et al. 1997*b*) are of great importance since they may provide a decrease in size of the Schur complement. The most difficult problem is still to identify efficient preconditioners for the Schur complement. The Element-by-Element preconditioners, particularly the Subspace-by-Subspace one, offers some opportunity of taking advantage of the hidden structures within the Schur complement.

Further experiments on real problems from the Harwell-Boeing collection and from the CUTE collection of optimization problems (see Conn et al., 1992 and Bongartz, Conn, Gould and Toint, 1993) are currently being performed.

References

Alvarado, F. L. (1997), 'Matrix enlarging methods and their application', *BIT* **37**(3), 473–505.

Andersen, K. D. (1996), 'A modified schur-complement method for handling dense columns in interior-point methods for linear programming', *ACM TOMS* **22(3)**, 348–356.

Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D. (1990), Lapack : A portable linear algebra library for high-performance computers, Technical Report Report CS-90-105, Computer Science Departement, University of Tennessee. Technical Report LAPACK Working Note 20.

Axelsson, O. (1996), *Iterative Solution Methods*, Cambridge University Press, Cambridge, USA.

Bongartz, I., Conn, A. R., Gould, N. I. M. and Toint, P. L. (1993), CUTE: Constrained and Unconstrained Testing Environment, Technical Report TR/PA/93/10, CERFACS, Toulouse, France.

Chan, T. and Goovaerts, D. (1988), A note on the efficiency of domain decomposed incomplete factorizations, Technical Report CAM 88–18, Department of Mathematics, UCLA, Los Angeles CA 90024–1555.

Chan, T. and Mathew, T. (1992), 'The interface probing technique in domain decompositi on', *SIAM J. Matrix Anal. Applics.* **13**, 212–238.

Coleman, T. and Moré, J. (1984), 'Estimation of sparse hessian matrices and graph coloring problems', *Mathematical Programming* **28**, 243–270.

Conn, A. R., Gould, N. I. M. and Toint, P. L. (1992), *LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A)*, number 17 *in* 'Springer Series in Computational Mathematics', Springer Verlag, Heidelberg, Berlin, New York.

Curtis, A. R., Powell, M. J. and Reid, J. K. (1974), 'On the estimation of sparse jacobian matrices', *J. Inst. Math. Appl.* **13**, pp. 117–120.

Daydé, M., Décamps, J. P. and Gould, N. I. M. (1997*a*), Subspace-by-subspace preconditioners for structured linear systems, Technical report, ENSEEIHT-IRIT, Toulouse, France. to appear.

Daydé, M. J., L'Excellent, J. Y. and Gould, N. I. M. (1997*b*), 'On the use of element-by-element preconditioners to solve large scale partially separable optimization problems', *SIAM Journal on Scientific Computing* **18(6)**, 1767–1787.

Daydé, M. J., L'Excellent, J. Y. and Gould, N. I. M. (1997*c*), PAREBE : parallel element-by-element preconditioners for the conjugate gradient algorithm, Technical report, ENSEEIHT-IRIT, Toulouse, France. To appear.

Duff, I. S., Grimes, R. G. and Lewis, J. G. (1989), 'Sparse matrix test problems', *ACM Trans. Math. Softw.* **15**, 1–14.

Duff, I. S., Grimes, R. G. and Lewis, J. G. (1992), Users' guide for the Harwell-Boeing sparse matrix collection, Technical Report TR/PA/92/86, CERFACS, Toulouse, France.

Erhel, J., Traynard, A. and Vidrascu, M. (1991), 'An element-by-element preconditioned conjugate gradient method implemented on a vector computer', *Parallel Computing* **17**, 1051–1065.

Gill, P. E., Murray, W. and Wright, M. H. (1981), *Practical Optmiization*, Academic Press, London and New York.

Golub, G. H. and Loan, C. F. V. (1989), *Matrix Computations*, 2nd edn, The Johns Hopkins University Press, Baltimore, MD.

Grcar, J. F. (1990), Matrix stretching for linear equations, Technical Report SAND90-8723, Sandia National Laboratories.

Griewank, A. and Toint, P. L. (1982*a*), 'Local convergence analysis for partitioned quasi-Newton updates', *Numerische Mathematik* **39**, 119–137.

Griewank, A. and Toint, P. L. (1982*b*), 'Partitioned variable metric updates for large structured optimization problems', *Numerische Mathematik* **39**, 429–448.

Haynsworth, E. (1968), 'On the schur complement', *Basel Math. Notes 20*.

Hughes, T. J. R., Ferencz, R. M. and Hallquits, J. O. (1987), 'Large-scale vectorized implicit calculations in solid mechanics on a CRAY X-MP/48 utilizing EBE preconditioned conjugate gradients', *Computational Methods in Applied Mechanics and Engineering* **61**, 215–248.

Hughes, T. J. R., Levit, I. and Winget, J. (1983), 'An element-by-element solution algorithm for problems of structural and solid mechanics', *Compututational Methods in Applied Mechanics and Engineering* **36**, 241–254.

Keyes, D. E. and Gropp, W. D. (1987), 'A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation', *SIAM J. Sci. Statist. Comput.* **8**, s166–s202.

Keyes, D. E. and Gropp, W. D. (1989), Domain decomposition techniques for the parallel solution of nonsymmetric systems of elliptic BVPS, *in* T. Chan, R. Glowinski, J. Périaux and O. . Widlund, eds, 'Domain Decomposition Methods', Society for Industrial and Applied Mathematics.

Ortiz, M., Pinsky, P. M. and Taylor, R. L. (1983), 'Unconditionally stable element-by-element algorithms for dynamic problems', *Compututational Methods in Applied Mechanics and Engineering* **36**, 223–239.

Powell, M. J. D. and Toint, P. L. (1979), 'On the estimation of sparse hessian matrices', *SIAM Journal on Numerical Analysis* **16**, 1060–1074.

Vanderbei, R. J. (1991), 'Splitting dense columns in sparse linear systems', *Linear Algebra Appl.* **152**, 107–117.

Wathen, A. J. (1989), 'An analysis of some element-by-element techniques', *Computational Methods in Applied Mechanics and Engineering* **74**, 271–287.

Zienkiewicz, O. C. (1977), *The Finite Element Method*, McGraw-Hill, London.