2005

Study and Design of an Intelligent Preconditioner Recommendation System

Shuting Xu
University of Kentucky

ABSTRACT OF DISSERTATION

Shuting Xu

The Graduate School

University of Kentucky

2005

STUDY AND DESIGN OF AN INTELLIGENT PRECONDITIONER RECOMMENDATION SYSTEM

ABSTRACT OF DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By

Shuting Xu

Lexington, Kentucky

Director: Dr. Jun Zhang, Associate Professor of Computer Science

Lexington, Kentucky

2005

ABSTRACT OF DISSERTATION

STUDY AND DESIGN OF AN INTELLIGENT PRECONDITIONER RECOMMENDATION SYSTEM

There are many scientific applications in which there is a need to solve very large sparse linear systems. The preconditioned Krylov subspace methods are considered the preferred methods in this field. The preconditioners employed in the preconditioned iterative solvers usually determine the overall convergence rate. However, choosing a good preconditioner for a specific sparse linear system arising from a particular application is the combination of art and science, and presents a formidable challenge for many design engineers and application scientists who do not have much knowledge of preconditioned iterative methods.

We tackled the problem of choosing suitable preconditioners for particular applications from a nontraditional point of view. We used the techniques and ideas in knowledge discovery and data mining to extract useful information and special features from unstructured sparse matrices and analyze the relationship between these features and the solving status of the sparse linear systems generated from these sparse matrices. We have designed an Intelligent Preconditioner Recommendation System, which can provide advice on choosing a high performance preconditioner as

well as suitable parameters for a given sparse linear system. This work opened a new research direction for a very important topic in large scale high performance scientific computing.

The performance of the various data mining algorithms applied in the recommendation system is directly related to the set of matrix features used in the system. We have extracted more than 60 features to represent a sparse matrix. We have proposed to use data mining techniques to predict some expensive matrix features like the condition number. We have also proposed to use the combination of the clustering and classification methods to predict the solving status of a sparse linear system. For the preconditioners with multiple parameters, we may predict the possible combinations of the values of the parameters with which a given sparse linear system may be successfully solved. Furthermore, we have proposed an algorithm to find out which preconditioners work best for a certain sparse linear system with what parameters.

KEYWORDS: Sparse Matrix, Preconditioning, Data Mining, Classification, Clustering.

Shuting Xu

7/31/05

STUDY AND DESIGN OF AN INTELLIGENT PRECONDITIONER RECOMMENDATION SYSTEM

By

Shuting Xu

Dr. Jun Zhang
Director of Dissertation

Dr. Grzegorz W. Wasilkowski
Director of Graduate Studies

7/31/05

RULES FOR THE USE OF DISSERTATIONS

Unpublished dissertations submitted for the Doctor's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgements.

Extensive copying or publication of the dissertation in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this dissertation for use by its patrons is expected to secure the signature of each user.

DISSERTATION

Shuting Xu

The Graduate School

University of Kentucky

2005

STUDY AND DESIGN OF AN INTELLIGENT PRECONDITIONER RECOMMENDATION SYSTEM

DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By

Shuting Xu

Lexington, Kentucky

Director: Dr. Jun Zhang, Associate Professor of Computer Science

Lexington, Kentucky

2005

To my parents and Shuhua.

ACKNOWLEDGEMENTS

The work with this dissertation has been extensive and trying, but in the first place it is exciting, instructive, and fun. Without help, support, and encouragement from other people, I would have never been able to finish this work. Here is my pleasure to express my gratitude to all of them.

First of all, I would like to thank my supervisor, Dr. Jun Zhang, for his inspiring and encouraging way to guide me to a deeper understanding of knowledge, and his invaluable comments during the whole work with this dissertation.

Besides my advisor, I would like to thank the rest of my Advisory Committee: Dr. Jerzy W. Jaromczyk, Dr. Alexander Dekhtyar and Dr. Qiang Ye who always give insightful comments and useful suggestions on my work. I would also like to thank the outside examiner Dr. Cai-Cheng Lu for his helpful comments on my dissertation.

Thanks also to all the friendly members in our lab who made the lab a great place to work. Let me say "thank you" to the following people: Dr. Chi Shen, Dr. Jeonghwa Lee, Dr. Kai Wang, Dr. Haiwei Sun, Dr. Samir Karaa, Mr. Ning Kang, Ms. Eun-Joo Lee, Mr. Wensheng Shen, Mr. Ning Cao, Mr. Hao Ji, Ms. Jie Wang, Mr. Yin Wang, Mr. Dianwei Han and other group members in our lab. Working together with all of you has been not only a unforgettable experience, but a great pleasure as well.

Last, but not least, I would like to thank my parents, for giving me life in the first place, for educating me, for unconditionally supporting and encouraging me to pursue my interests. Specially, I also would like to thank my husband for his love and support.

The research work with this dissertation was supported in part by:

- U.S. National Science Foundation (NSF) under grants ACR-0202934, ACR-0234270.
- Kentucky New Economy Safety and Security Initiative (NESSI) Consortium.
- Kentucky Science and Engineering Foundation.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

For many scientific applications there is a need to solve large sparse linear systems. For example, the applications to develop the next generation electromagnetics simulators [34], to track nerve fibers in human brains [31], or to simulate laminar diffusion flames [32], are just a few of them. The systems of large sparse linear equations arising from many of these applications can be written abstractly as:

$$A\,x = b, \tag{1.1}$$

where A is a real (or complex) valued sparse coefficient matrix of order n, x is the unknown vector and b is the known right-hand side vector. Generally there are two classes of methods available to solve this matrix equation. The first class is the direct sparse solution methods, represented by the Gauss elimination method. The second class is the iterative solution methods, of which the Krylov subspace methods are considered to be the most effective ones currently available [48]. For large sparse linear systems, direct methods are usually avoided due to their potentially large memory requirements and CPU consumption. Thus iterative methods become more and more attractive as the size of the coefficient matrices to be solved becomes larger and larger.

In a preconditioned iterative solver, which consists of an iterative method and a preconditioner, the quality of the preconditioner is of the most importance. Precon-

ditioning is a technique that transforms the original linear system into an equivalent one that is easier to solve with an iterative solver. If chosen properly and implemented properly, a preconditioning technique could improve the efficiency and robustness of the iterative methods, by reducing the number of overall iterations and the overall CPU time.

Choosing a good preconditioner for a specific sparse linear system is considered to be a combination of art and science. Some preconditioning techniques, such as the ILU0, rely on a fixed sparsity pattern, obtained from the original sparse matrix, or from a sparsified coefficient matrix by dropping small magnitude entries in certain applications [48]. Therefore the structural and numerical features of the matrix will greatly influence the performance of the preconditioned solvers. There are also other circumstances where a certain kind of preconditioners are only efficient in solving matrices from certain application domains with certain characteristics. For most people who are the end users of the preconditioned iterative methods, a good choice of suitable preconditioners for their particular application problems is difficult. If the matrix is large, it is very time-consuming and impractical to try all the preconditioners with all the possible parameter sets to find out which preconditioner with what parameters works best for the matrix. That is the reason why we proposed to build an Intelligent Preconditioner Recommendation System (IPRS). The system could recommend one or more suitable preconditioners for solving a given sparse linear system. After a matrix is submitted, the recommendations can be given in a short period of time. Such a system should be very useful in helping engineers and application scientists to choose suitable preconditioners for their particular applications [63].

The design and implementation of IPRS is very challenging as it requires knowledge and techniques from scientific computing, data mining, and knowledge discovery. First, we need to extract special features and structures of general sparse matrices

that may contribute positively or negatively to the performance of the preconditioned iterative solvers. Some matrix features like symmetry or diagonal dominance are positive features. A sparse linear system is easier to be solved if its coefficient matrix has such features. Some other features like the low diagonal sparsity rate are negative features. A sparse linear system usually may not be solved successfully by a certain preconditioner if its coefficient matrix has such negative features. Such features and structures may also provide us with some inspirations to construct better preconditioners. For example, we may design certain techniques to transform the negative features into positive features. Second, we will find out the relationships between these matrix features and the performance of various preconditioned iterative solvers, and set up models and/or patterns to represent these relationships. Finally, given a sparse matrix, the system should be able to recommend efficient preconditioned iterative solver(s) with appropriate parameters.

This chapter is organized as follows: The motivation of this dissertation research is explained in Section 1.1. We introduce the preconditioned Krylov methods in Section 1.2, including the solver and the preconditioners used in the experiments. An outline of the sparse matrices from which the sparse linear systems were generated is given in Section 1.3. The structure of the Intelligent Preconditioner Recommendation System is described in Section 1.4. Finally, the contributions and organization of the dissertation are listed in Section 1.5.

1.1 Motivation

Computational science and engineering (CSE) has been emerging as a new means of scientific research that utilizes the ever increasing power of computers to investigate and to understand the mysteries of science and engineering, that are very complex and are beyond the reach of traditional analytic and experimental methodologies.

Barring from revolutionary breakthroughs in fundamental computing technologies, the incremental advance in computing power alone is not sufficient to meet the need of very large scale high performance computer simulations. Use of better and smarter algorithms to reduce CPU time in computer simulations can be of considerable importance.

The innermost computational kernel of many scientific and engineering simulations is the solution of some large sparse linear systems of equations. The sparse linear systems arise frequently from discretized governing systems of partial differential equations in many computer simulation applications. The solution process of large sparse linear systems typically consumes a large fraction of the overall computational time in large scale high performance computer simulations.

Iterative solution techniques are often considered as the only viable means of dealing with large sparse linear systems [48]. Various such methods have been proposed in the past 50 years [21]. However, the sheer size of the coefficient matrices encountered in practical applications as well as their extreme ill-conditioning make current iterative solution methods unreliable and difficult to use. Unreliability is associated with the convergence of a given iterative method, which may fail for certain practical problems. Difficulty of use can be attributed to the fact that there are so many iterative methods to choose from, and some of the preconditioned iterative methods have many parameters to set. In fact, the traditional problem that an application scientist dealing with solving system (1.1) encounters is the *selection of an appropriate method* (and finding appropriate parameters) for solving a particular sparse matrix system. This problem is exacerbated by the fact that many of the engineers and research scientists who have to deal with solving sparse matrix systems are not familiar with the details of different iterative methods, and therefore, cannot even approach the selection problem.

There is an enlarging gap between the development of more and more sophisticated preconditioned iterative solvers by the computational linear algebra community and the ability to understand and to properly use these solvers by the application scientists and engineers to solve their more and more complex modeling and simulation problems. High performance computers and numerical algorithms will be less useful if they are not matched with the intended application problems. Our work will fill this gap and establish an information-based interface between the matrix solver developers and the application scientists and engineers.

Current approaches to recommending iterative solvers and preconditioners are qualitative and categorical. Although mathematical theorems can be proved to guarantee the convergence of certain preconditioned iterative methods for a given model problem [2, 22, 30], they are not very useful for solving problems encountered in practical applications. Thus a recommendation approach that permits tradeoffs and that can be built and modified incrementally, based on increased knowledge, is potentially useful. The success of the software environment approach for other problems in scientific computing [26] suggests that it can be very useful when applied to the problem of finding the right path to solve a sparse matrix.

There has been a considerable effort made by several researchers and organizations to collect various sparse matrices in order to use them for test purposes. The National Institute of Standard and Technology (NIST) has been playing a leading role in this endeavor and currently hosts one of the largest such repositories: MatrixMarket [38]. Several other collections have been contributed by engineers, scientists and numerical analysts, e.g., the well known Harwell-Boeing sparse matrix collection and the University of Florida Sparse matrix collections [12]. NIST has done some categorization work and published some preliminary information on these matrices. For each matrix this information includes its type, dimensions, condition number, nonzero structure,

etc. MatrixMarket is becoming a standard source of sparse test matrices for testing various direct and iterative solution methods.

However, *there is no information* regarding which matrix can be solved by what method using what parameters. Such information would be extremely helpful for application scientists as it would enable them to choose suitable sparse linear system solvers for certain class of applications. Furthermore, solving some of the matrices in the MatrixMarket does not provide much information to help an application scientist or engineer whose matrix in question may not match any of the matrices in the MatrixMarket *exactly*.

Knowing which matrices from a sparse matrix repository can be solved by what methods solves only part of the general problem, as this information alone is clearly insufficient for determining the appropriate solver and its parameters for an *arbitrary new sparse matrix*. *Our proposed solution* to this general problem lies in relating the new matrix to one or more matrices from the repository and using information about the solving status of these matrices to decide which solver to choose and how to configure it (selecting parameters).

In our point of view, the applicability of particular solvers to sparse matrices is guided for a large part by various *properties/features* of the matrices. We are not interested in solving some randomly generated sparse matrices. Sparse matrices that do require solution usually come from computer simulation applications. Such matrices should have certain properties/features unique to particular applications. Thus, information about presence or absence of some such features in a sparse matrix may suggest that a particular solver has a high chance of success (or failure) in solving the matrix. Both positive and negative information is equally important here.

Some of the features that may affect the applicability of different solvers are fairly intricate and complex (see Chapter 2.1 for a few such examples). To our knowledge,

no information on the presence/absence of these features (except for the most obvious information) in the matrices from the MatrixMarket repository is available. In fact, it is not even clear *what* features to look for in the matrices. This data repository could be much more useful if some intrinsic information about the matrix solving status can be made known to scientists and engineers.

Thus, our objective is to fill these information gaps, and to apply the information acquired to the general problem of finding an appropriate solver for an arbitrary sparse matrix. That is the motivation to build the Intelligent Preconditioner Recommendation System (IPRS). In the IPRS, given a sparse matrix, the recommendation about which preconditioners are the best to choose as well as the appropriate parameters will be suggested in a short period of time.

1.2 Preconditioned Krylov Methods

Many iterative linear system solvers are published in literature and used in practice, like Jacobi method, Gauss-Seidel method, Successive Overrelaxation method and Krylov Subspace methods [48]. Among them, the class of Krylov Subspace methods are the most promising general purpose iterative methods. The solver that we use in the experiments is PGMRES (Preconditioned Generalized Minimum Residual Method), which is one of the widely used Krylov Subspace methods.

There are also a variety of preconditioners, such as Jacobi, Successive Overrelaxation, and Symmetric Successive Overrelaxation preconditioners, Incomplete LU Factorization Preconditioners, Approximate Inverse Preconditioners, block preconditioners, and so on [48]. In our experiments, we choose some variants of Incomplete LU Factorization Preconditioners: ILU0, MILU, ILUK, ILUT, ILUTP, ILUD, and ILUDP. They are all constructed as a lower triangular matrix (L) and an upper triangular matrix (U), and the product matrix LU is used to approximate the original

matrix A. The code for the preconditioned solver and the preconditioners is taken from the SPARSKIT package [51].

1.2.1 The Solver PGMRES

Generalized Minimum Residual Method (GMRES) belongs to the group of Krylov subspace methods. Krylov subspace methods are based on some projection processes, both orthogonal and oblique, onto the Krylov subspaces [48].

A *Krylov subspace* is in the form

$$\mathcal{K}_m(A, v) = \text{span}\{v, Av, A^2v, \ldots, A^{m-1}v\}, \tag{1.2}$$

for some nonzero vector v. *Projection techniques* are to extract an approximate solution x_m to the linear equation (1.1) from an affine subspace $x^{(0)} + \mathcal{K}_m$ of dimension m, such that

$$b - Ax_m \perp \mathcal{L}_m, \tag{1.3}$$

where $\mathcal{L}_m$ is another subspace of dimension m. In a Krylov subspace method, we may choose $v = r^{(0)}$, where $r^{(0)} = b - Ax^{(0)}$ with $x^{(0)}$ being the initial guess to the solution. Different Krylov subspace method uses different choices of $\mathcal{L}_m$.

GMRES constructs two subspaces $\mathcal{K}_m$ and $\mathcal{L}_m = A\mathcal{K}_m$ by using $v = r^{(0)}/\|r^{(0)}\|_2$. Such choices result in a technique that minimizes residual norm over all vectors in $x^{(0)} + \mathcal{K}_m$.

If we apply GMRES to a preconditioned system, this yields *preconditioned* GMRES (PGMRES). In the case of GMRES, three options for applying the preconditioning operation, namely left, split, and right preconditioning, are available. Assume M is a preconditioner, the left-preconditioned GMRES algorithm is defined as the GMRES algorithm applied to the equivalent system:

$$M^{-1}A\,x = M^{-1}\,b,$$

if M is the result of a factorization of the form

$$M = LU,$$

then there is the option of using GMRES on the split-preconditioned system:

$$L^{-1}AU^{-1}u = L^{-1}b, \; x = U^{-1}u.$$

The right-preconditioned GMRES algorithm is based on solving:

$$AM^{-1}u = b, \quad \text{with} \quad u = Mx.$$

In our numerical experiments in this dissertation, right preconditioner and the right preconditioned GMRES are used. In this case, the Arnoldi loop builds an orthogonal basis of the right-preconditioned Krylov subspace:

$$\text{span}\{r_0, AM^{-1}r_0, (AM^{-1})^2 r_0, \ldots, (AM^{-1})^{m-1} r_0\}. \tag{1.4}$$

Algorithm 1.2.1 describes the GMRES algorithm with right preconditioning [48]. The right preconditioned GMRES gives rise to what is called a flexible variant, that is, a variant in which the preconditioner can change at each step. This capability can be very useful in some applications [48], but is not utilized in our current study.

ALGORITHM **1.2.1. GMRES with right preconditioning**.

1. Compute $r_0 = b - Ax_0, \beta = \| r_0 \|_2$, and $v_1 = r_0/\beta$
2. For $j = 1, \ldots, m$, Do:
3. Compute $w = AM^{-1}v_j$
4. For $i = 1, \ldots, j$, Do:
5. $h_{i,j} = (w, v_i)$
6. $w = w - h_{i,j}v_i$

7. EndDo

8. Compute $h_{j+1,j} = \| w \|_2$ and $v_{j+1} = w/h_{j+1,j}$

9. Define $V_m = [v_1, \ldots, v_m]$, $\bar{H}_m = \{h_{i,j}\}_{1 \leq i \leq j+1; 1 \leq j \leq m}$

10. EndDo

11. Compute $y_m = \text{argmin}_y \| \beta e_1 - \bar{H}_m y \|_2$, and $x_m = x_0 + M^{-1} V_m y_m$

12. If convergence criteria satisfied, Stop, else set $x_0 = x_m$ and goto 1.

1.2.2 The Preconditioners

Roughly speaking, a preconditioner is any form of implicit or explicit modification of the original coefficient matrix of a linear system which makes it "easier" to solve by a given iterative method [48, 1]. Consider a linear equation,

$$M^{-1} A x = M^{-1} b, \tag{1.5}$$

in which M is a nonsingular matrix of the same order of the matrix A. It is obvious that the equations (1.1) and (1.5) are equivalent and have the same solution. If M is close to A in some sense, and if it is inexpensive to solve $Mx = y$, for some vector y, the resulting system (1.5) can be solved by a Krylov subspace method and may require fewer steps to converge than solving the original system (1.1). System (1.5) is called a *preconditioned system*, and M is referred to as a *preconditioner*.

The usefulness of a preconditioner depends very much on how much it can reduce the condition number of the coefficient matrix, and on the cost of constructing and applying the preconditioning matrix M. The preconditioners used in this dissertation are constructed based on one or another form of incomplete Gauss elimination (incomplete LU factorization) [39, 48].

ILU0 and MILU0

ILU0 is the ILU factorization technique with no fill-in, which takes the zero pattern P to be precisely the zero pattern of the matrix A [48]. The definition of the ILU0 factorization is: "any pair of the matrices L (unit lower triangular) and U (upper triangular) so that the elements of $A - LU$ are zero in the locations of $NZ(A)$" [48], where $NZ(A)$ is the nonzero pattern of the matrix A. The procedure of the ILU0 factorization is described in Algorithm 1.2.2 [48].

ALGORITHM **1.2.2. The ILU0 algorithm**.

1. For $i = 2, \ldots, n$, Do:
2. For $k = 1, \ldots, i-1$, and for $(i,k) \in NZ(A)$ Do:
3. Compute $a_{ik} = a_{ik}/a_{kk}$
4. For $j = k+1, \ldots, n$, and for $(i,j) \in NZ(A)$ Do:
5. Compute $a_{ij} = a_{ij} - a_{ik}a_{kj}$
6. EndDo
7. EndDo
8. EndDo

ILU0 is simple to implement, its computational cost is inexpensive, and it is effective for some problems, such as those from low-order discretizations of scalar elliptic PDEs and diagonally dominant matrices. However, for more difficult and realistic problems the no-fill factorizations result in too crude an approximation to the original matrix, and more sophisticated preconditioners, which allow some fill-in in the incomplete factors, are needed [4].

In ILU0, the entries dropped out are just discarded. However, there are some techniques that attempt to reduce the effect of dropping by compensating for the

discarded entries. The modified ILU (MILU) factorization is to add up all the elements that have been dropped out in the kth loop of the algorithm. Then this sum is subtracted from the diagonal entry in U [48]. This strategy guarantees that the row sums of the matrix A are equal to those of LU. MILU often works well for PDEs. For other problems, such as problems with discontinuous coefficients, MILU usually is not better than its ILU counterparts, in general. MILU0 is an MILU factorization technique with no fill-in.

ILUK

ILUK is the ILU factorization with *level of fill* to be k. A level of fill is attributed to each element that is processed by Gaussian elimination and dropping is based on the value of the level of fill. The rationale is that the level of fill should be indicative of the size of an element: the higher the level, the smaller the element. The initial level of fill of an element a_{ij} of a sparse matrix A is defined by [48]:

$$\text{lev}_{ij} = \begin{cases} 0, & \text{if } a_{ij} \neq 0, \text{ or } i = j \\ \infty, & \text{otherwise.} \end{cases}$$

Each time the value of a_{ij} is modified in the Gaussian Elimination, its level of fill is updated by:

$$\text{lev}_{ij} = \min\{\text{lev}_{ij}, \text{lev}_{ik} + \text{lev}_{kj} + 1\}. \tag{1.6}$$

In ILUK, all fill-in elements whose level of fill does not exceed k are kept. The ILUK algorithm is shown in Algorithm 1.2.3 [48]. In our experiments, we only consider $k = 1, 2, 3$ respectively.

There are also some drawbacks of ILUK. For example, the amount of fill-in cannot be predicted in advance and the cost of updating the levels can be quite high. Furthermore, the algorithms may drop large size elements and result in an inaccurate incomplete factorization in some cases.

ALGORITHM **1.2.3. The ILUK algorithm**.

1. For all nonzero elements a_{ij}, define $lev(a_{ij}) = 0$
2. For $i = 2, \cdots, n$, Do:
3. For each $k = 1, \cdots, i-1$ and for $lev(a_{ik}) \leq k$, Do:
4. Compute $a_{ik} := a_{ik}/a_{kk}$
5. Compute $a_{i*} := a_{i*} - a_{ik}a_{k*}$
6. Update the levels of fill of the nonzero $a_{i,j}$'s using (1.6)
7. EndDo
8. Replace any element in row i with $lev(a_{ij}) > k$ with zero
9. EndDo

ILUT and ILUTP

Incomplete factorizations that rely on the levels of fill are blind to numerical values because elements that are dropped depend only on the structure of the matrix. This can cause some difficulties for realistic problems that arise in many applications [48]. A generic ILU algorithm with threshold (ILUT) can be derived from the IKJ version of Gaussian elimination by including a set of rules for dropping small elements [47]. The drop strategy works as follows: (1) The threshold in L and U is set by tol. Any element whose magnitude is less than tol (relative to the absolute value of the diagonal element of the current row) is dropped. (2) Keeping only the largest $lfil$ elements in the ith row of L and the largest $lfil$ element in the ith row of U (excluding the diagonal elements) [48]. The ILUT algorithm is depicted in Algorithm 1.2.4 [48].

ALGORITHM **1.2.4. The ILUT algorithm**.

0. Set $u_{1k} = a_{1k}, k = 1, \ldots, N$

1. For $i = 2, \ldots, N$, Do:
2. $w = a_{i*}$
3. For $k = 1, \ldots, i-1$ and when $w_k \neq 0$, Do:
4. $w_k = w_k / u_{kk}$
5. Apply a dropping rule to w_k
6. If $w_k \neq 0$ then
7. $w = w - w_k * u_{k*}$
8. EndIf
9. EndDo
10. Apply a dropping rule to row w
11. $l_{i,j} = w_j$ for $j = 1, \ldots, i-1$
12. $u_{i,j} = w_j$ for $j = i, \ldots, N$
13. $w = 0$
14. EndDo

The dual dropping strategy of the ILUT is implemented using the two parameters tol and $lfil$. The small entries of w_k (with respect to tol and relative to a certain norm of the current row) are dropped in Line 5. In Line 10, small entries are dropped again. A search algorithm is used to find out the largest $lfil$ entries in magnitude. These $lfil$ largest entries are kept, the others are dropped. Roughly speaking, $lfil$ can be viewed as a parameter that helps control memory usage, while tol helps reduce computational cost.

The ILUT may also fail for many of the matrices that arise from real applications. One of the reasons is that the ILUT procedure encounters a zero pivot. The ideal solution in this case is to use pivoting. ILUTP is ILUT with pivoting. Because of the data structure used in ILUT, column pivoting can be implemented rather easily [48]. The complexity of the ILUTP is virtually the same as that of ILUT. A tolerance

parameter *permtol* may be applied to help determine whether or not to permute columns: A nondiagonal element a_{ij} is a candidate for a permutation only when $permtol * |a_{ij}| > |a_{ii}|$. Furthermore, pivoting may be restricted to take place only within diagonal blocks of a fixed size.

ILUD and ILUDP

The preconditioner ILUD computes the ILU factorization with a standard threshold dropping: at the ith step of the elimination, an element $a(i, j)$ in row i is dropped if its magnitude is smaller than the product of the average magnitude of the elements in the ith row and the threshold *tol*. There is no control on memory size required for the factors as is done in ILUT. This preconditioner computes also various diagonal compensation just like MILU. A parameter α is used to control whether to use diagonal compensation or not.

ILUD also suffers from the same zero pivot problem as ILUT does. Like ILUTP, ILUDP is ILUD with column pivoting.

1.3 Sparse Linear Systems

The sparse linear systems, or the sparse matrices, analyzed in this dissertation are briefly described in this section. All the sparse linear systems were formed by assuming that the exact solution is a vector of all ones and the initial guess is a zero vector.

We use 319 sparse matrices in the experiments, all of which were downloaded from MatrixMarket [38]. The matrices come from the following industrial or engineering problems: dynamic or static analysis in structural engineering, air traffic control, astrophysics, power networks, finite-element structures problems in aircraft design, chemical engineering, circuit physics, economic modeling, power flow mod-

eling, simulation studies in computer systems, partial differential equations, nuclear reactor modeling, oil reservoir simulation, oceanic modeling, demography, petroleum engineering, etc.

Of the 319 matrices, 255 of them are of the type "RUA", which means they are real, unsymmetric and assembled matrices. The other 64 matrices are of the type "RSA", which means they are real, symmetric and assembled matrices. The size of the matrices varies from 27 to 23560. Some basic features of these 319 matrices are listed in the Appendix. The meaning of the features will be explained in the next chapter.

1.4 Structure of the Intelligent Preconditioner Recommendation System

The purpose of the Intelligent Preconditioner Recommendation System (IPRS) is to recommend the best preconditioners and the appropriate parameters for a given sparse matrix. Here "the best" means the sparse linear system generated from the sparse matrix can be solved using the least memory cost or time cost. The main structure of the IPRS is described in Figure 1.1. It works as follows: When a matrix is submitted through an interface, the preprocessing unit calculates the attributes of the matrix and passes on the computed attributes to the predicting unit. The predicting unit uses the attributes of the matrix and the models from Knowledgebase to predict which preconditioners would work best for the matrix as well as the suitable parameters. Then it sends the suggestions back to the interface. The whole processing time includes the time to calculate attributes and the time to make predictions, hence the user may expect a quick response time to get recommendations.

The other part of the work is done in background (the part within the dashed-line block in Figure 1.1). After a matrix is submitted, it is saved in the Database. The

attributes of the matrix are saved in the attribute table. Then the matrix is solved using different kinds of preconditioned iterative solvers and the results are stored in respective tables. The data mining unit periodically reconstructs models using the attribute table and the preconditioned solver tables. With more and more matrices stored in the database, the models in the Knowledgebase will be more accurate, thus the suggestions made by the system will be more and more helpful. The system is actually a closed-loop feedback system.

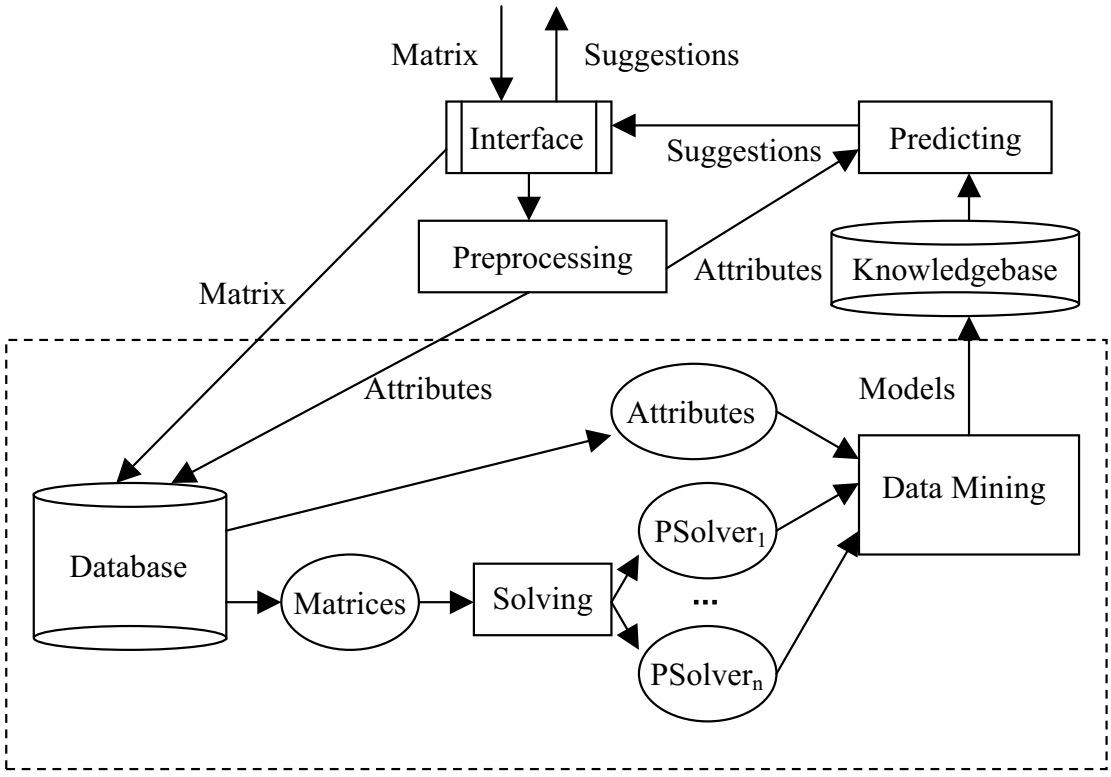


Figure 1.1: Structure of the Intelligent Preconditioner Recommendation System (IPRS).

The functions of each component of IPRS are described below:

- Interface. It is a friendly GUI. User may submit the matrices to be solved and get suggestions on which preconditioner to choose and what parameters are

suitable to be used according to user's performance expectations. It can also list the values of matrix features for the user's reference.

- Preprocessing unit. It computes the various attributes of the matrices. It also does some preprocessing on the values of the attributes. For example, for some attributes with large value ranges (e.g., $10^{-10} - 10^{30}$), it calculates their 10-base logarithm.

- Predicting unit. Using the models saved in the Knowledgebase, it predicts which preconditioners with what parameters may solve the linear systems. Then it chooses the preconditioners and parameters with the least memory and/or time cost and returns the suggestions to the interface.

- Data mining unit. It constructs models from the attribute table and preconditioned solver tables from Database and saves the models in Knowledgebase. The data mining algorithms used in the system include K-means clustering algorithm, Support Vector Machine Classification and Regression algorithms, some feature selection algorithms and the new algorithms proposed by us. Such algorithms will be introduced in later chapters.

- Solving unit. It generates a sparse linear system from a sparse matrix. It assumes that the exact solution is a vector of all ones and calculate the right hand side. Then it uses various preconditioned GMRES (PGMRES) with different parameters to solve the linear system and saves the results in respective PSolver tables.

- Database. It stores all the matrices submitted to the system. It has an Attribute table containing all the attributes of the matrices in the system. It also has some Psolve tables which store the solving status, time and memory cost,

and other parameters of solving all the matrices in the system using various preconditioners with different parameters.

- Knowledgebase. It stores the models obtained from the data mining unit.

1.5 Contributions of the Dissertation

Our research work for the dissertation is focused on studying and designing the Intelligent Preconditioner Recommendation System. The contributions of the dissertation are:

- We have designed the structure of the Intelligent Preconditioner Recommendation System. It is the first mathematical application tool that applies data mining techniques to find out the relationship between matrix features and the solving status of sparse linear systems using preconditioned iterative solvers. It is also the first time that data mining techniques instead of mathematical methods are used to predict the solving status of sparse linear systems and their memory costs.

- We have extracted more than 60 matrix features to represent a sparse matrix. Some of them are well-known, the others are derived from the construction of preconditioners or from our experience of solving sparse linear systems.

- We have proposed a new method to estimate condition number of a matrix. It is the first time that some expensive matrix features like condition number are estimated using data mining methods. Such techniques can be applied to estimate other expensive matrix features. The low cost of obtaining such matrix features may lead to the invention of more efficient mathematical methods or algorithms.

- We have compared some feature selection methods and proposed an efficient feature selection method utilizing the difference of a matrix and its preconditioned counterparts.

- We have proposed some new algorithms in predicting the solving status of sparse linear systems with different preconditioners. We can predict whether a sparse linear system may be solved by a certain preconditioned solver (preconditioner). If a failure status is predicted, we can also predict the reason why it fails.

- Instead of solving the sparse linear systems by PGMRES with all the possible parameters and comparing them, we have proposed a much faster method to find out which preconditioners work best for a certain sparse linear system with what parameters.

The organization of this dissertation is as follows:

- Matrix feature extraction is discussed in Chapter 2. The features of a matrix are directly related to the precision of the prediction system. Thus, the first problem we must address in building IPRS is determining the set of features that would represent the matrices in the database. In this chapter we first show some examples of special matrix features and then describe all the features used in the system.

- In Chapter 3, we analyze in a statistical way the relationship among the the solving status of the sparse linear systems, the choice of preconditioned solvers with different parameters, and the attributes of the matrices.

- Condition number is an important matrix feature as well as a widely used measure in numerical analysis and linear algebra. In Chapter 4, we propose to

use the Support Vector Regression methods to predict condition number from the matrix features. We also use some feature selection methods to further reduce the response time and improve accuracy.

- In order to construct the IPRS we need to predict whether a given sparse linear system can be solved by a certain preconditioned solver (preconditioner). In Chapter 5, we propose a prediction method which combines clustering and classification to predict whether a sparse linear system can be solved by PGMRES with preconditioner ILU0 or ILUK.

- Chapter 6 is an extension of Chapter 5 in which we predict whether a sparse linear system can be solved with a more complex preconditioner ILUT with two parameters.

- In Chapter 7, we propose an algorithm to choose the best preconditioner and parameters for a given sparse matrix. Our criteria for the best preconditioner and parameters are that the sparse linear system can be successfully solved by the chosen preconditioner and parameters, and at the same time, the memory cost and/or the time cost is the lowest.

- We conclude the dissertation in Chapter 8 and point out some directions for future work.

Chapter 2

Matrix Feature Extraction

For the various data mining algorithms we plan to use for establishing the possible connections between matrix features and the solving status to work, we need first to construct the appropriate dataset of these features for the known sparse matrices (such as those in the MatrixMarket repository). Thus, the first problem we must address in building IPRS is: determining the set of features that would represent the matrices in the database.

As will be seen from Section 2.1, some of the features of interest are quite complex, and determining their presence/absence in matrices may be a fairly involved task. This is exactly why it is important for us to determine the right set of features for representing sparse matrices. However, before we can make decisions on what features will be helpful to solve a matrix, we want to extract as many features as possible from the matrix. Then we will use some feature selection methods to choose the appropriate feature set, which will be explained in Chapter 4.

The organization of this chapter is: Some examples of special matrix features are given in Section 2.1. Most of the features used in the IPRS system are described in detail in Section 2.2.

2.1 Examples of Special Matrix Features

The features we are looking for to include in our IPRS can be quite intricate, possibly requiring a lot of computation. Some of these features are generally not known to the engineers who would need to solve a particular sparse matrix. A few examples below illustrate the complexity of such features.

2.1.1 Small Magnitude Pivots

Some well known features such as the diagonal dominance, positive definiteness, symmetry, etc., of a sparse matrix, may be obvious for application scientists. They are useful in choosing suitable iterative solvers [2]. We shall discuss some features that may not be so obvious but are useful in building ILU preconditioners.

The preconditioner M is constructed as the product of two matrices, L and U, where L is a lower triangular matrix with a unit diagonal and U is an upper triangular matrix. In general, the matrices L and U are some approximations of the lower-upper (LU) factors of A. The approximation is achieved by computing an incomplete LU factorization of A, i.e., we have a relation of the form

$$R = A - LU \tag{2.1}$$

where R is the error matrix representing the difference between the preconditioner $M = LU$ and the coefficient matrix A. It is well known that the size of R, denoted by $\|R\|$, affects the convergence rate of the preconditioned iterative methods [17]. For well behaved matrices, $\|R\|$ can be reduced by allowing more fill-in entries in the incomplete LU factorization. However, for ill conditioned matrices, merely allowing more fill-in entries in the ILU factorization is not enough to guarantee the quality of the preconditioner. This is because the stability of the preconditioner also affects the preconditioning effect [18]. In fact, it is shown in [62] that the following inequality

holds

$$\frac{\|w - \bar{w}\|}{\|w\|} \leq \|(LU)^{-1}\| \, \|R\|. \tag{2.2}$$

Here $\bar{w}$ is the correction from the preconditioning, and w is the exact correction required for an exact solution. In (2.2), $\|(LU)^{-1}\|$ measures the stability of the triangular solution processes involved in solving the preconditioning system. Thus, Inequality (2.2) shows that the quality of the preconditioning is directly related to the size of both $(LU)^{-1}$ and R.

The size of the entries in the L and U factors is mainly determined by the magnitude of the pivots in the process of the LU factorization [33]. The instability problems are often caused by small or zero pivots. These small magnitude pivots may lead to unstable and inaccurate factorizations [18]. The size of the entries in the LU factors may be very large which lead to inaccurate factorization due to large computer rounding errors [62].

Thus, our first identified *negative* feature of sparse matrices is **small magnitude pivots** in ILU factorization. Two natural questions following this identification are

- how to identify if a sparse matrix will encounter small magnitude pivots in an ILU factorization *before* we perform the ILU factorization?
- how to design a suitable strategy to transform this *negative* feature into a *positive* feature, i.e., without encountering small magnitude pivot in the ILU factorization?

Based on the study in [62], we anticipate that small magnitude pivots are likely to be encountered in an ILU factorization if the main diagonal entries in certain rows of the matrix is zero or small. Although this is not a 100% accurate statement, since small magnitude main diagonal entries may be enlarged or reduced during the ILU factorization, it can happen for many of the indefinite matrices arising from realistic

applications [62]. Such an inaccurate or not exact statement may satisfy the need of an application scientist or engineer.

The transformation strategy proposed in [62] is to identify these rows with small magnitude or zero main diagonals, and place these rows as the last few rows in the matrix via a symmetric permutation. The matrix A is permuted to have a block format

$$\tilde{A} = P\,A\,P^T = \begin{pmatrix} D & F \\ E & C \end{pmatrix},$$

where the submatrix D contains rows with relatively large magnitude main diagonals, and the submatrix C contains rows with small magnitude or zero main diagonals. A partial ILU factorization is then performed on the permuted matrix $\tilde{A}$ as

$$\tilde{A} = \begin{pmatrix} D & F \\ E & C \end{pmatrix} \approx \begin{pmatrix} I & 0 \\ E(LU)^{-1} & I \end{pmatrix} \begin{pmatrix} D & F \\ 0 & A_1 \end{pmatrix},$$

where $LU \approx D$ is the ILU factorization of the submatrix D, and $A_1 \approx C - E(LU)^{-1}F$ is the approximate Schur complement matrix with respect to the submatrix C. Since the submatrix D has large magnitude diagonal entries and is well conditioned, its ILU factorization LU should be stable. The formulation of the (approximate) Schur complement matrix A_1 avoids the factorization of the submatrix C in this first phase, thus avoids the potential instability problem that may be caused by small magnitude or zero pivots. The hope is that some of the small magnitude or zero diagonals may be enlarged when forming the (approximate) Schur complement matrix A_1. An ILU factorization on those rows of A_1 may be stable, the remaining small magnitude or zero diagonal rows are forced into another smaller Schur complement matrix A_2, and so on. This recursive procedure can be repeated for a few times. Each time, part of the matrix is *stably* factored. Analytic results and numerical experiments in [62] indicate that this multilevel recursive ILU factorization produces a much better ILU preconditioner than the standard ILU factorizations do.

2.1.2 Zero Diagonals in Matrices from 3D Coupled Navier-Stokes Simulations.

The 3D stationary Navier-Stokes equations for an incompressible viscous fluid can be written in velocity-pressure (primitive variable) form [61]. The finite element discretized sparse nonlinear system of equations can be written as

$$\nu \boldsymbol{C}\boldsymbol{u} + \boldsymbol{g}(\boldsymbol{u}) - \boldsymbol{B}^T\boldsymbol{p} = \boldsymbol{f}, \tag{2.3}$$

$$-\boldsymbol{B}\boldsymbol{u} = \boldsymbol{0}, \tag{2.4}$$

where $\boldsymbol{u}$ and $\boldsymbol{p}$ denote the nodal values of velocity and pressure. $\boldsymbol{C}$ and $\boldsymbol{B}$ are submatrices associated with the finite element discretization [9]. The nonlinear system of equations (2.3) and (2.4) is solved in a fully coupled sense using the Newton iteration method: given $\boldsymbol{S}_0$, for $n = 0, 1, 2, \ldots$, solve

$$\boldsymbol{J}_n \delta \boldsymbol{S}_n = -\boldsymbol{F}_n,$$

where

$$\boldsymbol{J}_n = \begin{pmatrix} \nu\boldsymbol{C} + \frac{\partial \boldsymbol{g}(\boldsymbol{u})}{\partial \boldsymbol{u}}\Big|_{\boldsymbol{u}_n} & -\boldsymbol{B}^T \\ -\boldsymbol{B} & \boldsymbol{0} \end{pmatrix} \tag{2.5}$$

is the Jacobian matrix. It is comprised of a symmetric part associated with the viscous Laplace operator and a nonsymmetric part due to the contribution $\partial \boldsymbol{g}(\boldsymbol{u})/\partial \boldsymbol{u}$ at $\boldsymbol{u}_n$ from the nonlinear convective term.

The matrix (2.5) is not expressed in the standard form coded in most implementations since the nodal variables have been re-ordered for convenience with all $\boldsymbol{u}$ variables listed first, followed by the p variables. In practice most finite element codes apply a different ordering of variables by nodes with velocity nodal values and pressures listed consecutively node by node so that the bandwidth is reduced. The $\boldsymbol{u}$ variable of the $(i+1)$st node is listed after the p variable of corner node i, and so on. In the case where additional transport variables are present their nodal values

are grouped with the velocity nodal values and precede them. For example, at node i we have (T_i, u_i, v_i, w_i) in the coupled thermal-viscous (variable viscosity) calculations [64].

It is noted by a few authors that an ILU factorization on the Jacobian matrix will encounter a zero pivot in the first few steps [11, 64, 65]. Thus it is important to permute all rows corresponding to the p variables to the last few rows of the matrix [62, 66]. If an ILU factorization *retaining the coefficient matrix structure* is applied on the permuted matrix, no zero pivot will be encountered. These are useful matrix features and information that may help build intelligent preconditioned iterative solvers.

2.1.3 Dense Blocks in Matrices from Semiconductor Diffusion Simulations.

One of the key steps in fabricating a semiconductor device such as a MOSFET is the "doping" of the semiconductor with impurities such as phosphorus, boron, and germanium using various implant techniques. The 5-species model for phosphorus diffusion proposed by Richardson and Mulvaney is representative of such a process modeling systems for impurity doping [45, 42, 65]. The associated system consists of 5 coupled reaction diffusion equations [45]. In [42] a class of semi-implicit Runge-Kutta methods is used to integrate the semi-discrete system of ODEs. The Runge-Kutta integrator can be written as

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \Delta t \sum_{i=1}^{q} \alpha_i \boldsymbol{k}_i,$$

where q denotes the number of stages and $\boldsymbol{k}_i$ has the form

$$\boldsymbol{A}\,\boldsymbol{k}_i = \left[I - \Delta t a_i \boldsymbol{J}(\boldsymbol{u}^n + \Delta t \sum_{j=1}^{i-1} c_j \boldsymbol{k}_j) \right] \boldsymbol{k}_i = \boldsymbol{F}(\boldsymbol{u}^n + \Delta t \sum_{j=1}^{i-1} b_j \boldsymbol{k}_j).$$

Here $\boldsymbol{J}$ is the Jacobian matrix of $\boldsymbol{F}$ with respect to $\boldsymbol{u}$ and α_i, a_i, b_j and c_j are constants. $\boldsymbol{k}_i$ is the unknown vectors to be solved for.

Since 5 equations are defined at any grid point, the matrix $\boldsymbol{A}$ has 5 by 5 dense blocks if the variables at each grid point are listed consecutively. This information can be very useful in building block version of ILU preconditioners. However, careful examination reveals that not all five variables appear in all five equations, which means each of the 5 by 5 dense block contains entries that are numerically zero. Thus if the matrix is stored in a standard sparse matrix format, such as the compressed sparse row format [48], these dense 5 by 5 blocks will be difficult to detect and to exploit in a block ILU factorization. Strategies that can detect and exploit block structure in a general sparse matrix are obviously useful in such applications for effective and intelligent preconditioning. Note that extracting useful information from a sparse matrix does not destroy the generality of the ILU preconditioner. On the contrary, it provides problem specific information to the preconditioning techniques so that a better preconditioner can be constructed.

2.2 Matrix Features Extracted

The features of a matrix are directly related to the precision of the prediction system. As illustrated by the previous examples, some matrix features are closely related to the performance of preconditioned iterative solvers. We will compute the features of a matrix first and then use such information to make prediction. The features of a matrix are a reflection of its sparsity and the locations and the size of the nonzero elements. We have extracted more than 60 features such as the matrix structure, value, bandwidth and diagonal related statistics. There may be other useful features that we can extract in the future and add them to the feature space. Part of the matrix features are calculated using SPARSKIT [51].

Figure 2.1 is an illustration of the sparsity pattern structures of two different sparse matrices from the MatrixMarket [38]. Figure 2.1(a) shows the sparsity pattern of the matrix NOS7, which has 729 columns and 2673 nonzero elements. The matrix NOS7 is from finite difference approximation to diffusion equation with varying diffusivity in a 3D unit cube with Dirichlet boundary conditions. Figure 2.1(b) depicts the sparsity pattern of the matrix NNC261, which has 261 columns and 1500 nonzero elements. The matrix NNC261 is from a nuclear reactor model. It can be seen that the two matrices have quite different sparsity patterns. Our aim is to extract features that can represent these and other properties of the sparse matrices and to use data mining techniques to predict the solving status of the sparse matrices by the preconditioned iterative solvers, based on their sparsity pattern and properties (features).

2.2.1 Structure

This group of features describe the distribution of nonzero entries of a sparse matrix.

- $nnzrt$ - sparsity rate (the number of nonzero elements divided by the number of all elements) of the whole matrix.
- $lowfillrt$ - sparsity rate (the number of nonzero elements divided by the number of all elements in the lower triangular part) of the lower triangular part.
- $upfillrt$ - sparsity rate (the number of nonzero elements divided by the number of all elements in the upper triangular part) of the upper triangular part.
- $diagfillrt$ - sparsity rate (the number of nonzero elements divided by the number of all elements in the main diagonal) of the main diagonal.
- $avnnzprow$ - the average nonzero entries per row.
- $sdavnnzprow$ - the standard deviation of nonzero entries per row.

- *avnnzpcol* - the average nonzero entries per column.
- *sdavnnzpcol* - the standard deviation of nonzero entries per column.
- *maxnnzprow* - the maximum number of nonzero elements per row.
- *minnnzprow* - the minimum number of nonzero elements per row.
- *maxnnzpcol* - the maximum number of nonzero elements per column.
- *minnnzpcol* - the minimum number of nonzero elements per column.
- *nzdiags* - the total number of non-void diagonals, i.e., the number of diagonals which have at least one nonzero element among the $2n - 1$ diagonals of the matrix.
- *symmc* - it measures whether a matrix is symmetric, i.e., $A = A^T$.
- *relsymm* - it describes the relative symmetry rate of a matrix. It is the ratio of the number of elements that match divided by nnz. An element $a(i, j)$ in the matrix A matches if it satisfies the following condition: if $a(i, j)$ is nonzero then $a(j, i)$ is nonzero.
- *normal* - if a matrix is a normal matrix, *normal* is equal to 1, otherwise it is 0. A matrix A is said to be normal if it commutes with its transpose conjugate, i.e., if it satisfies the relation $A^H A = AA^H$.
- *blocksize* - it reflects whether a matrix has a block structure or not. The matrix has a block structure if it consists of square blocks that are dense. The value of *blocksize* greater than one represents the size of the largest block.

2.2.2 Value

The attributes in this group sum up the value distribution of a matrix.

- *onenorm* - the one norm of a matrix, which equals to the maximum of the sum of the columns.
- $infnorm$ - the infinity norm of a matrix, which equals to the maximum of the sum of the rows.
- $frnorm$ - the Frobenius norm of a matrix, which is defined as $(\sum_{j=1}^{m}\sum_{i=1}^{n}|a_{ij}|^2)^{1/2}$.
- *minonenorm* - the minimum of the sum of the columns.
- $mininfnorm$ - the minimum of the sum of the rows.
- $symfnorm$ - Frobenius norm of the symmetric part of a matrix.
- $nsymfnorm$ - Frobenius norm of the unsymmetric part of a matrix.
- *avnnzval* - the average of the absolute value of all nonzero entries in a matrix.
- *sdavnnzval* - the standard deviation of the absolute value of all nonzero entries in a matrix.
- *avdiag* - the average of the absolute value of all nonzero entries in the main diagonal.
- *sdavdiag* - the standard deviation of the absolute value of all nonzero entries in the main diagonal.
- *avuptrig* - the average of the absolute value of all nonzero entries in the upper triangular part.

- *sdavuptrig* - the standard deviation of the absolute value of all nonzero entries in the upper triangular part.

- *avlowtrig* - the average of the absolute value of all nonzero entries in the lower triangular part.

- *sdavlowtrig* - the standard deviation of the absolute value of all nonzero entries in the lower triangular part.

- *pup* - ratio of the sum of the absolute value of entries in upper triangular part to the sum of entries of the whole matrix.

- *plow* - ratio of the sum of the absolute value of entries in lower triangular part to the sum of entries of the whole matrix.

- *pdiag* - ratio of the sum of the absolute value of entries in main diagonal to the sum of entries of the whole matrix.

2.2.3 Bandwidth

This group of features describe the bandwidth of a matrix. The bandwidth provides a measure of the clustering of nonzero entries about the main diagonal.

- *lowband* - lower bandwidth of a matrix. It is defined as the largest value of $i - j$, where $a(i, j)$ is nonzero.

- *upband* - upper bandwidth of a matrix. It is defined as the largest value of $j - i$, where $a(i, j)$ is nonzero.

- *maxband* - maximum bandwidth of a matrix. It is defined as $\max(\max(j) - \min(j))$, where $a(i, j)$ is nonzero.

- *avband* - average bandwidth of a matrix. It is defined as the average width of all rows: $avg(\max(j) - \min(j))$, where $a(i, j)$ is nonzero.

2.2.4 Diagonal

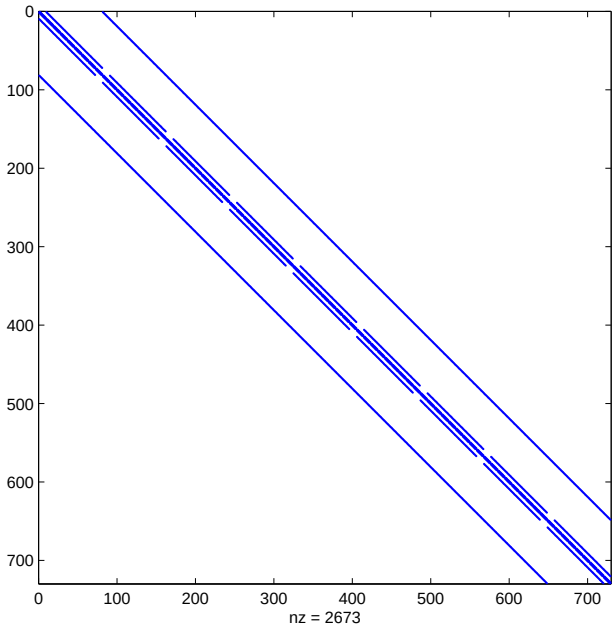
The features in this group are diagonal related.

- *avdisfd* - the average distance from each entry to the diagonal.
- *sdavdisfd* - the standard deviation of distance from each entry to the diagonal.
- *avvalfd* - the average of the value difference from each of the entry to its diagonal value.
- *sdavvalfd* - the standard deviation of the value difference from each of the entry to its diagonal value.
- *avmaxvalfd* - the average of the difference from the largest value in a row to the diagonal value.
- *sdavmaxvalfd* - the standard deviation of the difference from the largest value in a row to the diagonal value.
- *diagdomrow* - the percentage of weakly diagonally dominant rows. Row i is weakly diagonally dominant if $|a_{ii}| \geq \sum_{j=1, j \neq i}^{j=n} |a_{ij}|$.
- *diagdomcol* - the percentage of weakly diagonally dominant columns. Column j is weakly diagonally dominant if $|a_{jj}| \geq \sum_{i=1, i \neq j}^{i=n} |a_{ij}|$.
- *diagvalrate* - the ratio of the minimum absolute diagonal element value (except zero) to the maximum absolute diagonal element value.

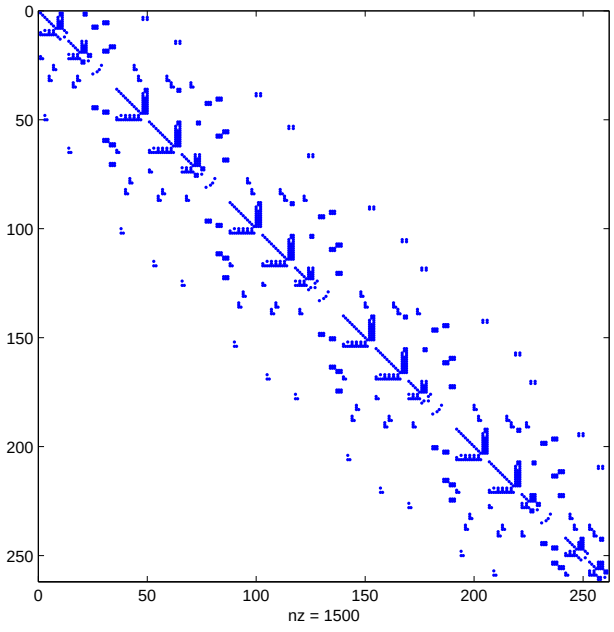
2.2.5 Others

We also include some other features which are listed below:

- *strzpiv* - the number of structural zero pivots (a structural zero pivot is a null column above or null row to the left of a zero diagonal element).
- *zpivrow* - whether a matrix has a null row to the left of a zero diagonal element.
- *zpivcol* - whether a matrix has a null column above a zero diagonal element.
- *szvdiag* - the smallest nonzero diagonal element with the dot product of its left vector and up vector being zero.
- *minvalcol* - the minimum of the smallest nonzero value in each column with a zero diagonal element.
- *minpivrate* - if a diagonal element has nonzero value, find the smallest nonzero value in that column and divide it by the diagonal element. Find the minimum of such values among all columns.



(a) Matrix NOS7



(b) Matrix NNC261

Figure 2.1: Illustration of different sparsity patterns of two different sparse matrices.

Chapter 3

Solving Status of the Sparse Linear Systems

In this chapter, we analyze the relationship among the the solving status of the sparse linear systems with respect to different preconditioners, the choice of preconditioned solvers with different parameters, and the attributes of the matrices [56, 57]. We show some of the interesting experimental results obtained from solving the 319 sparse linear systems with different preconditioned iterative solvers. Here we say a sparse linear system is solved if the preconditioner can be successfully constructed with a moderate condest value (condest is defined as $\|(LU)^{-1}e\|_\infty$, where e is the vector of all ones), the preconditioned solver converges within a preset number of iteration times and the relative residual norm is smaller than a preset limit.

The structure of this chapter is as follows: We analyze the solving status of the 319 sparse linear systems with one kind of preconditioners in each of the first 6 sections. In the last section, we show some comprehensive results for all the preconditioners.

3.1 ILU0

Most general purpose preconditioners are derived from incomplete (lower-upper) LU (ILU) factorization of the coefficient matrix A. It has been noted [13, 48] that stan-

dard ILU preconditioner, ILU0, is not robust for solving difficult problems from realistic applications. Our experimental results show that, of the 319 matrices, the ILU0 preconditioner can be successfully constructed for 185 matrices, and among them, 139 matrices are successfully solved.

When solving the 319 sparse linear systems generated from the 319 sparse matrices, there are 4 kinds of returning status. If $solvstat = -100$, it means that the program fails because zero pivot is encountered in constructing the preconditioner. If $solvstat = -7$, it means the condest value is very large (larger than 10^{10}) and the preconditioner is unstable. The condest value measures the stability of the triangular solvers [10]. When $solvstat = -1$, it means that the preconditioned solver does not converge when the number of iterations is greater than the preset limit (the limit is set to be 500). If $solvstat = 0$, the problem is solved successfully with the ILU0 preconditioner. We divide the 319 matrices into 4 categories according to the returning status of solving the corresponding sparse linear systems. The number of matrices in each category is shown in Table 3.1.

Category	Number of Matrices	solvstat
1	132	-100
2	18	-7
3	30	-1
4	139	0

Table 3.1: Number of matrices in each category with respect to ILU0.

3.2 MILU0

MILU0 preconditioner is ILU0 with the diagonal compensation. It also suffers from the same zero pivot problem as ILU0 does. Generally speaking, if the main diagonal of the matrix has small magnitude value or low density, MILU0 will not work

well. However, it is better than ILU0 for certain problems, thanks to the diagonal compensation technique. Of the 319 sparse linear systems, the preconditioner can be successfully constructed for 190 of them, and 164 of them are successfully solved (see Table 3.2).

Category	Number of Matrices	solvstat
1	129	-100
2	24	-7
3	2	-1
4	164	0

Table 3.2: Number of matrices in each category with respect to MILU0.

Compared with ILU0, we can see that the number of matrices in Category 3 is greatly reduced (from 30 to 2). That means that the number of matrices for which MILU0 cannot converge decreases. The number of matrices in Categories 1 and 2 does not change much. Thus the total number of matrices that can be successfully solved increases about 18%.

3.3 ILUD

The preconditioner ILUD computes the ILU factorization with only the standard threshold dropping. In ILUD, the parameter α indicates whether or not the diagonal compensation is used. If $\alpha = 0$, it means that there is no diagonal compensation, and only the standard threshold dropping is used. If $\alpha = 1$, it means that the diagonal compensation is applied in addition to the standard threshold dropping. We test ILUD under these two cases separately. In each case we test the matrices with three different dropping tolerance values, which are 0.0001, 0.001 and 0.01 respectively.

3.3.1 $\alpha = 1$

First, we analyze the relationship among the tolerance value, the solving status of the sparse linear systems, and the attributes of the matrices. With the decrease of the tolerance value, more sparse linear systems are solved. As the decrease of the tolerance value allows more fill-in elements, the performance of ILUD begins to improve. For example, when $tol = 0.01$, 197 linear systems are solved. When tol decreases to 0.001, 53 more linear systems that cannot be solved when tol is 0.01 are now solved. When tol decreases to 0.0001, 19 more are solved. Table 3.3 shows these solving results.

Class	Number of Matrices	solvability
1	197	solved when $tol = 0.01$
2	197 + 53	solved when $tol = 0.001$
3	197 + 53 + 19	solved when $tol = 0.0001$

Table 3.3: Number of matrices in each class in testing ILUD with $\alpha = 1$.

We analyze the average value of attributes in the three classes in Table 3.4. It reveals that there are some trends in the attributes of the matrices in the three classes. For example, when tol becomes smaller, more matrices with smaller values of AVNNZPCOL, AVBND, AVDIAG and NZDIAGS are solved. Table 3.4 shows the average value of AVNNZPCOL is 0.2604 in Class 1, which decreases to 0.0231 in Class 2 and further decreases to 0.0053 in Class 3.

From Table 3.3, we know when $tol = 0.0001$ ILUD works the best. The results in Table 3.5 are obtained with $tol = 0.0001$. In this case, of the 319 sparse linear systems, 265 of them were successfully solved with ILUD. When solving the 319 problems, there are also four kinds of returning status, the same as with ILU0. The only difference is when $solvstat = -100$, the construction of the preconditioner fails because a zero row is encountered, instead of a zero pivot in ILU0. The number of

Attributes	Class 1	Class 2	Class 3
AVNNZPCOL	0.2604	0.0231	0.0053
NNZLOW	0.5098	0.4759	0.4717
NNZUP	0.4966	0.4816	0.4460
NNZDIAG	0.1936	0.0396	0.0809
LOWBAND	0.2972	0.1505	0.1427
UPBAND	0.3002	0.1566	0.1429
MAXBAND	0.3626	0.2412	0.2161
AVBAND	0.1946	0.1222	0.0893
AVDISFD	0.0812	0.0522	0.0493
SDAVDISFD	0.0864	0.0491	0.0520
AVDIAG	0.0202	0.0113	0.0023
SDAVDIAG	0.0218	0.0089	0.0016
AVVALFD	1.77E+10	5.57E+6	3.83E+5
SDAVVALFD	1.80E+10	4.35E+6	3.67E+5
AVMAXVALFD	2.98E+10	9.40E+7	7.94E+5
SDAVMAXVALFD	1.95E+10	4.68E+6	4.20E+5
RELSYMM	0.9134	0.9187	0.9274
NZDIAGS	0.1393	0.0963	0.0449
DIAGDOMCOL	0.4591	0.0942	0.1201
DIAGDOMROW	0.4337	0.0753	0.1111

Table 3.4: Average attributes of the sparse linear systems solved by ILUD with $\alpha = 1$.

matrices in each category is shown in Table 3.5.

Category	Number of Matrices	solvstat
1	5	-100
2	45	-7
3	4	-1
4	265	0

Table 3.5: Number of matrices in each category with respect to ILUD with $\alpha = 1$.

3.3.2 $\alpha = 0$

Now let us look at the some of the experimental results of ILUD with $\alpha = 0$, which means only the standard threshold dropping is applied and no diagonal compensation

is used. Following the sequence of analyzing ILUD with $\alpha = 1$, We will also analyze the relationship among the tolerance value, the solving status of the sparse linear systems and the attributes of the matrices at first.

Table 3.6 shows the similar three classes of matrices as in Table 3.3. With the decrease of the tolerance value, the number of linear systems that can be successfully solved increases, too. As no diagonal compensation is used, the number of matrices in each class is much less compared with using diagonal compensation. For example, there are 197 matrices in Class 1 when $\alpha = 1$, but there are only 161 matrices in Class 1 when $\alpha = 0$.

Class	Number of Matrices	solvability
1	161	solved when $tol = 0.01$
2	161 + 44	solved when $tol = 0.001$
3	161 + 44 + 24	solved when $tol = 0.0001$

Table 3.6: Number of matrices in each class in testing ILUD with $\alpha = 0$.

Table 3.7 shows the average value of attributes in the three classes. We can observe that some attributes have similar trends as in Table 3.4. For example, when *tol* becomes smaller, more sparse linear systems with smaller values of AVNNZPCOL, AVBAND, AVDIAG and NZDIAGS are solved.

Figures 3.1 - 3.4 compare the trends of the average values of some attributes in each class with $\alpha = 1$ and $\alpha = 0$. In these figures ILUD means ILUD with $\alpha = 1$, while ILUDA means ILUD with $\alpha = 0$. It can be easily observed from these figures that the attributes AVNNZPCOL, AVBAND, AVDIAG and NZDIAGS have very similar trends in the dropping of the average values with the decrease of the tolerance parameter *tol*.

Like the way we analyze the case with $\alpha = 1$, Table 3.8 is obtained by setting the tolerance value to be 0.0001. In this case, 224 out of the 319 sparse linear systems

Attributes	Class1	Class2	Class3
AVNNZPCOL	0.2773	0.0293	0.0116
NNZLOW	0.5172	0.4750	0.4771
NNZUP	0.5083	0.4694	0.4661
NNZDIAG	0.2206	0.0512	0.0556
LOWBAND	0.3215	0.1620	0.1939
UPBAND	0.3258	0.1579	0.1939
MAXBAND	0.3867	0.2534	0.2734
AVBAND	0.2123	0.1317	0.1299
AVDISFD	0.0914	0.0523	0.0528
SDAVDISFD	0.0963	0.0457	0.0584
AVDIAG	0.0224	0.0146	0.0067
SDAVDIAG	0.0237	0.0121	0.0062
AVVALFD	2.16E+10	2.29E+7	4.01E+7
SDAVVALFD	2.20E+10	2.20E+7	3.46E+7
AVMAXVALFD	3.65E+10	3.98E+7	7.19E+7
SDAVMAXVALFD	2.39E+10	2.50E+7	3.71E+7
RELSYMM	0.9086	0.9258	0.9058
NZDIAGS	0.1501	0.1037	0.0726
DIAGDOMCOL	0.5331	0.1351	0.1071
DIAGDOMROW	0.4945	0.1034	0.1228

Table 3.7: Average attributes of the sparse linear systems solved by ILUD with $\alpha = 0$.

were successfully solved with ILUD with $\alpha = 0$. Unlike the case with $\alpha = 1$, there are 5 categories of returning status this time. Four of them are the same as with $\alpha = 1$, the only difference is $solvstat = -3$, which means that there is an unanticipated break-down or divide by zero error when running GMRES. The number of matrices in each category is shown in Table 3.8. Compared with Table 3.5, 224 matrices can be successfully solved with $\alpha = 0$, while 265 can be successfully solved with $\alpha = 1$. Without diagonal compensation, 7 more matrices have the problem of large condest value, 16 more have the problem of unanticipated break-down, and 32 more can not be solved in 500 iterations.

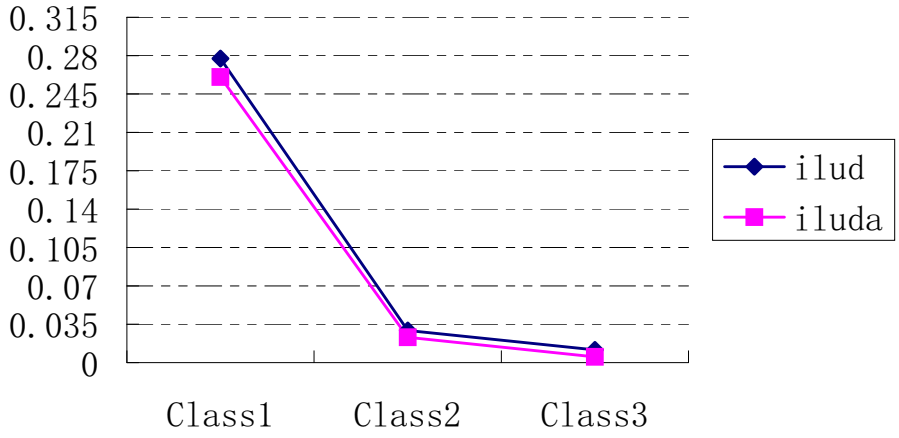


Figure 3.1: The trend of the values of AVNNZPCOL in ILUD and ILUDA.

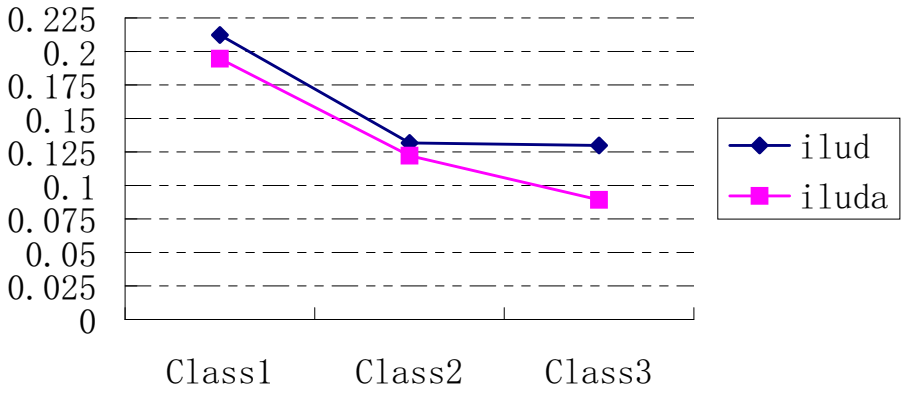


Figure 3.2: The trend of the values of AVBAND in ILUD and ILUDA.

3.4 ILUDP

ILUDP is ILUD with column pivoting. As with ILUD, we test ILUDP with or without diagonal compensation separately. We use the same set of tolerance values, which are 0.0001, 0.001 and 0.01.

3.4.1 $\alpha = 1$

We first test ILUDP with the diagonal compensation. Table 3.9 illustrates the relationship between the tolerance value and the solving status of the sparse linear systems. With the decreasing of tolerance value, more sparse linear systems are

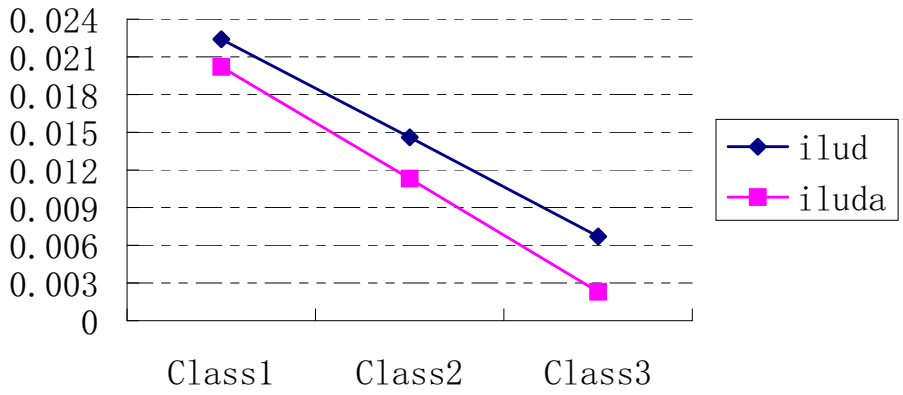


Figure 3.3: The trend of the values of AVDIAG in ILUD and ILUDA.

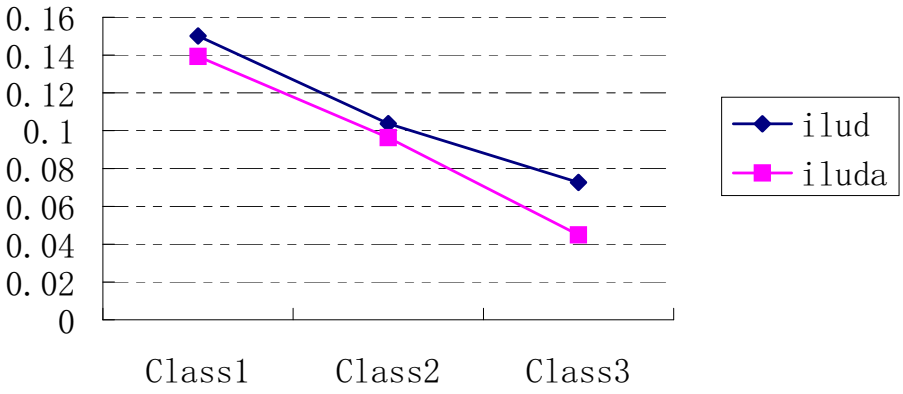


Figure 3.4: The trend of the values of NZDIAGS in ILUD and ILUDA.

solved. Compared with Table 3.3, a larger number of matrices appears in each class. For example, when $tol = 0.01$, only 197 sparse linear systems are successfully solved. After applying column pivoting, 62 more sparse linear systems are solved.

Table 3.10 shows the average value of attributes in each of the three classes. Unlike ILUD, there is no obvious trend in the attributes of the matrices with the dropping of the tolerance value tol.

From Table 3.9, we know when $tol = 0.0001$ ILUDP works the best. The results in Table 3.11 are obtained with $tol = 0.0001$. When solving the 319 sparse linear systems, there are only three different kinds of returning status, all of which have appeared with ILUD. The difference is that for ILUDP, there is no category with

Category Number	Number of Matrices	solvstat
1	5	-100
2	38	-7
3	16	-3
4	36	-1
5	224	0

Table 3.8: Number of matrices in each category with respect to ILUD with $\alpha = 0$.

Class	Number of solved Matrices	solvability
1	259	solved when $tol = 0.01$
2	259 + 23	solved when $tol = 0.001$
3	259 + 23 + 6	solved when $tol = 0.0001$

Table 3.9: Number of matrices in each class in testing ILUDP with $\alpha = 1$.

$solvstat = -1$, which means that the construction of the preconditioner fails because the maximum number of iterations is reached. The number of matrices in each category is shown in Table 3.11. Compared with Table 3.5, not only there is no category with $solvstat = -1$, the number of matrices with $solvstat = -7$ also drops. Once again the results show that using column pivoting can improve the solvability of ILUD with $\alpha = 1$.

3.4.2 $\alpha = 0$

Now let us look at the some of the experimental results of ILUDP with $\alpha = 0$, which means no diagonal compensation is used. Following the sequence of analyzing ILUDP with $\alpha = 1$, We also analyze the relationship among the tolerance value, the solving status of the sparse linear systems and the attributes of the matrices at first.

As we may expect, for ILUDP with $\alpha = 0$, with decrease of the tolerance value tol, more sparse linear systems are solved. The number of solved linear systems in each class in Table 3.12 is smaller than in Table 3.9, as no diagonal compensation is used here. However, the solving ability of ILUDP with $\alpha = 0$ is much better than that of

Attributes	Class1	Class2	Class3
AVNNZPCOL	0.0252	0.0234	0.0234
NNZLOW	0.4925	0.4921	0.4920
NNZUP	0.5035	0.5015	0.4997
NNZDIAG	0.1559	0.1458	0.1445
LOWBAND	0.3337	0.3103	0.3110
UPBAND	0.3405	0.3163	0.3160
MAXBAND	0.4112	0.3850	0.3858
AVBAND	0.1997	0.1855	0.1859
AVDISFD	0.1031	0.0951	0.0949
SDAVDISFD	0.0972	0.0901	0.0901
AVDIAG	0.0172	0.0161	0.0160
SDAVDIAG	0.0179	0.0166	0.0166
AVVALFD	1.34E+10	1.23E+10	1.21E+10
SDAVVALFD	1.36E+10	1.25E+10	1.23E+10
AVMAXVALFD	2.27E+10	2.08E+10	2.04E+10
SDAVMAXVALFD	1.48E+10	1.36E+10	1.33E+10
RELSYMM	0.8321	0.8421	0.8398
NZDIAGS	0.1784	0.1678	0.1673
DIAGDOMCOL	0.3660	0.3402	0.3365
DIAGDOMROW	0.3408	0.3149	0.3142

Table 3.10: Average attributes of the sparse linear systems solved by ILUDP with $\alpha = 1$.

ILUD with $\alpha = 0$ (see Table 3.6). For example, after applying column pivoting, 73 more sparse linear systems are successfully solved when $tol = 0.01$.

Like the situation with ILUDP with $\alpha = 1$, there is no obvious trend in the average of the matrix attributes with respect to the dropping of tol so we just omit the table here.

Like the way we analyze the case with $\alpha = 1$, Table 3.13 is obtained by setting the tolerance value to be 0.0001. When solving the 319 sparse linear systems, we observe 4 kinds of returning results. All of them are the same as in ILUD. The number of matrices in each category is shown in Table 3.13. Compared with Table 3.11, it shows that without using diagonal compensation, the number of matrices that

Category	Number of Matrices	solvstat
1	5	-100
2	26	-7
3	288	0

Table 3.11: Number of matrices in each category with respect to ILUDP with $\alpha = 1$.

Class	Number of solved matrices	solvability
1	234	solved when $tol = 0.01$
2	234 + 29	solved when $tol = 0.001$
3	234 + 29 + 9	solved when $tol = 0.0001$

Table 3.12: Number of matrices in each class in testing ILUDP with $\alpha = 0$.

fall in Category 2 and Category 3 increases. However, compared with Table 3.8, because of the adoption of column pivoting, the number of sparse linear systems with $solvstat = -3$ and $solvstat = -1$ decreases, thus the total number of sparse linear systems successfully solved increases by 48.

Category	Number of Matrices	solvstat
1	5	-100
2	34	-7
3	8	-1
4	272	0

Table 3.13: Number of matrices in each category with respect to ILUDP with $\alpha = 0$.

3.5 ILUT

ILUT is incomplete LU factorization with double dropping strategies. There are two important parameters for ILUT. One is drop tolerance tol, the other is the number of fill-in elements $lfil$. The smaller the tolerance value, the more elements will remain in the preconditioner, thus the better the result. We can see from Figure 3.5 that with the increase of tol, the number of sparse linear systems solved decreases. We

can think of *tol* as the parameter that helps reduce computational cost.

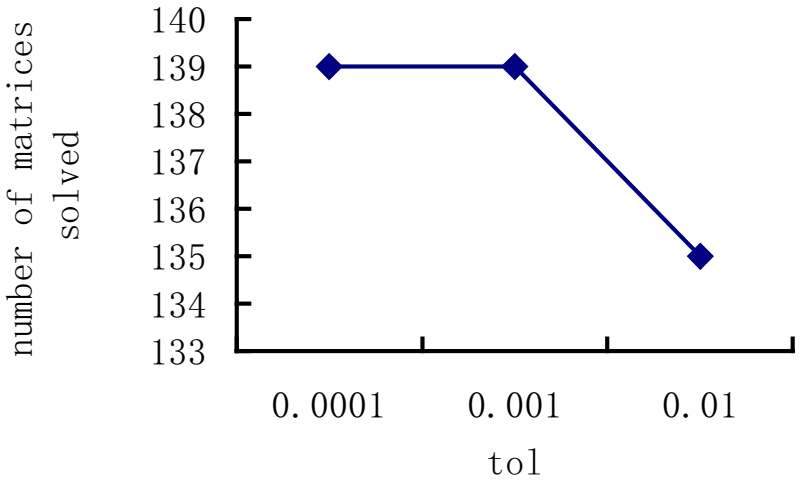


Figure 3.5: The number of sparse linear systems solved by ILUT with different *tol* values.

To use the ILUT algorithm 1.2.4, the number of fill-in elements $lfil$ has to be set for each matrix. As the matrices vary in size and sparsity, it is difficult to set a specific value for $lfil$ for all the matrices. Thus we use another parameter $filr$, which denotes the fill-in rate. $lfil$ can be obtained by multiplying $filr$ with the average number of non-zero entries of all the rows. Now $lfil$ may get different values for different matrices. Figure 3.6 shows the number of sparse linear systems solved with different $filr$ values. Here we choose $filr$ to increase from 0.8 to 1.5. When the fill-in rate grows, the number of the sparse linear systems solved increases steadily.

The results in Table 3.14 are obtained by setting *tol* to be 0.0001 and $filr$ to be 1.5. When solving the 319 sparse linear systems, we got 5 kinds of returning status, the same categories as we obtained with ILUD with $\alpha = 0$. The number of matrices in each category is shown in Table 3.14.

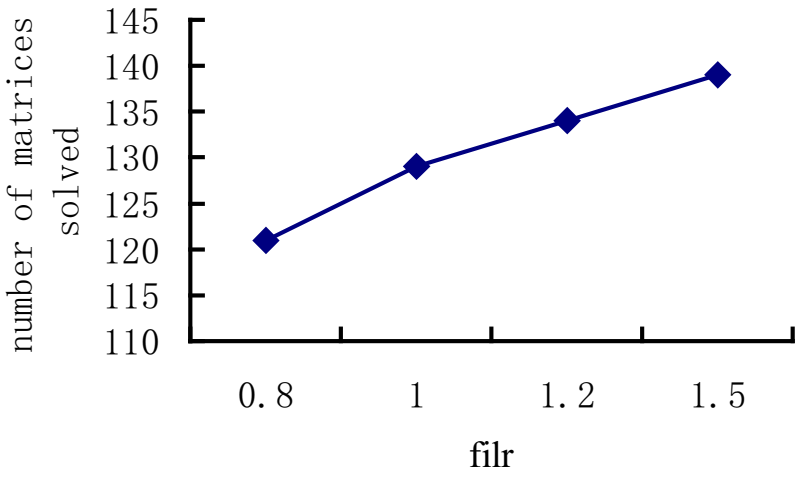


Figure 3.6: The number of sparse linear systems solved by ILUT with different $filr$ value.

Category Number	Number of Matrices	solvstat
1	5	-100
2	118	-7
3	11	-3
4	46	-1
5	139	0

Table 3.14: Number of matrices in each category with respect to ILUT.

3.6 ILUTP

ILUTP is ILUT with column pivoting. Like ILUT, there are also two important parameters for ILUTP, drop tolerance tol, and the number of fill-in elements $lfil$. Again like ILUT, with the increase of tol, the number of sparse linear systems solved decreases (see Figure 3.7).

Based on the same reason mentioned in the previous section, we use the fill-in rate $filr$ instead of $lfil$ in Figure 3.8. With the increase of $filr$, the number of sparse linear systems solved grows steadily as in Figure 3.6.

From Figure 3.7 and Figure 3.8, we know that when tol is 0.0001 and $filr$ is 1.5,

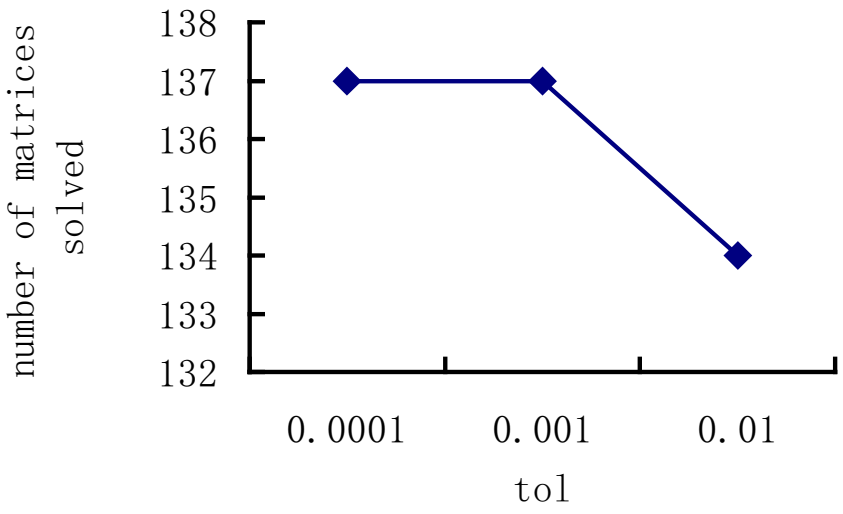


Figure 3.7: The number of sparse linear systems solved by ILUTP with different *tol* value.

ILUTP achieves the best result. So we set the *tol* to be 0.0001 and $filr$ to be 1.5 to partition the matrices in categories in Table 3.15. When solving the 319 sparse linear systems, we got the same 5 kinds of returning status as with ILUT. The number of matrices in each category is shown in Table 3.15. Compared with Table 3.15, we can find out that using column pivoting, there are less sparse linear systems suffering from unanticipated break-down and failing to convergence, but more linear systems suffering from large condest value. Thus the total number of successfully solved linear systems almost remains the same as without column pivoting.

Category Number	Number of Matrices	solvstat
1	5	-100
2	150	-7
3	6	-3
4	21	-1
5	137	0

Table 3.15: Number of matrices in each category with respect to ILUTP.

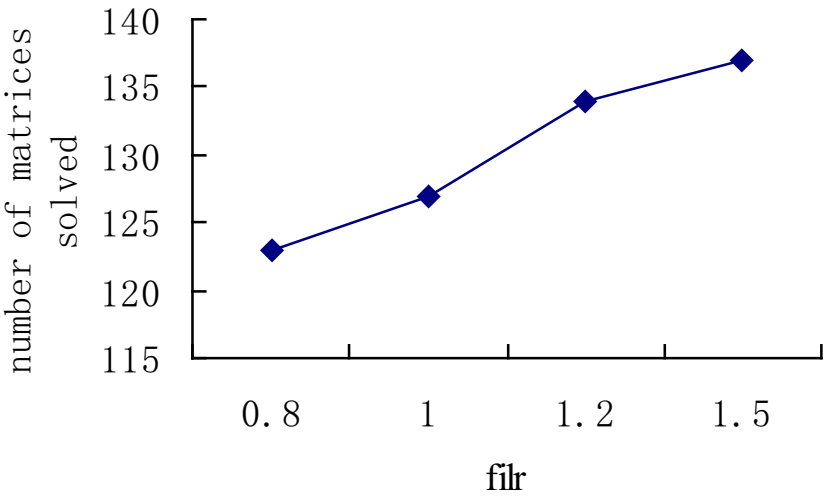


Figure 3.8: The number of sparse linear systems solved by ILUTP with different $filr$ value.

3.7 Comprehensive Results

We sum up the attributes of the matrices that can be successfully solved by the preconditioned iterative solvers in Table 3.16. Here ILUD means ILUD with $\alpha = 1$, while ILUDA means ILUD with $\alpha = 0$. Similarly, ILUDP means ILUDP with $\alpha = 1$, while ILUDPA means ILUDP with $\alpha = 0$. $\Uparrow$ means that the average value of this category is the highest among all the categories. $\Downarrow$ means that the average value of this category is the lowest among all the categories. $\uparrow$a means that the average value of this category is not the highest among all the categories, but higher than the total average. On the contrary, $\downarrow$a means that the average value of this category is lower than the total average. So the category of the problems can be successfully solved has the highest values of NNZDIAG, AVDIAG, AVVALFD and AVMAXVALFD compared with the average values with other returning statuses. And the values of DIAGDOMCOL and DIAGDOMROW are generally high, at least higher than the average. Also, the RELSYMM values are higher than the average

and the NZDIAGS values are lower than the average.

Preconditioners	ILU0	MILU0	ILUD	ILUDA	ILUDP	ILUDPA	ILUT	ILUTP
NNZDIAG	⇑	⇑	⇑	⇑	⇑	⇑	⇑	⇑
AVDIAG	⇑	⇑	⇑	⇑	⇑	⇑	⇑	↑a
AVVALFD	⇑	⇑	⇑	⇑	⇑	⇑	⇑	⇑
AVMAXVALFD	⇑	⇑	⇑	⇑	⇑	⇑	⇑	⇑
AVBAND	⇑	⇑	↓a	↓a	↓a	↑a	↑a	↑a
DIAGDOMCOL	⇑	⇑	↑a	⇑	↑a	↑a	⇑	⇑
DIAGDOMROW	⇑	⇑	↑a	↑a	↑a	↑a	⇑	⇑
RELSYMM	↑a	↑a	⇑	↑a	⇑	↑a	↑a	↑a
NZDIAGS	↓a	↓a	⇓	↓a	⇓	↓a	↓a	↓a

Table 3.16: Attributes of the matrices that can be successfully solved.

According to our experimental results, we verify that most of the diagonal dominant and symmetric matrices can be easily solved. We also find out that if a matrix satisfies the condition that ncol=nnzdiag, then it probably can be solved by some preconditioned solver, too.

3.7.1 Symmetric Matrices

All symmetric matrices can be solved using at least one of the preconditioners except two matrices: BCSSTM13 and ZENIOS. They cannot be solved by any preconditioned iterative solvers, using the 6 preconditioners under our consideration. The reasons why they failed are listed in Table 3.17 and Table 3.18.

Preconditioner	result	explanation
ILU0	precresult = 4	zero pivoting encountered at step 4
MILU0	precresult = 4	zero pivoting encountered at step 4
ILUD	precresult = -3	zero row encountered
ILUDP	precresult = -3	zero row encountered
ILUT	precresult = -5	zero row encountered
ILUTP	precresult = -5	zero row encountered

Table 3.17: Solving result of BCSSTM13 using different preconditioners.

Preconditioner	result	explanation
ILU0	precresult = 1	zero pivoting encountered at step 1
MILU0	precresult = 1	zero pivoting encountered at step 1
ILUD	precresult = -3	zero row encountered
ILUDP	precresult = -3	zero row encountered
ILUT	precresult = -5	zero row encountered
ILUTP	precresult = -5	zero row encountered

Table 3.18: Solving result of ZENIOS using different preconditioners.

3.7.2 Diagonally Dominant Matrices

Among the 319 matrices, there are all together 27 diagonally dominant matrices. Among them, SHERMAN3 is the only matrix that cannot be solved by any preconditioned iterative solvers. The reasons are the same for all the preconditioned solvers: the condest value is so large that the constructed preconditioners are unstable. FS-680-2 and FS-680-3 cannot be solved by ILUT and ILUTP, but can be solved by other preconditioned solvers. The reasons that they fail are also the same: the solvers cannot converge in the preset number of iterations.

3.7.3 The Case ncol=nnzdiag

We also find out that when ncol=nnzdiag, i.e., the entries in the main diagonal are all nonzero, most of the matrices can be successfully solved no matter what preconditioner is used. In the 319 matrices, there are 164 matrices with such a property. Table 3.19 shows the number of matrices solved by different preconditioned iterative solvers. We can see that 76% of matrices with this property are successfully solved by ILUT and ILUTP. If using ILUD and ILUDP, more than 95% of the matrices are solved.

Preconditioners	ILU0	MILU0	ILUD	ILUDP	ILUT	ILUTP
Matrices solved	128	144	157	160	125	125
Percentage	78%	88%	96%	98%	76%	76%

Table 3.19: Percentage of matrices with the property ncol=nnzdiag that can be solved.

3.7.4 Solving Status of the Sparse Linear Systems

Of the 319 the sparse linear systems, 300 of them can be solved by one or several preconditioned iterative solvers. We consider that a preconditioner is the best to a matrix if the preconditioned iterative solver runs successfully on that matrix with the smallest running time. If there is a tie in the running time, we list all the preconditioners that can be used to get that time. Our experiments show that all the preconditioners have the chance to be the best preconditioners, but their chances are quite different. Table 3.20 lists the preconditioners and for how many matrices they are the best. Here ILUD is the most favorable preconditioner as 65% of the matrices think it is the best choice.

Preconditioners	ILU0	MILU0	ILUD	ILUDP	ILUT	ILUTP
Best for how many matrices	39	91	208	161	62	60
Percentage of all the matrices	12%	29%	65%	50%	19%	19%

Table 3.20: Chances of each of the preconditioners to be the best.

Nineteen sparse linear systems cannot be solved by any preconditioned iterative solvers. Table 3.21 shows the returning status with different preconditioned solvers. We can see that almost all of these matrices either fail at the construction of the preconditioner (result $= -100$) or the condest value of the constructed preconditioner is too large (result $= -7$) to be stable. Here S_{restat} denotes the returning status of preconditioned solver using the preconditioner ILU0. The meaning of other column titles are similar to it. The returning status of constructing the preconditioners are

listed in Table 3.22. In this table $S_{precstat}$ denotes the return status of constructing preconditioner ILU0. The meaning of other column titles are similar. If the returning status is a positive number, it means zero pivot is encountered at that step. If it is equal to -3, it means the matrix U overflows the array. If it is equal to -5, it means zero row is encountered. If it is zero, it means the preconditioner is successfully constructed.

NAME	S_{restat}	M_{restat}	D_{restat}	DP_{restat}	T_{restat}	TP_{restat}
BCSSTM13	-100	-100	-100	-100	-100	-100
IMPCOLA	-100	-100	-7	-7	-7	-7
MBEACXC	-100	-100	-100	-100	-100	-100
MBEAFLW	-100	-100	-100	-100	-100	-100
MBEAUSE	-100	-100	-100	-100	-100	-100
MHD3200A	-100	-100	-7	-7	-3	-7
MHD416A	-100	-100	-7	-7	-7	-7
MHD4800A	-100	-100	-7	-7	-3	-7
NNC1374	-100	-100	-7	-7	-7	-7
RW5151	-100	-100	-7	-7	-7	-7
SHERMAN3	-7	-7	-7	-7	-7	-7
WEST0167	-100	-100	-7	-7	-7	-7
WEST0479	-100	-100	-7	-7	-7	-7
WEST0497	-100	-100	-7	-7	-7	-7
WEST0655	-100	-100	-7	-7	-7	-7
WEST0989	-100	-100	-7	-7	-7	-7
WEST1505	-100	-100	-7	-7	-7	-7
WEST2021	-100	-100	-7	-7	-7	-7
ZENIOS	-100	-100	-100	-100	-100	-100

Table 3.21: Return status of the unsolved matrices.

NAME	$S_{precstat}$	$M_{precstat}$	$D_{precstat}$	$DP_{precstat}$	$T_{precstat}$	$TP_{precstat}$
BCSSTM13	4	4	-3	-3	-5	-5
IMPCOLA	1	1	0	0	0	0
MBEACXC	1	1	-3	-3	-5	-5
MBEAFLW	1	1	-3	-3	-5	-5
MBEAUSE	1	1	-3	-3	-5	-5
MHD3200A	2	2	0	0	0	0
MHD416A	2	2	0	0	0	0
MHD4800A	2	2	0	0	0	0
NNC1374	9	9	0	0	0	0
RW5151	1	1	0	0	0	0
SHERMAN3	0	0	0	0	0	0
WEST0167	1	1	0	0	0	0
WEST0479	1	1	0	0	0	0
WEST0497	1	1	0	0	0	0
WEST0655	1	1	0	0	0	0
WEST0989	1	1	0	0	0	0
WEST1505	1	1	0	0	0	0
WEST2021	1	1	0	0	0	0
ZENIOS	1	1	-3	-3	-5	-5

Table 3.22: Return status of constructing the preconditioners for the unsolved matrices.

Chapter 4

Matrix Condition Number Prediction

Condition number of a matrix is an important feature which is closely related to the solvability of the preconditioned solvers. Condition number is also a widely used measure in numerical analysis and linear algebra. The general approach to obtaining it is through direct computation or estimation. The time and memory cost of such approaches are very high, especially for large size matrices. However, if we could find out the condition number of a matrix using some kind of "quick" method, it can benefit not only the IPRS, but also many mathematicians and engineers.

In this chapter, we propose a quite new approach to estimating the condition number of a sparse matrix [58]. That is, after computing the features of a matrix, we use support vector regression (SVR) technique to predict its condition number. We also use feature selection strategies to further reduce the response time and improve accuracy. We use a feature selection criterion which combines the weights from SVR and the weights from comparison of matrices with their preconditioned counterparts. Our experiments show that the response time of the prediction method is on average 15 times faster than the direct computation approaches, which makes it suitable for online condition number query. The accuracy of our prediction method is not as

precise as the general direct computation methods. Our experiments also show that using the predicted condition number will not improve the prediction accuracy in the IPRS. However, many application engineers are only interested in knowing whether a matrix is *well-conditioned* or *ill-conditioned* or the order of the condition number, not the exact value of the condition number. For such users, a rough prediction with quick response time is probably a better choice than a precise value after waiting for hours or days.

The structure of this chapter is as follows: Matrix condition number concept is introduced in Section 4.1. We briefly review SVM regression in Section 4.2. Three feature selection methods are described in Section 4.3. The computational experiments are carried out and the results are discussed in Section 4.4. We draw some conclusions in Section 4.5.

4.1 Matrix Condition Number

The condition number $k(A)$ of a nonsingular matrix A with respect to a matrix norm is formally defined as $||A|| \cdot ||A^{-1}||$ [20]. The condition number corresponding to the Frobenius norm will be denoted by $k_F(A)$ and the condition number corresponding to the p-norm will be denoted by $k_p(A)$. There are some relationships among the condition numbers based on different norms. For example, if $A \in R^{n \times n}$, then

$$\frac{1}{n} k_2(A) \leq k_1(A) \leq n k_2(A),$$

$$\frac{1}{n} k_\infty(A) \leq k_2(A) \leq n k_\infty(A),$$

$$\frac{1}{n^2} k_1(A) \leq k_\infty(A) \leq n^2 k_1(A).$$

In this chapter, we only stress on the condition number corresponding to the 1-norm $k_1(A)$. If $k(A)$ is relatively small, then the matrix A is called a *well-conditioned*

matrix, but if $k(A)$ is large, then A is an *ill-conditioned matrix* (e.g., around 10^5 for a 5×5 Hilbert matrix).

Condition number is a widely used matrix feature in many areas, such as in numerical analysis and linear algebra. In numerical analysis, the condition number is basically a measure of stability or sensitivity of a matrix (or the linear system it represents) to numerical operations. For example, the condition number associated with the linear equation $Ax = b$ gives a bound on how inaccurate the solution will be after the numerical solution. Suppose A is nonsingular, $\hat{x}$ is an approximate solution to x, r is the residual, and $b \neq 0$, then:

$$\frac{1}{k(A)} \frac{||r||}{||b||} \leq \frac{||x - \hat{x}||}{||x||} \leq k(A) \frac{||r||}{||b||}.$$

It means that the relative error in the computed solution is bounded by the condition number of the matrix A times the relative size of the residual. If A is ill-conditioned, the relative error may not be small even if the relative size of the residual is small. $k(A)$ can also measure how close A is to being singular:

$$\frac{1}{k(A)} = \min \frac{||A - B||}{||A||}, \ B \text{ is singular.}$$

A can be approximated by a singular matrix B if and only if $k(A)$ is large.

Condition number can also be used to predict the convergence of iterative methods. For example, for the conjugate gradient (CG) method, the error can be bounded in terms of $k_2(M^{-1}A)$ [2], where M is a preconditioner. If A is symmetric positive definite, then for CG with a symmetric positive definite preconditioner M, it can be shown that:

$$||\hat{x}^{(i)} - x||_A \leq 2\alpha ||\hat{x}^{(0)} - x||_A,$$

where $\alpha = (\sqrt{k_2(M^{-1}A)} - 1)/(\sqrt{k_2(M^{-1}A)} + 1)$ [22, 30].

There are several ways to obtain condition number. The direct method is to compute A^{-1} first and then multiply its norm with the norm of A. However, computing

A^{-1} is equivalent to solving a linear system which is very time and memory consuming. Another method is using a less expensive algorithm to estimate the condition number. There are some estimation algorithms in literature [25]. For example, LAPACK uses subroutine SGECON to compute the condition number. Its time cost is $O(n^2)$ extra beyond the $O(n^3)$ cost of solving $Ax = b$, where n is the dimension of A. If the size of the matrix is relatively large (e.g., $n > 20000$), the memory will be depleted before the computation is completed on our SunBlade 150 workstations. In most cases, the estimated condition number is within a factor of 10 of the true condition number, but there exist some counter-examples with large estimation errors. MATLAB uses LINPACK [16] for computing reciprocal of $k_1(A)$ and uses Higham's modification [25] of Hager's method to estimate $k_1(A)$. Such methods have similar problems with respect to memory and computing costs for large size matrices.

We propose a new approach to estimating the condition number of a *sparse* matrix. Instead of direct computation or estimation, we predict condition number from matrix features using data mining techniques. The predictor used is SVM regression (SVR) [24, 50, 53]. We also apply some feature selection methods [23, 41] to further reduce the time cost and improve precision. We propose a feature selection criterion which combines the weights from SVR with the weights from comparison of matrices with their preconditioned counterparts. Although the condition number predicted is not as precise as the above-mentioned direct computation methods, (in our experiments, if a relative deviation of 10^2 between the computed values and the predicated values of the condition number is acceptable, our prediction error is smaller than 25%), it has much smaller time and memory cost, especially for large size matrices, which is suitable for online condition number query. Furthermore, many users are only interested in knowing whether a matrix is *well-conditioned* or *ill-conditioned*, or the approximate order of the condition number. In these situations, response time and

reliability are at least as important as accuracy.

4.2 SVM Regression

SVM regression is an approach to predicting real-valued outputs. It has been successfully applied in many areas such as financial forecasting [52, 60], image recognition [35, 53] and signal processing [53]. SVM regression using the ε-insensitive loss function is called ε-SV regression [54]. In ε-SV regression, the goal is to find a function $f(x)$ that has at most ε deviation from the actually obtained targets y_i for all the training data, and at the same time is as flat as possible [50]. We can visualize this as a tube of size 2ε around $f(x)$ and data fall out of the tube are errors [3].

Suppose a linear function $f(x)$ is of the form:

$$f(x) =< w, x > +b, \quad w \in X, \quad b \in \Re,$$

where $< w, x >$ denotes the dot product of the vectors w and x. To account for the errors, we introduce slack variables ξ_i and ξ_i^*. ξ_i computes the error for underestimating the function while ξ_i^* computes the error for overestimating the function The ε-insensitive loss function $|\xi|_\varepsilon$ is expressed as:

$$|\xi|_\varepsilon := \begin{cases} 0, & \text{if } |\xi| \leq \varepsilon, \\ |\xi| - \varepsilon, & \text{otherwise.} \end{cases}$$

Then we have the following convex optimization problem:

$$\text{minimize } \frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{l} (\xi_i + \xi_i^*),$$

$$\text{subject to } \begin{cases} y_i - < w, x_i > -b \leq \varepsilon + \xi_i, \\ < w, x_i > +b - y_i \leq \varepsilon + \xi_i^*, \\ \xi_i,\ \xi_i^* \geq 0, \\ C > 0, \end{cases}$$

where C determines the trade-off between the flatness of $f(x)$ and the amount up to which deviation larger than ε is tolerated.

The above optimization problem can be solved more easily in its dual formulation [50]. We can use a standard dualization method utilizing Lagrange multipliers. After solving it, we can get:

$$w = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) x_i, \tag{4.1}$$

$$f(x) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) < x_i, x > + b. \tag{4.2}$$

The variable b can be computed as:

$$b = \begin{cases} y_i - < w, x_i > - \varepsilon, & \text{for } \alpha_i \in (0, C), \\ y_i - < w, x_i > + \varepsilon, & \text{for } \alpha_i^* \in (0, C). \end{cases}$$

For the nonlinear case, we apply a mapping $\Phi : X \to F$ to map input space into some feature space F. Here we use a kernel function, $K(x, x_i) =< \Phi(x), \Phi(x_i) >$, which is a symmetric function and satisfies the Mercer's condition. We substitute $K(x, x_i)$ for the dot product, which maps the input space into some reproduced kernel feature space. The commonly used kernel functions are:

$$\text{Polynomial} : K(x, x_i) = (< x, x_i > + c)^d,$$

$$\text{RBF} : K(x, x_i) = e^{-\frac{||x - x_i||^2}{2\sigma^2}},$$

$$\text{Neural Network} : K(x, x_i) = \tanh(\eta < x, x_i > + \vartheta).$$

4.3 Feature Selection

Our experiments show that the accuracy of the condition number predicted based on the matrix features described in Section 2.2 seems to be good. But such a collection of features may contain some redundant information. We apply feature selection methods to remove the redundancy. Feature selection may also bring other benefits: reduce the computation time, save memory space, remove noise and possibly optimize the prediction accuracy. For an online condition number prediction system, it is

crucial to lower the response time and improve precision. In this section we investigate 3 feature selection methods.

4.3.1 Correlation

Correlation is one of the simplest feature selection methods. It computes the correlation of the input vector x_i and the target vector y as follows:

$$Cor_i = \frac{\sum_{k=1}^{m}(x_{k,i} - \bar{x}_i)(y_k - \bar{y})}{\sqrt{\sum_{k=1}^{m}(x_{k,i} - \bar{x}_i)^2 \sum_{k=1}^{m}(y_k - \bar{y})^2}},$$

where the bar stands for an average over the index k. In linear regression, Cor_i^2 represents the fraction of the total variance around the mean value $\bar{y}$ that is explained by the linear relation between x_i and y. Correlation criteria can only detect linear dependencies between variables and target [23].

4.3.2 Weights from SVR

There have been some feature selection methods based on the weights from the SVM classification model [6, 28, 40, 44]. Using the weights from SV-regression works in the same way.

Using the kernel function $K(x, x_i)$, Equations (4.1) and (4.2) can be written as:

$$w = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)K(x_i),$$

$$f(x) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)K(x_i, x) + b.$$

Like in neural networks, the output prediction is of the form:

$$predict(x) = G(\sum_j w_j x_j + b),$$

where $G(x)$ is an activation function. A feature j with a larger weight w_j has more effect on the prediction than a feature with a smaller weight [40, 44]. Shih *et al.* jus-

tified in [49] that the features with higher weights are more influential in determining the width of the margin. Thus $||w||^2$ is a suitable criterion for feature selection.

4.3.3 Combinational Method

It is known that a good preconditioner can improve the condition number of a matrix [48]. Suppose M is a preconditioner of a matrix A. We compare the condition number as well as the matrix features of the original matrix A and the preconditioned matrix M to find out which features contribute more to the improvement of the condition number. Certain features have larger influence on the condition number and should be kept in feature selection. Assume we have l matrix examples, and k^A represents the condition number vector for all the original matrices, while k^M represents the condition number vector for all the preconditioned matrices. v_i^A is the vector of features for the ith original matrix, likewise, v_i^M is the vector of features for the ith preconditioned matrix. Then we can obtain the weight vector w_{cmp} to rank the features:

$$w_{cmp} = \sum_{i=1}^{l} \left(|k_i^A - k_i^M| * |v_i^A - v_i^M| \right). \tag{4.3}$$

w_{cmp} seems to be a reasonable criterion for feature selection. However, as we cannot successfully construct a useful preconditioner for all the general matrices, w_{cmp} is biased towards the features of the matrices that can be preconditioned. To remedy this problem, we propose to use the weight w_{comb}, which combines w_{cmp} and the weight from SVM regression w_{SVM}, as

$$w_{comb} = \mathrm{nml}(w_{cmp}) + \mathrm{nml}(w_{SVM}), \tag{4.4}$$

where the function $\mathrm{nml}(x)$ normalizes vector x. Thus w_{comb} is the sum of the normalized w_{cmp} and w_{SVM}.

4.4 Experiments and Results

In this section, we report our experiments on the accuracy and response time of the condition number prediction methods. We use SVM^{Light} [29] for SVM regression. There are 277 matrices from Matrix Market [38] tested in the experiments. We use altogether 60 matrix features, most of which are explained in Section 2.2. The experiments are carried out on a SunBlade 150 workstation.

4.4.1 Accuracy

First, we test how accurate the predicted condition numbers are, compared with the directly computed condition numbers using LAPACK. The accuracy is obtained using a 5-fold cross validation. The matrices are assigned to each part in the training data and the test data in a round-robin way. We choose C to be 10000 for SVR which works best according to the results of the 5-fold cross validation. Other parameters for the kernels are chosen in the same way. The feature selection criteria used are correlation, w_{SVM}, w_{cmp}, and w_{comb}. We compare them together with the SVR without feature selection on three kernels: linear kernel, polynomial kernel and RBF kernel. Here all the feature selection methods choose 50% of the features.

Figure 4.1 shows the accuracy comparison using a RBF kernel ($\gamma = 0.1$). The figure illustrates the percentage of all matrices for which the relative differences between the computed values and the predicted values of the condition number are within 10, 10^2, 10^3, 10^4, respectively. For the RBF kernel, w_{cmp} does not work well. Its accuracy is the lowest. For all the other methods, we can safely say that more than 70% of the matrices have relative differences smaller than 10^2. Among them, feature selection with w_{comb} works best for all the difference scales except the first one. Using feature selection with w_{comb}, 76.2% of the matrices have relative differences smaller

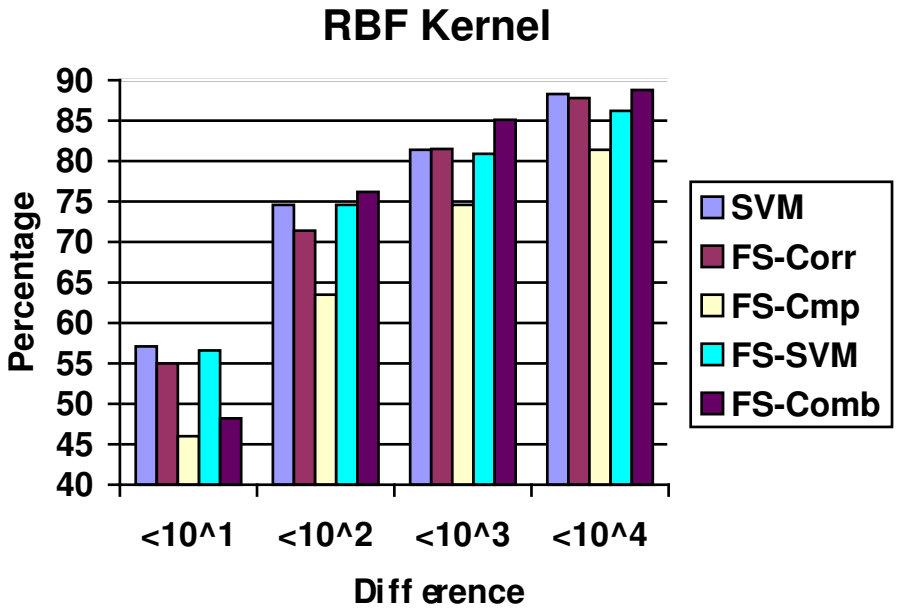


Figure 4.1: Comparison of accuracy with a RBF kernel.

than 10^2. It has better accuracy than using SVR alone, although it only uses half of the features. Other feature selection methods can also obtain similar accuracy to that obtained from SVR without feature selection.

Figure 4.2 displays the accuracy comparison using a linear kernel. Here both feature selection criteria w_{comb} and w_{SVM} work well. Their accuracy are higher than using SVR alone for all the difference scales. For the first two difference scales, w_{SVM} is better than w_{comb}, while for the last two, w_{comb} exceeds w_{SVM}. Correlation criterion performs worst for the linear kernel.

The accuracy obtained using a polynomial kernel ($d = 2$) is depicted in Figure 4.3. In this figure, all the feature selection methods obtain much better accuracy than that without feature selection. Here w_{SVM} and w_{comb} perform similarly, both are better than the other methods.

Put these 3 figures together, we can see that the best accuracy is obtained using the RBF kernel, then the linear kernel, and the polynomial kernel does not seem to

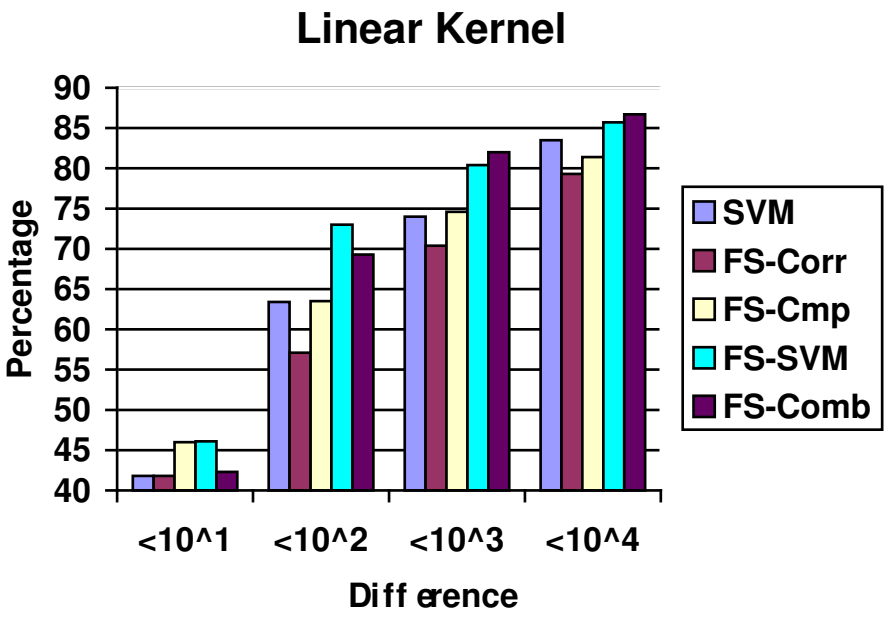


Figure 4.2: Comparison of accuracy with a linear kernel.

fit for this job. For the RBF kernel SVR without feature selection works rather well, thus the advantage of the feature selection methods over it is moot. However, for the polynomial kernel, when SVR without feature selection performs poorly, using feature selection methods can remarkably improve accuracy. Among the feature selection methods, the performance of the criteria using w_{comb} or w_{SVM} are consistently good.

Next, we compare the performance of the feature selection methods with different number of features selected. We test the accuracy using 25%, 50%, 75% of the features respectively, and make comparison with using 100% of the features, that is, running SVM without any feature selection. Here we choose the percentage of matrices with relative condition number differences smaller than 10^2 as accuracy. Figure 4.4 illustrates the results obtained with a RBF kernel ($\gamma = 0.1$). Only feature selection with correlation has the property that with more features used the system becomes more accurate. For feature selection using w_{comb} and w_{SVM}, choosing 50% of the features seems to yield optimal results, with which they gain the highest accuracy.

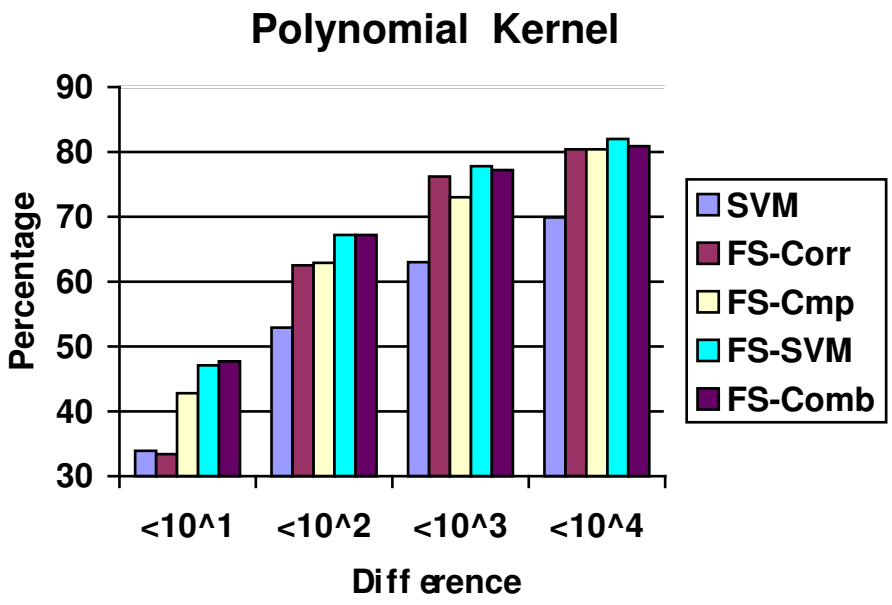


Figure 4.3: Comparison of accuracy with a polynomial kernel.

For feature selection using w_{cmp} it is an opposite story, choosing 50% of the features gives the worst results.

Table 4.1 shows the first 50% of the features chosen by the combinational method with a RBF kernel. The features are ranked according to their w_{cmp} values. The meaning of the features are explained in Section 2.2. This table tells us which features have more influence on the prediction of matrix condition number according to the combinational feature selection method.

For linear kernel, all feature selection criteria except w_{cmp} seem to find their optimized feature sets with 50% of the features. They get the best accuracy with 50% of the features. w_{cmp}, like using RBF kernel, performs the worst with 50% of the features (see Figure 4.5).

In Figure 4.6, nearly all the feature selection criteria get their best accuracy with 25% of the features. Even with the only exception w_{SVM}, the accuracy obtained with 25% of the features is very close to its best accuracy obtained with 50% of the

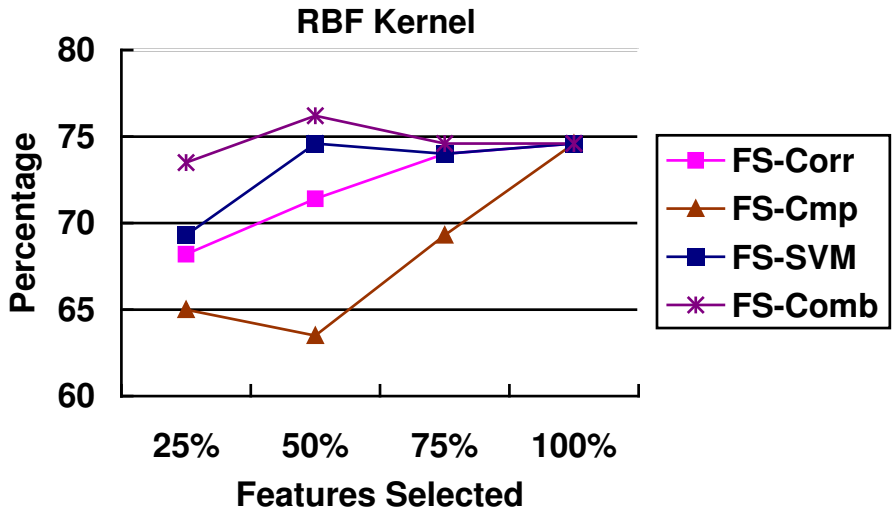


Figure 4.4: Comparison of accuracy with a RBF kernel with different percentage of features selected.

features. Figure 4.6 explains why polynomial kernel ($d = 2$) does not work as well as RBF kernel and linear kernel in Figures 4.1 - 4.3. In these 3 figures, 50% of the features are used. Thus almost all the feature selection methods find their optimal feature sets for RBF kernel and linear kernel, but not for the polynomial kernel. The feature selection methods work well with 25% of the features for the polynomial kernel. Figure 4.6 also suggests that the polynomial kernel is worthwhile to try, as the fewer features used, the less the response time.

4.4.2 Response Time

Given a matrix, the time used to obtain the condition number is referred to as the response time. The response time for the LAPACK method is the time to compute the condition number using LAPACK routines. The response time for the prediction method includes the time to compute matrix features and the time for prediction. Here we also compare the response time for prediction using the whole matrix features and using half of the features selected based on w_{SVM}.

Table 4.1: The first 50% of the features chosen by the combinational method with a RBF kernel.

Rank	Feature	w_{cmp}	Rank	Feature	w_{cmp}
1	diagvalrate	0.6218	16	diagdomrow	0.2402
2	onenorm	0.4992	17	avband	0.2274
3	upfillrt	0.4688	18	nnzrt	0.2201
4	minonenorm	0.3978	19	maxnnzpcolrt	0.2169
5	sdavnnzpcolrt	0.3941	20	nnz	0.2156
6	sdavlowrt	0.3679	21	symfnormrt	0.2053
7	avnnzpcolrt	0.3164	22	plow	0.2030
8	blocksize	0.2870	23	sdavvalfdrt	0.1983
9	nsymfnormrt	0.2812	24	ncol	0.1983
10	minnnzprowrt	0.2798	25	frnorm	0.1782
11	relsymm	0.2797	26	sdavnnzval	0.1768
12	lowfillrt	0.2749	27	lowbandrt	0.1735
13	avlowrt	0.2695	28	nzdiagsrt	0.1702
14	pdiag	0.2578	29	szvdiagrt	0.1701
15	sdavuprt	0.2572	30	avmaxvalfdrt	0.1687

Table 4.2: Average response time (in seconds).

LAPACK	prediction(all)	prediction(FS)
99.23	6.56	6.32

Table 4.2 shows the average response time for the 277 matrices used in our tests. The prediction methods are 15 times faster than using LAPACK on average. 6 seconds is also an acceptable time for an online query system. Prediction with feature selection is only slightly faster than without any feature selection. Using half of the features does not mean reducing the time cost by half. In our system, we usually compute a group of features in a function. Thus the time cost for calculating one feature using the function is the same as calculating all the features provided by the function. We will consider code optimization to make feature selection more beneficial in response time, once we decide which selected features are to be computed.

The prediction method is especially advantageous in response time for large size

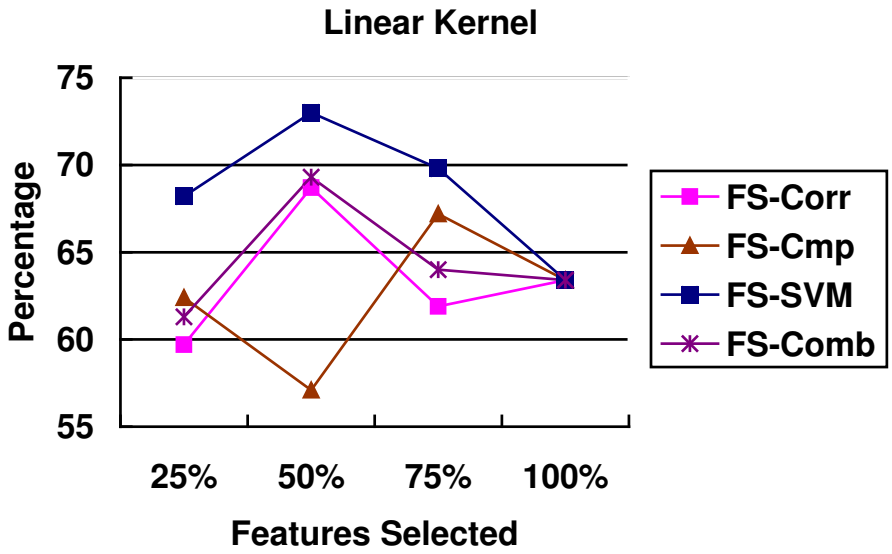


Figure 4.5: Comparison of accuracy with a linear kernel with different percentage of features selected.

Table 4.3: Average response time for larger size matrices (in seconds).

Size	NumMat	LAPACK	prediction(all)	prediction(FS)
≥ 1000	119	227.22	15.17	14.62
≥ 2000	78	340.74	22.81	21.99

matrices. For example, in Table 4.3 the average LAPACK response time for the 78 matrices with size larger than 2000 is around 6 minutes, while using the prediction methods, the response time is only about 20 seconds. A matrix with size greater than 1000 is large in our experiments though this may not be the case for some other computing systems. As LAPACK will run out of memory on our computers with matrices of size larger than 20000, we can only test matrices under this size for comparison.

Table 4.4 gives some examples of how the prediction methods exceed the LAPACK method in response time. For instance, LAPACK uses about two and a half hours to compute the condition number of the matrix ADD20, the prediction methods only need less than one second. Although this may not be true for all the matrices, the

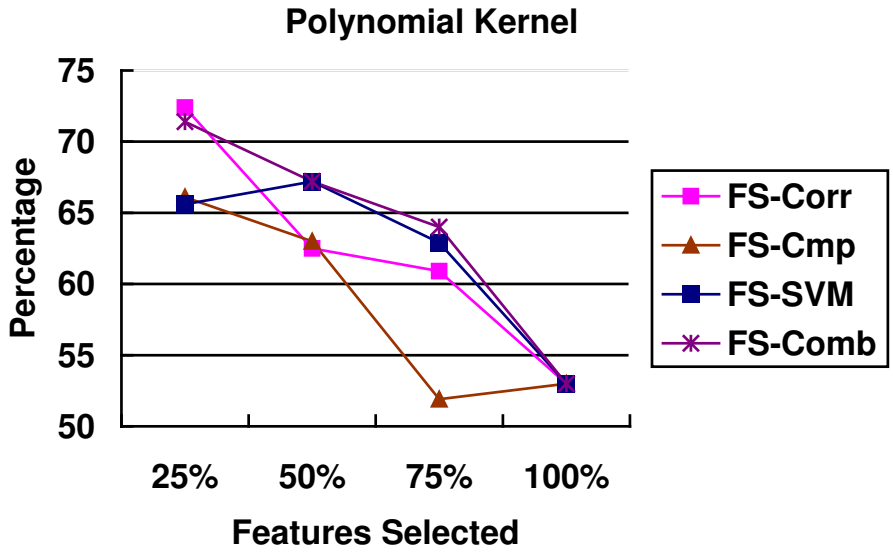


Figure 4.6: Comparison of accuracy with a polynomial kernel with different percentage of features selected.

Table 4.4: Performance comparison for some large size matrices (in seconds).

Matrix name	Size	nnz	LAPACK	prediction(all)	prediction(FS)
ADD20	2395	13151	8206.7	0.94	0.81
CRY10000	10000	49699	2262.4	23.41	24.01
LNS_3937	3937	25407	2977.0	1.38	1.33
PSMIGR_1	3140	543160	2129.8	15.82	3.31

significantly reduced response time is exactly our motivation for using prediction and feature selection.

4.5 Conclusion

In this chapter we propose a new approach to estimating the condition number of a matrix - predicting them from the matrix features. We use SVM regression with feature selection. The experiments show that around 75% of the matrices can be predicted with a relative difference from the computed condition number within 10^2. The accuracy is low compared with direct computation or estimation, but it is enough

for those people who just want to know whether the matrix in question is *well-conditioned* or *ill-conditioned*. The advantage of the prediction method is that the response time is very low, especially for large size matrices. Thus it is desirable for an online condition number query.

We also tried several feature selection methods. We proposed a combinational feature selection criterion which uses both the weights from SVR and from comparison of a matrix and its preconditioned counterpart. The experimental results show that using feature selection can reduce the time cost and improve or maintain the accuracy. The combinational feature selection criterion is one of the best methods tested.

Our future work includes further improve the accuracy of the prediction method. We may also work out the error bound for the accuracy of the proposed method to increase the confidence of the user. We need to find more representative matrix features, as well as try other feature selection methods to better utilize these features. Following the idea of this chapter, we may also try to predict other important matrix attributes such as rank and eigenvalue distribution.

Chapter 5

ILU0 and ILUK Prediction

One of the crucial parts of the Intelligent Preconditioner Recommendation System is to predict the solving status of the sparse matrices, which is the main topic of this and the next chapter. That is, given a sparse matrix, predicting whether the matrix can be solved by a certain preconditioned solver (preconditioner). If not, predicting the reason why it fails. The preconditioners that we choose to study in this chapter are ILU0 and ILUK. Such preconditioners are chosen because they are relatively simple, and they do not have many free parameters used in the construction of the preconditioners that can influence their performance. Thus the results of such preconditioned solvers are easier to predict. We will deal with more sophisticated preconditioner ILUT in the next chapter.

Our method of predicting the solvability of ILU0 and ILUK works as follows. First, we extract features from matrices, then these features are used to classify the matrices into groups, with matrices in the same group having the same solving status. There are many classification algorithms in literature. We choose Support Vector Machine Classification (SVC) [53, 54] because of its success in many classification problems. We also propose a new prediction method [59]: The matrices are first divided into clusters according to their similarity in the features. We define the *purity* of a cluster to be the maximum percentage of the sparse matrices in the cluster that

have the same solving status. Then SVC is applied only to the clusters with low purity values. Our experimental results show that we not only improve the accuracy of the prediction but also gain some insights into the relationship between matrix features and solving status of the matrix by the preconditioned solvers. The overall accuracy of the prediction is above 90% for the ILU0 preconditioner and above 87% for the ILUK preconditioners.

This chapter is organized as follows: We briefly review the clustering and classification algorithms used in the IPRS system in Section 5.1. The proposed prediction method is described in Section 5.2. The computational experiments are carried out and the results are discussed in Section 5.3. We sum up this chapter in Section 5.4.

5.1 Clustering and Classification

In this section, we briefly review the clustering and classification algorithms we use in the system.

5.1.1 K-means Clustering

The K-means algorithm [36] (with its many variants) is a popular clustering method in data mining. It gains its popularity due to its simplicity and intuition. The algorithm is an iteration procedure and requires that the number of clusters, k, be given *a priori*. Suppose that the k initial cluster centers are given, the algorithm iterates as follows:

(1) It computes the Euclidean distance from each of the objects to each cluster center. An object is assigned to the cluster with the smallest distance.

(2) Each cluster center is recomputed to be the mean of its constituent objects.

(3) Repeat steps (1) and (2) until the convergence is reached.

The criterion function for the convergence can be computed, e.g., as

$$f_r = \frac{1}{n}\sum_{i=1}^{n}(\text{Edist}^2(d_i, c_j^{(r)})),$$

where r is the step of the iterations. The function $\text{Edist}(d_i, c_j)$ computes the Euclidean distance between the object d_i and a cluster center c_j. Given a convergence criterion ϵ, the K-means algorithm stops when $|f_{r+1} - f_r| < \epsilon$. Note that f_r is a monotonically decreasing function with a lower bound. So its limit exists [15].

5.1.2 SVM Classification

The SVM (Support Vector Machine) is based on a structural risk minimization theory [53]. It has been successfully applied to many applications like face identification, text categorization, bioinformatics, etc [7, 8, 35].

In SVM classification the goal is to find a hyperplane that separates the examples with maximum margin. Given l examples (x_1, y_1), ..., (x_l, y_l), with $x_i \in R^n$ and $y_i \in \{-1, 1\}$ for all i, SVM classification can be stated as a quadratic programming problem:

$$\text{minimize } \frac{1}{2} \parallel w \parallel^2 + C\sum_{i=1}^{l}\xi_i$$

$$\text{subject to } \begin{cases} y_i(< w, x_i > + b) \geq 1 - \xi_i \\ \xi_i \geq 0 \\ C > 0 \end{cases}$$

where C is a user-selected regularization parameter, and ξ_i is a slack variable accounting for errors. After solving it, we can get the following decision function:

$$f(x) = \sum_{i=1}^{l}\alpha_i y_i < x_i, x > + b. \tag{5.1}$$

where $0 \leq \alpha_i \leq C$.

For the nonlinear case, we apply a kernel function, $K(x, x_i) =< \Phi(x), \Phi(x_i) >$, which maps the input space into some reproduced kernel feature space. Then Equa-

tion (5.1) can be rewritten as:

$$f(x) = \sum_{i=1}^{l} \alpha_i y_i K(x, x_i) + b. \tag{5.2}$$

The above methods only work for classifying two classes. If the matrix features are to be classified into more than two classes, we have to use one of the multi-class classification methods [27, 43, 55]. Two of the commonly used multi-class methods are "one-against-one" and "one-against-all". Suppose there are n classes, "one-against-all" method constructs n classifiers. Classifier i divides the data into the class belonging to class i and those not belonging to class i. The "one-against-one" method constructs classifiers for each of the class pairs, thus totally constructing $n(n-1)/2$ classifiers. Hsu *et al.* showed in [27] that "one-against-one" is more suitable for practical use. However, a recent publication [46] argued that "one-against-all" is as accurate as any other approaches. Since using "one-against-all" can save more training time, we adopt the "one-against-all" method in this chapter.

5.2 Prediction Method

We now explain how our prediction method works. The prediction method actually consists of two parts, the training algorithm and the classification algorithm.

ALGORITHM **5.2.1.** Training Algorithm.

1. Compute the feature matrix M_f;
2. Apply K-means(k,M_f);
3. For clusters $c_1, \ldots, c_k$
4. If ($Purity_i < \omega$)
5. SVC_train(c_i).

In the training algorithm (Algorithm 5.2.1), we first calculate the features of each sparse matrix and save them in a feature matrix M_f. Then we use the K-means algorithm to cluster all the objects (matrices) into k clusters according to their features. For each of the clusters, if its purity value is greater than a threshold value ω, most matrices in this cluster are considered to have the same solving status. If the purity value of a cluster is smaller than ω, we use the SVM classification algorithm to build training models for the cluster.

ALGORITHM **5.2.2.** Classification Algorithm.

1. Given a matrix A, compute its features;
2. Find its nearest cluster c_i;
3. If($Purity_i > \omega$)
4. $\quad SS_A = SS_{c_i}$
5. Else
6. $\quad SS_A =$ SVC_classify(c_i,A).

Given a new matrix A, if we want to predict its solving status, we will apply the classification algorithm (Algorithm 5.2.2). First, the matrix features are computed. Then, the distance of the feature vector to the center of each cluster is calculated. The matrix is assigned to the cluster with the shortest distance. If the purity value of the cluster is greater than ω, the solving status of the matrix is defined as the solving status of the majority of the matrices in the cluster. Otherwise, the SVC training models of that cluster are used to classify the matrix to its corresponding solving status group.

After a certain number of new matrices have been inserted into each cluster, the training algorithm will be run again to reflect the changes brought by the newly added matrices.

5.3 Experiments and Results

We conduct some experiments to test the prediction method. The linear systems are constructed by using sparse matrices from MatrixMarket [38]. The right hand sides of the linear systems are constructed by assuming that the solutions are a vector of all ones. The initial guessed solutions are a vector of all zeros. The maximum number of iterations is 500. The convergence stopping criterion is that the 2-norm of the residual vectors is reduced by 7 orders of magnitude. The iterative method used is GMRES(20) and the preconditioners are matrix structure-based incomplete LU factorizations ILU0 and ILUK. We choose K to be 1, 2, 3 respectively and denote them as ILUK1, ILUK2 and ILUK3. We first predict whether a matrix can be solved by these preconditioned solvers. If we predict that a matrix cannot be solved by a particular preconditioner, we will also predict the reason(s) of the failure. Table 5.1 shows the possible solving status (SS) of running these preconditioned solvers and their meanings. If the prediction method encounters any problem during its processing, it will stop and return a corresponding status to indicate the problem. We use SVM^{Light} [29] for SVM classification. The results are obtained by using a 5-fold cross validation.

Table 5.1: Solving status (SS) and the meanings.

SS	Meaning
-100	zero pivot in constructing a preconditioner
-5	zero row in constructing a preconditioner
-8	large condest, unstable preconditioner
-1	solver cannot converge in 500 iterations
0	successfully solved

5.3.1 Solving Status Prediction

Here we compare the prediction results obtained by using the proposed prediction method (CSVC) and by using SVM classification (SVC) alone. A total of 66 matrix features are extracted and used. In the K-means algorithm, we choose k to be 12. ω is set to be 85%. Thus if more than 85% of the matrices in a cluster have the same SS, we will not do further classification on that cluster. In SVM classification, we apply the one-against-all method, constructing classifiers for each of the solving status and choosing the one with the highest value as a sparse matrix's final predicted solving status.

Table 5.2: Predicted solving status related to ILU0.

SS	0	-100	-1	-8	Total
NM	143	132	32	12	319
NM_{SVC}	127	128	13	2	270
CT_{SVC}	88.8%	97.0%	40.6%	16.7%	84.6%
NM_{CSVC}	136	128	21	5	290
CT_{CSVC}	95.1%	97.0%	65.6%	41.7%	90.9%

The predicted solving status related to ILU0 are listed in Table 5.2. Here NM denotes the number of matrices, while CT denotes the correct rate (of successful prediction). Of all the 143 successfully solved matrices, we can correctly predict 136 of them by using CSVC, with the correct rate of 95.1%. On the other hand, if we use SVC, the correct rate is only 88.8%. If a matrix factorization encounters a zero pivot error, we can predict it with a correct rate of 97.0% with both SVC and CSVC. The accuracy is not high for predicting the cases of SS $= -1$ or SS $= -8$, the major reason for the poor predictions is that there are too few examples of such cases in the data sets. However, we can see the improvement of the prediction accuracy by using CSVC in stead of SVC. The correct rates are raised by 25% for both SS $= -1$ and SS $= -8$. Finally, the total correct rate for predicting ILU0 by SVC is 84.6%, while

using CSVC it rises to 90.9%.

Table 5.3: Predicted solving status related to ILUK.

SS	0	-5	-1	-8	Total
ILUK1					
NM	197	74	37	11	319
NM_{SVC}	187	61	15	6	269
CT_{SVC}	94.9%	82.4%	40.5%	54.6%	84.3%
NM_{CSVC}	185	67	26	3	281
CT_{CSVC}	93.9%	90.5%	70.3%	27.3%	88.1%
ILUK2					
NM	198	69	34	18	319
NM_{SVC}	182	59	16	7	264
CT_{SVC}	91.9%	85.5%	47.1%	38.9%	82.8%
NM_{CSVC}	183	63	22	10	278
CT_{CSVC}	92.4%	91.3%	64.7%	55.6%	87.1%
ILUK3					
NM	203	69	31	16	319
NM_{SVC}	182	58	17	5	262
CT_{SVC}	89.7%	84.1%	54.8%	31.3%	82.1%
NM_{CSVC}	187	64	21	8	280
CT_{CSVC}	92.1%	92.8%	67.7%	50.0%	87.8%

We compare the accuracy in predicting the solving status of ILUK1, ILUK2 and ILUK3 in Table 5.3. For ILUK1, CSVC excels SVC in predicting SS $= -5$ and SS $= -1$, but its correct rate of predicting SS $= 0$ is slightly lower than that of SVC. It performs worse than SVC in predicting SS $= -8$, too. The total correct rate of CSVC is 88.1%, which is higher than that of SVC (84.3%). Compared with the results of ILUK1, CSVC works better for ILUK2 and ILUK3. It has higher accuracy in predicting every solving status. For ILUK2, it increases the CT by around 17% for SS $= -5$ and SS $= -1$. The total correct rate is raised from 82.8% to 87.1%. For ILUK3, CSVC exceeds SVC for every solving status, too. It gets 87.8% in total correct rate while SVC only gets 82.1%.

In conclusion, the prediction accuracy is improved in almost all the categories

with CSVC for ILUK. Using CSVC, the correct rate of predicting whether a sparse matrix can be successfully solved is above 92% for all levels of the ILUKs. And the total correct rate is above 87%.

5.3.2 Choice of ω

In the previous subsection, we choose ω to be 85% because it works best for our system, after a large number of experiments. Figure 5.1 shows the change of total correct rates of using CSVC for ILU0 and ILUKs with the increase of ω. We can see that the trend of the lines representing ILU0 and ILUKs are very similar. When ω is not large enough, with the increase of ω, CT increases. It means when the cluster is not very pure, choosing the solving status of the majority of the matrices as the solving status for every matrix will incur a large error. In this case, it is better to construct classification models for each of the solving status group in the cluster and predict the solving status of each matrix using these models. However, after ω reaches some point (85% in our experiments), the prediction error of the classification models is larger than not using these models. We do not need to construct classification models in this situation and can save training time.

Another point of Figure 5.1 is that even the lowest CT of ILU0 and ILUK in the figure is higher than the CT obtained by using SVC alone. It suggests that even if the ω value used is not the best one, it can also improve the prediction accuracy by using the CSVC method.

5.3.3 Cluster Analysis

Table 5.4 lists some statistics of the 12 clusters. P_{ilu0} denotes the purity of the cluster with respect to the solving status related to ILU0, and SS_{ilu0} denotes the solving status of the majority of the matrices in the cluster. Other columns for ILUKs have

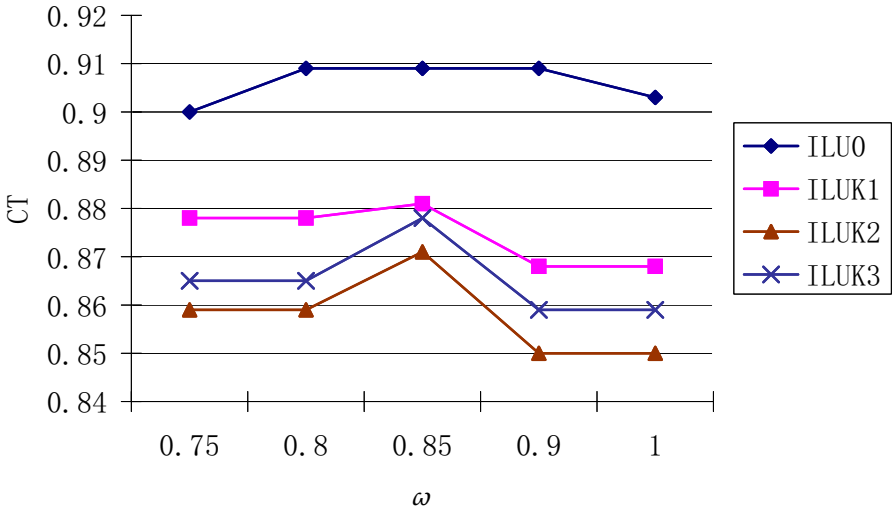


Figure 5.1: Relation of ω and the total correct rate.

the similar meanings. There are 6 clusters with the purity value equal to 1 for both ILU0 and ILUKs. Matrices in Cluster 1 and Cluster 9 all have SS $= -100$ for ILU0 and SS $= -5$ for ILUKs. All matrices in Clusters 7, 10, 11, and 12 can be successfully solved. The last line of the table shows the average purity values for ILU0 and ILUKs. We can get:

$$\text{average}(P_{ilu0}) > \text{average}(P_{iluk3}) >$$

$$\text{average}(P_{iluk2}) > \text{average}(P_{iluk1})$$

If we compare that with the improvement of total correct rate obtained by using CSVC instead of SVC, we can get a similar sequence:

$$\text{improvement}(CT_{ilu0}) > \text{improvement}(CT_{iluk3}) >$$

$$\text{improvement}(CT_{iluk2}) > \text{improvement}(CT_{iluk1})$$

It seems that CSVC works better with higher average purity values. Our experiences in the experiments also show this trend. It suggests that if we use some clustering methods which can achieve higher purity value, we can gain higher total correct rate.

We will do some experiments in the future to verify whether this hypothesis is true or not.

Table 5.4: Cluster statistics.

Clusters	NM	P_{ilu0}	SS_{ilu0}	P_{iluk1}	SS_{iluk1}	P_{iluk2}	SS_{iluk2}	P_{iluk3}	SS_{iluk3}
Cluster 1	11	1.0	-100	1.0	-5	1.0	-5	1.0	-5
Cluster 2	72	0.69	-100	0.48	0	0.50	0	0.56	0
Cluster 3	26	0.88	-100	0.62	0	0.69	0	0.73	0
Cluster 4	27	0.37	-100	0.26	-1	0.30	-1	0.37	-1
Cluster 5	16	0.69	0	0.75	0	0.81	0	0.81	0
Cluster 6	18	0.78	0	0.83	0	0.83	0	0.83	0
Cluster 7	5	1.0	0	1.0	0	1.0	0	1.0	0
Cluster 8	50	0.92	0	0.90	0	0.86	0	0.86	0
Cluster 9	31	1.0	-100	1.0	-5	1.0	-5	1.0	-5
Cluster 10	41	1.0	0	1.0	0	1.0	0	1.0	0
Cluster 11	7	1.0	0	1.0	0	1.0	0	1.0	0
Cluster 12	15	1.0	0	1.0	0	1.0	0	1.0	0
Average	26.6	0.86	-	0.82	-	0.83	-	0.85	-

5.3.4 Stable Clusters

The clusters generated by the K-means algorithm is greatly influenced by the initial cluster centers. Both the size of each cluster and its component matrices will change if the initial cluster center is changed. However, we notice that whichever the initial cluster center is chosen, some matrices will always be clustered together. We call such groups of matrices stable clusters. If the matrices in a stable cluster have the same solving status, then the cluster is called a pure stable cluster. Table 5.5 describes the composition of some pure stable clusters, each of which has more than 10 matrices. The clusters are obtained by running K-means algorithm 10 times. K is also set to be 12. Matrices with the same name come from the same source. For example, BP(9) means 9 matrices of BP type. The ILU0 and ILUKs share the first 4 pure stable

clusters. $PSC5$ is only for ILUKs, not for ILU0. All the 10 matrices in $PSC5$ have the same solving status for ILUKs, but for ILU0, 6 of them get $SS = 0$, and 4 of them get $SS = -100$. Thus it is not a pure stable cluster for ILU0.

Table 5.5: Matrix composition of the pure stable clusters.

Clusters	NM	Matrices
PSC1	19	BP(9), SHL(3), STR(4), WEST(3)
PSC2	13	BUSS(4), BCSSTK(1),BCSSTM(4), NOS(4)
PSC3	12	BCSSTK(10), NOS(2)
PSC4	14	BCSSTM(14)
PSC5	10	CAVITY(1), E05R(3), FIDAP(2), FIDAPM(1), PORES(1), BCSSTK(2)

If a given new matrix with the feature vector very close to one of these pure cluster centers, we can predict that it will show similar property as the other matrices in the cluster when solved by some preconditioned solvers. Table 5.6 shows the prediction accuracy using the centers of the above pure stable clusters. Here we normalize each matrix feature vectors, and calculate the center of each pure stable cluster. Then we find out the maximum distance ($Maxdis$) from the component matrices to the cluster center. Next we test on the 319 matrices. The distance of each matrix to each cluster center is computed. If the distance of each matrix to its nearest pure stable cluster is within the maximum distance of that cluster, we predict the matrix has the same solving status as the pure stable cluster. In Table 5.6, TN means the total number of matrices with distance to a pure stable cluster center within $Maxdis$. CN means the number of matrices whose solving status is correctly predicted. TA denotes the total correct prediction rate, while EA denotes the correct prediction rate if the number of matrices in the pure stable cluster is taken out exclusively. For ILU0, the number of matrices with distance to the center of $PSC1$ smaller than 0.8088 is 34, among them, the solving status of 31 are correctly predicted. The total correct rate is 91.2%. If we

take out the 19 matrices that belong to $PSC1$, the exclusive correct rate is 12 out of 15, that is, 80.0%. If all 4 of the pure stable clusters are counted, the total TA for ILU0 is 92.5%, while the total EA is 85.5%.

Table 5.6: Prediction accuracy using pure stable cluster centers.

Clusters	PSC1	PSC2	PSC3	PSC4	PSC5	Total
Maxdis	0.8088	0.7479	0.9023	0.8681	0.8371	-
ILU0						
SS	-100	0	0	0	-	-
TN	34	29	43	14	-	120
CN	31	23	43	14	-	111
TA	91.2%	79.3%	100%	100%	-	92.5%
EA	80.0%	62.5%	100%	100%	-	85.5%
ILUK1						
SS	-5	0	0	0	0	-
TN	25	26	33	14	32	130
CN	25	22	33	14	26	120
TA	100%	84.6%	100%	100%	81.2%	92.3%
EA	100%	69.2%	100%	100%	72.7%	83.9%
ILUK2						
SS	-5	0	0	0	0	-
TN	25	26	33	14	32	130
CN	25	22	33	14	26	120
TA	100%	84.6%	100%	100%	81.2%	92.3%
EA	100%	69.2%	100%	100%	72.7%	83.9%
ILUK3						
SS	-5	0	0	0	0	-
TN	25	26	33	14	32	130
CN	25	22	33	14	26	120
TA	100%	84.6%	100%	100%	81.2%	92.3%
EA	100%	69.2%	100%	100%	72.7%	83.9%

For ILUK1, ILUK2 and ILUK3, although they have different stable clusters, they share the same pure stable clusters. The TA and EA for $PSC1$, $PSC3$ and $PSC4$ are all 100%. The prediction accuracy using the other two pure stable clusters is a little lower. On average, the total correct rate for ILUKs is 92.3% and the total exclusive correct rate is 83.9%.

The above results show that using the pure stable cluster center is a good way to predict the solving status of matrices. We will save such pure stable cluster center and $Maxdis$ pairs in the knowledge base to help improve the accuracy of prediction.

5.4 Conclusion

The experimental results show that the prediction method gets promising results in predicting the solving status of sparse matrices by the matrix structure-based ILU type preconditioners. The overall accuracy of the prediction is above 90% for the ILU0 preconditioner and above 87% for the ILUK preconditioners. Using CSVC can improve the prediction accuracy over using SVC alone, the pure stable clusters generated can also provide us with a good way to predict which kind of matrices can get what results when solved by these preconditioned solvers.

Chapter 6

ILUT Prediction

In this chapter we continue the work of the previous chapter to predict the solving status of a sparse linear system using a certain preconditioned solver. The preconditioner we work on in this chapter is ILUT, one of the popular preconditioners with many successful applications. Unlike ILU0 or ILUK, ILUT needs two preset parameters and it only works well under some sets of values of these parameters. Different sparse linear systems usually can be solved with ILUT with different parameters. Our aim in this chapter is to predict with which parameter sets the sparse linear systems can be solved by ILUT.

This chapter is organized as follows: Section 6.1 explains the problem encounters in predicting the solving status of a sparse linear system using ILUT and our method to resolve the problem. Singular value decomposition (SVD) and Sparsified SVD are introduced in Section 6.2. The experiments are carried out and the results are reported in Section 6.3. Conclusion of this chapter is in Section 6.4.

6.1 Introduction

ILUT is a kind of incomplete LU preconditioner with double dropping strategies. It is one of the popular preconditioners with many successful applications [34, 31].

PGMRES with ILUT can solve some sparse linear systems that would fail other preconditioners like ILU0 (e.g., matrix F2DB). With ILUT, PGMRES may reduce the number of iterations to lower the computational time [48].

Corresponding to the two dropping strategies, there are two parameters used in ILUT. One is tolerance value tol, the other is the number of fill-in $lfil$. As we have explained in Section 1.2, we will use the fill-in rate $filr$ instead of $lfil$. Whether a sparse linear system can be solved by ILUT and the number of iterations the PGMRES will take are closely related to the value of these two parameters. Given a sparse linear system, we want to predict all the possible combination of the parameters with which the linear system can be solved. It is a quite hard problem, as both of the parameters may take real positive values, the possible combination of the parameters may construct arbitrary areas in a two-dimensional parameter space. The left subfigure in Figure 6.1 illustrates the parameter space of ILUT. The closed grey area represents the parameter area within which a certain sparse linear system can be solved. As the area may be irregular and open and there may exists more than one such areas, it is very difficult to find some functions to describe such area(s).

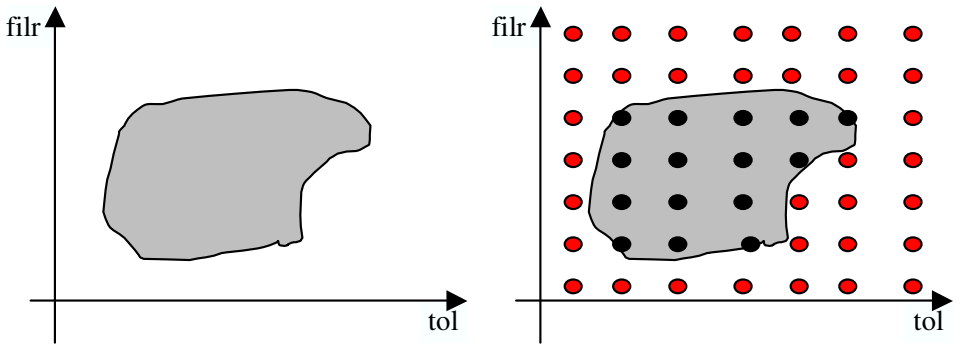


Figure 6.1: Parameter space of ILUT.

Our current method to solve this problem is to choose some points in the parameter space as samples. Studying the performance of ILUT with these sample parameters,

we can get the main idea of what kinds of combination of the parameters are favorable for a given sparse linear system. In the right subfigure in Figure 6.1, the dots represent samples. If we can correctly predict the solving status at these samples (red dot means that with the sample parameters the sparse linear system cannot be solved; back dot means it can be solved), we may obtain an outline of the parameter area(s) in which the sparse linear system can be solved. We use SVM classification to predict the solving status of the sparse linear systems by ILUT with a specific set of parameters. We also use SVD and sparsified SVD to preprocess the matrix features to improve the accuracy of prediction.

6.2 SVD and Sparsified SVD

6.2.1 Singular Value Decomposition

Singular Value Decomposition (SVD) [20] is a popular method in data mining and information retrieval [14]. It is usually used to reduce the dimensionality of the original dataset.

Let A be a sparse matrix of dimension $n \times m$ representing the original dataset. The rows of the matrix correspond to data objects and the columns to attributes. The singular value decomposition of the matrix A is [20]

$$A = UHV^T,$$

where U is an $n \times n$ orthonormal matrix, $H = \text{diag}[\sigma_1, \sigma_2, \ldots, \sigma_s]$ $(s = \min\{m, n\})$ is an $n \times m$ diagonal matrix whose nonnegative diagonal entries are in a descending order, and V^T is an $m \times m$ orthonormal matrix. The number of nonzero diagonals of H is equal to the rank of the matrix A.

Due to the arrangement of the singular values in the matrix H (in a descending order), the SVD transformation has the property that the maximal variation among

the objects is captured in the first dimension, as $\sigma_1 \geq \sigma_i$ for $i \geq 2$. Similarly much of the remaining variations is captured in the second dimension, and so on. Thus, a transformed matrix with a much lower dimension can be constructed to represent the original matrix faithfully. Define

$$A_k = U_k H_k V_k^T,$$

where U_k contains the first k columns of U, H_k contains the first k nonzero diagonals of H, and V_k^T contains the first k rows of V^T. The rank of the matrix A_k is k. With k being usually small, the dimensionality of the dataset has been reduced dramatically from $\min\{m, n\}$ to k (assuming all attributes are linearly independent). It has been proved that A_k is the best k dimensional approximation of A in the sense of Frobenius norm.

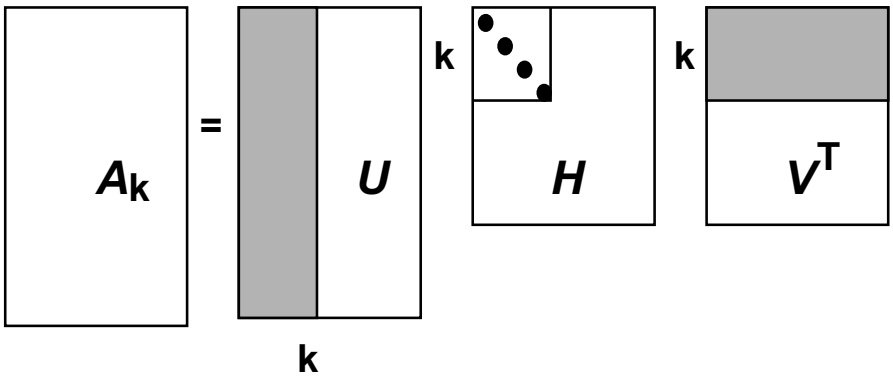


Figure 6.2: Singular value decomposition and reduced dimension.

In data mining applications, the use of A_k to represent A has another important function. The removed part $E_k = A - A_k$ can be considered as the noise in the original dataset A [5]. Thus, in many cases, mining on the reduced dataset A_k may yield better results than mining on the original dataset A.

6.2.2 Sparsified SVD

The SVD sparsification concept was proposed by Gao and Zhang in [19] for reducing the storage cost and enhancing the performance of SVD in text retrieval applications. Several sparsification strategies were proposed and experimented in [19]. The one that we used in this section is the simplest one.

After reducing the rank of the SVD matrices, we set some small size entries, which are smaller than a certain threshold ϵ, in U_k and V_k^T, to zero. We refer to this operation as the dropping operation [19]. For example, given a threshold value ϵ, we drop u_{ij} in U_k if $|u_{ij}| < \epsilon$. Similarly, an element v_{ij} in V_k^T is also dropped if $|v_{ij}| < \epsilon$. Let $\overline{U}_k$ denote U_k with dropped elements and $\overline{V}_k^T$ denote V_k^T with dropped elements, we can represent the sparsified data matrix $\overline{A}_k$, with

$$\overline{A}_k = \overline{U}_k H_k \overline{V}_k^T.$$

The sparsified SVD method is equivalent to further removing noise from the dataset A_k. Denote $E_\epsilon = A_k - \overline{A}_k$, we have

$$A = \overline{A}_k + E_k + E_\epsilon.$$

6.3 Experiments and Results

We conduct some experiments to test the prediction accuracy of the solving status of the 319 sparse linear systems by PGMRES with preconditioner ILUT with different parameter sets. The sparse linear systems are constructed by using sparse matrices from MatrixMarket [38]. The parameter setting in PGMRES is the same as in Chapter 5. We use SVM^{Light} [29] for SVM classification and Matlab [37] for SVD. The results are obtained by using a 5-fold cross validation.

6.3.1 Prediction with SVM Classification

In this section we test the accuracy of predicting the solving status of sparse linear systems with the combination of parameters using SVM classification. For parameter tol, we choose the most often used values 0.1, 0.01, 0.001, 0.0001 and 0.00001. For parameter $filr$, the sample values we choose are 1, 2, 3, 4, and 5. The kernel we used in SVM classification is RBF, which is expressed by:

$$\text{RBF} : K(x, x_i) = e^{-\frac{||x-x_i||^2}{2\sigma^2}}.$$

Figure 6.3 shows the average prediction accuracy with different combination of parameters. Here σ is set to be 0.1 in RBF kernel. We can see the highest prediction accuracy 92.79% is obtained with $tol = 0.001$ and $filr = 1$. The lowest prediction accuracy 83.6991% is obtained with $tol = 0.0001$ and $filr = 3$. Generally speaking, when tol is high, e.g., $tol = 0.1$ and $filr$ is low, e.g. $filr = 1$, the prediction accuracy is better. With the increase of $filr$ and the decrease of tol, prediction accuracy becomes lower.

tol \ filr	1	2	3	4	5
0.1	0.921630	0.909091	0.902821	0.915361	0.915361
0.01	0.921630	0.909091	0.899687	0.902821	0.921630
0.001	0.927900	0.902821	0.868339	0.862069	0.858934
0.0001	0.912226	0.899687	0.836991	0.843260	0.884013
0.00001	0.924765	0.899687	0.843260	0.862069	0.865204

Figure 6.3: Prediction accuracy of SVM Classification ($\sigma = 0.1$).

Figure 6.4 also shows the average prediction accuracy with different combination of parameters. The difference from Figure 6.3 is that σ is set to be 0.01 in RBF kernel. However, it does not depict the same prediction accuracy distribution as Figure 6.3

does. Here the highest prediction accuracy area is on the upper-right corner of the figure, with tol from 0.1 to 0.01 and $filr$ from 4 to 5.

tol \ filr	1	2	3	4	5
0.1	0.890282	0.912226	0.899687	0.915361	0.915361
0.01	0.896552	0.896552	0.890282	0.921630	0.921630
0.001	0.905956	0.896552	0.855799	0.880878	0.874608
0.0001	0.890282	0.902821	0.884013	0.890282	0.924765
0.00001	0.899687	0.909091	0.874608	0.884013	0.899687

Figure 6.4: Prediction accuracy of SVM Classification ($\sigma = 0.01$).

Figure 6.5 is obtained by setting σ to be 0.001 in RBF kernel. This figure shows another prediction accuracy pattern. This time the highest prediction accuracy is achieved with $tol = 0.01$ or $filr = 1$.

tol \ filr	1	2	3	4	5
0.1	0.905956	0.899687	0.890282	0.899687	0.899687
0.01	0.909091	0.909091	0.918495	0.921630	0.918495
0.001	0.905956	0.899687	0.846395	0.862069	0.868339
0.0001	0.905956	0.905956	0.874608	0.849530	0.902821
0.00001	0.905956	0.905956	0.880878	0.868339	0.896552

Figure 6.5: Prediction accuracy of SVM Classification ($\sigma = 0.001$).

Figure 6.6 is obtained by setting σ to be 0.0001 in RBF kernel. Its highest prediction accuracy area is also on the upper-right corner of the figure, like in Figure 6.4 but much longer. To be specific, it is the area with tol from 0.1 to 0.01 and $filr$ from 2 to 5.

Table 6.1 describes the total prediction accuracy of SVM Classification with different σ values. In calculating the total prediction accuracy we take into account

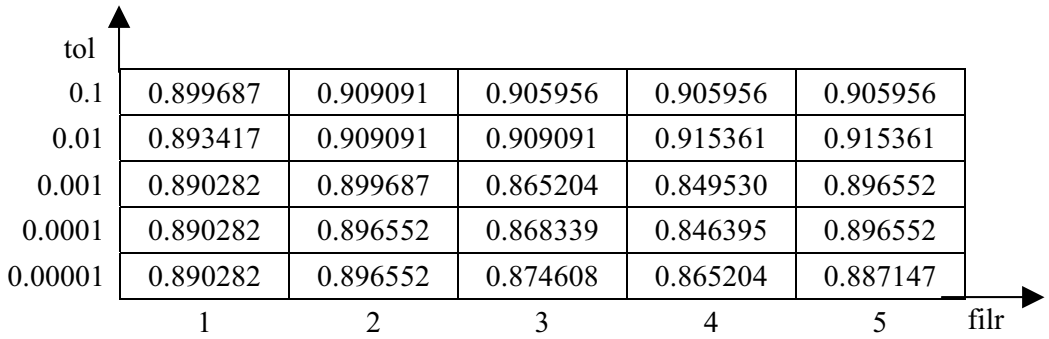


Figure 6.6: Prediction accuracy of SVM Classification ($\sigma = 0.0001$).

all the combinations of parameters. The table shows that the value of σ does not affect the total prediction accuracy much. All the total average prediction accuracy are between 89% and 90%, which means SVM Classification works well for solving status prediction. In the latter experiments, we set $\sigma = 0.001$, as it provides a good prediction accuracy and takes less time to train than using $\sigma = 0.01$.

σ	0.1	0.01	0.001	0.0001
Total Accuracy	0.892414	0.897304	0.894044	0.891285

Table 6.1: Total prediction accuracy of SVM Classification with different σ.

6.3.2 Applying SVD

In many data mining applications, using SVD to preprocess data can improve the performance of data mining algorithms. In this section, we first apply SVD to the original data set and then use SVM Classification to predict the solving status of the sparse linear systems to see if it can improve the prediction accuracy. To compare the results obtained with or without SVD, we set $\sigma = 0.001$ in RBF in this section.

Table 6.2 shows the prediction accuracy obtained by applying SVD with rank $k = 60$ in preprocessing. The highest prediction accuracy 92.163% is obtained with $tol = 0.01$ and $filr = 4$, which is similar to the highest prediction accuracy in Figure

6.3. The lowest prediction accuracy 84.6395% is obtained with $tol = 0.0001$ and $filr = 3$, which is a little bit higher than the lowest prediction accuracy in Figure 6.3. The high prediction accuracy areas lie on the row with $tol = 0.01$ and the column with $filr = 1$.

tol	$filr = 1$	$filr = 2$	$filr = 3$	$filr = 4$	$filr = 5$
0.1	0.905956	0.899687	0.890282	0.899687	0.899687
0.01	0.909091	0.909091	0.918495	0.92163	0.918495
0.001	0.905956	0.899687	0.846395	0.862069	0.868339
0.0001	0.905956	0.905956	0.874608	0.84953	0.902821
0.00001	0.905956	0.905956	0.880878	0.868339	0.896552

Table 6.2: Prediction accuracy after applying SVD with $k = 60$.

Table 6.3 is obtained with using $k = 50$ in SVD. The high prediction accuracy pattern is exactly the same as the one showed in Table 6.2. Actually, most of the combinations of tol and $filr$ parameters have same accuracy as in Table 6.2.

tol	$filr = 1$	$filr = 2$	$filr = 3$	$filr = 4$	$filr = 5$
0.1	0.905956	0.899687	0.887147	0.899687	0.899687
0.01	0.909091	0.912226	0.918495	0.92163	0.918495
0.001	0.905956	0.899687	0.846395	0.862069	0.868339
0.0001	0.905956	0.905956	0.877743	0.84953	0.902821
0.00001	0.905956	0.905956	0.880878	0.868339	0.896552

Table 6.3: Prediction accuracy after applying SVD with $k = 50$.

Table 6.4 is obtained with using $k = 40$ in SVD. Its high prediction accuracy pattern is quite different from the ones in Table 6.2 and Table 6.3. The high prediction accuracy area lies in the first two rows with $tol = 0.1$ and $tol = 0.01$.

Table 6.5 is obtained with using $k = 30$ in SVD. In this table the high prediction accuracy area lies in the column with $filr = 2$.

Table 6.6 is obtained with using $k = 20$ in SVD. In this table the high prediction accuracy area lies in the area with $tol = 0.01$ and $filr$ from 3 to 5.

tol	$filr = 1$	$filr = 2$	$filr = 3$	$filr = 4$	$filr = 5$
0.1	0.902821	0.905956	0.887147	0.902821	0.902821
0.01	0.902821	0.912226	0.915361	0.92163	0.918495
0.001	0.893417	0.896552	0.84326	0.865204	0.868339
0.0001	0.887147	0.902821	0.874608	0.84953	0.915361
0.00001	0.896552	0.902821	0.877743	0.874608	0.899687

Table 6.4: Prediction accuracy after applying SVD with $k = 40$.

tol	$filr = 1$	$filr = 2$	$filr = 3$	$filr = 4$	$filr = 5$
0.1	0.902821	0.902821	0.896552	0.899687	0.899687
0.01	0.896552	0.905956	0.802508	0.9279	0.915361
0.001	0.902821	0.899687	0.852665	0.880878	0.884013
0.0001	0.896552	0.905956	0.884013	0.805643	0.909091
0.00001	0.899687	0.902821	0.887147	0.874608	0.896552

Table 6.5: Prediction accuracy after applying SVD with $k = 30$.

To sum up all the patterns of high prediction accuracy areas appear in Table 6.2 - Table 6.6, we can see that high prediction accuracy is usually obtained with $tol = 0.1$ and $tol = 0.01$, or $filr = 1$ and $filr = 2$.

Table 6.7 shows the total prediction accuracy after applying SVD with different rank. The total prediction accuracy without SVD with the same σ value is 0.894044, which is the same as applying SVD with $k = 60$. When we choose $k = 50$, the total prediction accuracy after applying SVD is 0.894169, a little higher than without using SVD. But from then on, with the decrease of rank k, the total prediction accuracy

tol	$filr = 1$	$filr = 2$	$filr = 3$	$filr = 4$	$filr = 5$
0.1	0.877743	0.887147	0.880878	0.887147	0.887147
0.01	0.862069	0.890282	0.909091	0.918495	0.915361
0.001	0.874608	0.893417	0.858934	0.852665	0.880878
0.0001	0.880878	0.902821	0.874608	0.855799	0.890282
0.00001	0.884013	0.899687	0.884013	0.868339	0.890282

Table 6.6: Prediction accuracy after applying SVD with $k = 20$.

also drops. This table shows that applying SVD can improve the accuracy a little in this experiment. However, even the the rank k is set to be very small, e.g., less than one third of the number of attributes, the total prediction accuracy will not drop much either.

K	60	50	40	30	20
tol	0.894044	0.894169	0.892790	0.889279	0.884263

Table 6.7: Total prediction accuracy after applying SVD with different rank.

6.3.3 Applying Sparsified SVD

In sparsified SVD (SSVD), we drop small entries in U and V. This method can save memory space, it can also improve the performance of data mining algorithms in some applications. In this section, we conduct some experiments on preprocessing the original matrix features using sparsified SVD and then use SVM Classification to predict the solving status of the sparse linear systems to see if it can improve the prediction accuracy. To compare the results obtained with or without SSVD and with SVD, we set $\sigma = 0.001$ in RBF and $k = 50$ in SVD.

Table 6.8 shows the average prediction accuracy after applying SSVD with $\epsilon = 0.01$ using different combinations of tol and $filr$. The highest prediction accuracy 92.163% is obtained with $tol = 0.01$ and $filr = 4$, which is similar to the highest prediction accuracy in Figure 6.3 and the same as using SVD method. The lowest prediction accuracy 82.4451% is obtained with $tol = 0.001$ and $filr = 3$, which is a little lower than the lowest prediction accuracy obtained by without using SSVD and by using SVD. The high prediction accuracy area in this table lies in the area with $tol = 0.01$ and $filr$ from 3 to 5.

Table 6.9 shows the average prediction accuracy after applying SSVD with $\epsilon =$

tol	$filr = 1$	$filr = 2$	$filr = 3$	$filr = 4$	$filr = 5$
0.1	0.909091	0.896552	0.877743	0.896552	0.896552
0.01	0.896552	0.899687	0.902821	0.92163	0.912226
0.001	0.902821	0.887147	0.824451	0.84953	0.858934
0.0001	0.887147	0.890282	0.852665	0.846395	0.890282
0.00001	0.893417	0.890282	0.852665	0.846395	0.871473

Table 6.8: Prediction accuracy after applying SSVD with $\epsilon = 0.01$.

0.001. Besides the high prediction accuracy area appears in Table 6.8, there is one more small area with $tol = 0.1$ and $filr$ from 4 to 5.

tol	$filr = 1$	$filr = 2$	$filr = 3$	$filr = 4$	$filr = 5$
0.1	0.899687	0.899687	0.887147	0.902821	0.902821
0.01	0.899687	0.899687	0.902821	0.915361	0.912226
0.001	0.902821	0.890282	0.84326	0.858934	0.865204
0.0001	0.887147	0.890282	0.858934	0.84953	0.899687
0.00001	0.890282	0.893417	0.868339	0.846395	0.887147

Table 6.9: Prediction accuracy after applying SSVD with $\epsilon = 0.001$.

Table 6.10 shows the average prediction accuracy after applying SSVD with $\epsilon = 0.0001$. This time the high prediction accuracy area is exactly the same as the ones in Table 6.9.

We have also tested the prediction accuracy after applying SSVD with $\epsilon = 0.00001$. But the results are the same as the ones in Table 6.10 so we just omit it here. From the three tables in this section, we can see that the area with $tol = 0.01$ and $filr$ from 3 to 5 is the area that works best for SSVD. High prediction accuracy can be obtained by using parameters in this area with whichever ϵ values.

Table 6.11 shows the total prediction accuracy after applying SSVD with different dropping threshold ϵ. With the decrease of ϵ, the total accuracy drops as less elements in the matrix are set to be 0. Same accuracy is obtained with $\epsilon = 0.0001$ and $\epsilon = 0.00001$, as after a certain ϵ value, no more elements in the matrix will be

tol	$filr = 1$	$filr = 2$	$filr = 3$	$filr = 4$	$filr = 5$
0.1	0.899687	0.899687	0.887147	0.902821	0.902821
0.01	0.899687	0.899687	0.902821	0.915361	0.912226
0.001	0.902821	0.890282	0.84326	0.862069	0.865204
0.0001	0.887147	0.890282	0.858934	0.84953	0.899687
0.00001	0.890282	0.893417	0.868339	0.84953	0.887147

Table 6.10: Prediction accuracy after applying SSVD with $\epsilon = 0.0001$.

dropped. The best accuracy obtained by SSVD in this example is 88.2132%, which is lower than using SVD or without using any preprocessing method.

ϵ	0.01	0.001	0.0001	0.00001
Total Accuracy	0.882132	0.886144	0.886395	0.886395

Table 6.11: Total prediction accuracy after applying SSVD with different dropping threshold.

6.4 Conclusion

In this chapter, we try to predict the solving status of a sparse linear system using PGMRES with ILUT, which has two parameters. The problem is hard as it is difficult to predict all the possible areas in the parameter space that a given sparse linear system can be solved. We choose some sample points in the parameter space to predict the solving status of sparse linear systems at such sample points. Then we can have an idea of the outline of the area in the parameter space that the sparse linear system can be solved.

We use SVM classification in prediction and try SVD and sparsified SVD to preprocess the matrix features. The experimental results show that using SVM classification alone the total average accuracy of prediction on all the sample points are above 89%. The SVD method can improve the accuracy a little bit but the sparsified SVD method does not work well in this problem. We also analyze in detail the

area patterns in parameter space that can obtain high prediction accuracy in each prediction method.

Chapter 7

Best Preconditioner Selection

The goal of the Intelligent Preconditioner Recommendation System is to recommend one or more suitable preconditioners with appropriate parameters for solving a given sparse linear system. In the previous two chapters we predict whether a sparse linear system can be solved with a certain preconditioner. In this chapter we deal with the problem of selecting the preconditioners that work best for a sparse linear system.

This chapter is organized as follows: Section 7.1 introduces our algorithm to select the best preconditioners and parameters for a given sparse matrix. Memory cost analysis for PGMRES with the preconditioners ILU0, ILUK and ILUT are made in Section 7.2. The experiments are carried out and the results are reported in Section 7.3. Conclusion of this chapter is in Section 7.4.

7.1 Best Preconditioner Selection Algorithm

The goal of the Intelligent Preconditioner Recommendation System is to recommend one or more suitable preconditioners with appropriate parameters for solving a given sparse linear system. In this chapter we will select the preconditioners and parameters that work best for a sparse linear system. Here "the best" means that the sparse linear system can be solved with the least memory cost and/or time cost.

Our method works as follows. First we predict whether a sparse linear system can be solved with a certain preconditioner under some parameter settings. Then we predict the memory cost and/or time cost in solving the sparse linear system with the preconditioner under the chosen parameter settings. At last we choose the preconditioner and parameters with which the sparse linear system can be solved with the least value of $ObjValue$, which is defined as:

$$ObjValue = W_m * C_m + W_t * C_t,$$

where C_m denotes the memory usage in Kilobytes, and C_t denotes the total execution time of the preconditioned solver in number of floating point operations. W_m is the weight for memory cost and W_t is the weight for time cost. They are the parameters set by the users to represent their preference. The algorithm for our best preconditioner selection method is shown in Algorithm 7.1.1.

ALGORITHM **7.1.1.** Algorithm for Best Preconditioner Selection.

1. For each preconditioner P_i with parameter set S_i;
2. Predict solving status SS_i;
3. If (SS_i ==1) //it can be solved
4. Put preconditioner P_i with parameter set S_i into candidate pool CP;
5. Predict memory cost C_m^i;
6. Predict time cost C_t^i;
7. Calculate $ObjValue^i = W_m * C_m^i + W_t * C_t^i$
8. EndIf
9. EndFor
10. Return preconditioner P_j with parameter set S_j in CP with smallest $ObjValue^j$.

In Algorithm 7.1.1, the most difficult part is step 2, predicting the solving status of a sparse linear system by a preconditioned solver, and steps 5 and 6, predicting the memory and time cost of running the preconditioned solver. We have discussed the problem of predicting the solving status in the previous two chapters. As predicting the memory cost and the time cost are very similar, we will only focus on the problem of predicting the memory cost in this chapter. So we set $W_m = 1$ and $W_t = 0$ in Algorithm 7.1.1 which means we only care about the memory cost in selecting the best preconditioners. We think memory cost is more important than time cost since if a computer cannot allocate enough memory for a solver program, the solver program cannot run on that computer at all.

7.2 Memory Cost Analysis

In this section, we will analyze the memory cost of running PGMRES with preconditioners ILU0, ILUK and ILUT respectively. The code we used for PGMRES and the preconditioners and for memory analysis is from SPARSKIT [51].

Assume n denotes the size of a matrix; nnz denotes the number of nonzero elements in the matrix; and im denotes the size of the Krylov subspace, then the memory cost (in KB) of running PGMRES with ILU0 is:

$$C_{m\_ILU0} = 4(2n(im+5) + (im+1)im + 6im + 6nnz)/1024.$$

As the number of nonzero elements in the preconditioner ILU0 is the same as that of the original matrix A, the memory occupation of the preconditioner can be directly calculated from the attributes of A.

Let $lulen$ denote the number of nonzero elements in preconditioner, then the memory cost of running PGMRES with ILUK is:

$$C_{m\_ILUK} = 4(2n(im+5) + (im+1)im + 6im + 3nnz + 3lulen)/1024.$$

The memory cost of running PGMRES with ILUT is:

$$C_{m\_ILUT} = 4(2n(im+5) + (im+1)im + 6im + 3nnz + 3lulen)/1024.$$

We can see that the formula for calculating the memory cost of PGMRES with ILUK is the same as that of PGMRES with ILUT. However, for different preconditioners with different parameter settings, the value of *lulen* may vary greatly. And there is no exact formula to calculate the value of *lulen* for different preconditioners with different parameter settings. Here we apply SVM regression to predict the memory cost of PGMRES with ILUK and PGMRES with ILUT from matrix features.

7.3 Experiments and Results

We have conducted some experiments to test the absolute and relative errors of memory cost prediction. And we also show some example results of executing the Best Preconditioner Selection Algorithm. The parameter setting of PGMRES is the same as that in the previous chapters. We use SVM^{Light} [29] for SVM regression. The results are obtained by using a 5-fold cross validation.

7.3.1 ILUK Memory Cost Prediction

We use RBF kernel in SVM regression to predict the memory cost of PGMRES with ILUK. Table 7.1 shows the average absolute prediction error (in KB) for ILUK with different σ. The average absolute error is about 1 MB for ILUK. ILUK1 obtains the smallest absolute error with $\sigma = 0.01$, while ILUK2 and ILUK3 obtain the smallest absolute error with $\sigma = 0.001$. With the increase of level of fill k in ILUK, more fill-in elements are allowed, thus the memory cost grows, and the absolute prediction error grows as well.

The average relative prediction error for ILUK with different σ is shown in Table

σ	0.01	0.001	0.0001
ILUK1	962.4	1054.9	1628.2
ILUK2	1584.4	1262.9	1787.6
ILUK3	1842.3	1819.2	2245.7

Table 7.1: Average absolute prediction error (in KB) for ILUK with different σ.

7.2. Similar as in Table 7.1, the smallest relative error is obtained with $\sigma = 0.01$ for ILUK1, and with $\sigma = 0.001$ for ILUK2 and ILUK3. $\sigma = 0.0001$ is not a good choice for this regression task, as all the ILUK preconditioners get the largest absolute and relative error with this parameter setting.

σ	0.01	0.001	0.0001
ILUK1	0.2263	0.2480	0.3829
ILUK2	0.2748	0.2190	0.3100
ILUK3	0.2508	0.2476	0.3057

Table 7.2: Relative prediction error for ILUK with different σ.

7.3.2 ILUT Memory Cost Prediction

In this section, we will show some results of predicting the memory cost of PGMRES with ILUT using SVM regression with RBF kernel.

An Example of Prediction Results

Figure 7.1 shows some sample prediction results of running PGMRES with ILUT for a given sparse linear system. A shaded cell means using the parameters representing that cell, the sparse linear system can be solved. The sparse linear system can be solved by ILUT with 10 different parameter settings. The number in each cell denotes the estimated memory cost in Kilobytes. The best parameter setting is $tol = 0.01$ and $filr = 1$, as its estimated memory cost is 777KB, much smaller than using other shaded cells.

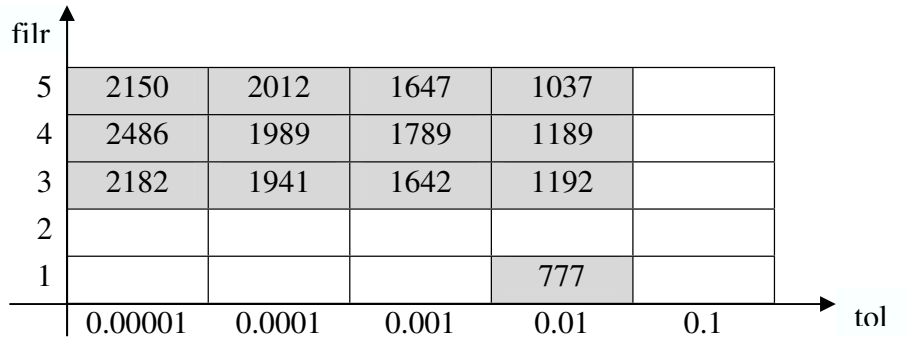


Figure 7.1: ILUT prediction results of solving status and memory cost of a matrix.

Figure 7.2 shows the real results of solving status and memory cost of the matrix used in Figure 7.1. Actually the linear system can be solved with the 10 cells predicted. There are some errors in estimated memory cost. The best parameter setting predicted in Figure 7.1 actually works best for the sparse linear system, with the real memory cost of 760KB. The prediction made in Figure 7.1 is neither the best nor the worst in our experiments. We just use it to illustrate how the solving status and memory cost predictions are made for ILUT in the IPRS system.

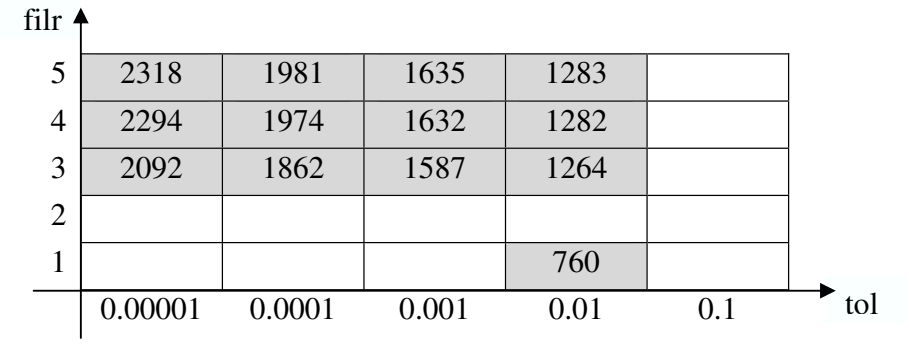


Figure 7.2: Actual results of solving status and memory cost of a matrix.

Average Error Comparison

Table 7.3 describes the total average absolute errors and relative errors of SVM regression with different σ values for ILUT. The average absolute error is about 300KB,

which is much smaller compared with that of ILUK. In calculating the total average errors we take into account all the parameter settings. The table shows that the value of σ has some effect on the prediction errors. The lowest absolute error is obtained with $\sigma = 0.1$, which is about 305KB. The lowest relative error is also obtained with $\sigma = 0.1$, which is about 24%. With the increase of σ, both the average absolute error and relative error become larger. $\sigma = 0.0001$ is the worst choice as both its absolute error and relative error are much larger than using the other σ values.

σ	0.1	0.01	0.001	0.0001
Total average absolute error (in KB)	305.9	343.4	360.0	483.5
Total relative error	0.2442	0.2742	0.2874	0.3860

Table 7.3: The comparison of total average prediction errors for ILUT with different σ.

Error Distribution

Table 7.4 shows the absolute prediction error with different combination of parameters. Here σ is set to be 0.1 in RBF kernel. With the increase of tol and $filr$, more fill-in elements are allowed in the preconditioner, thus the memory cost for storing the preconditioner as well as the absolute prediction error rise. We can see that the absolute prediction error with $tol = 0.00001$ and $filr = 5$ is more than twice as high as with $tol = 0.1$ and $filr = 1$.

tol	$filr = 1$	$filr = 2$	$filr = 3$	$filr = 4$	$filr = 5$
0.1	225.8	222.7	200.9	198.0	239.9
0.01	261.7	247.3	247.7	252.5	267.1
0.001	262.2	280.4	355.4	296.8	332.6
0.0001	264.2	286.9	314.2	436.5	407.3
0.00001	285.6	288.4	351.4	416.4	502.3

Table 7.4: Absolute prediction errors for ILUT with $\sigma = 0.1$.

Table 7.5 shows the relative prediction error with different combination of parameters. The parameter settings with large absolute errors do not necessarily mean the relative errors are high, too. We can see that the highest prediction error 29.83% is obtained with $tol = 0.1$ and $filr = 5$. The lowest prediction error 17.87% is obtained with $tol = 0.001$ and $filr = 5$. The highest relative prediction error area lies on the column $filr = 1$.

tol	$filr = 1$	$filr = 2$	$filr = 3$	$filr = 4$	$filr = 5$
0.1	0.2800	0.2771	0.2488	0.2479	0.2983
0.01	0.2908	0.2681	0.2616	0.2659	0.2799
0.001	0.2812	0.2777	0.2894	0.1880	0.1787
0.0001	0.2751	0.2659	0.2418	0.2504	0.1866
0.00001	0.2942	0.2676	0.2643	0.2573	0.2099

Table 7.5: Relative prediction errors for ILUT with $\sigma = 0.1$.

7.3.3 Results of the Best Preconditioner Selection Algorithm

In this section, we show two examples of the results of running the Best Preconditioner Selection Algorithm. The preconditioners to choose from are ILU0, ILUK1 - ILUK3, and ILUT with 25 different parameter settings.

In the first example, the sparse matrix is CAVITY09, which comes from a finite element modeling problem. Table 7.6 lists the prediction results of executing the Best Preconditioner Selection Algorithm. By prediction, CAVITY09 can be solved by 6 preconditioners with different parameter settings. The preconditioners with their parameter settings are ranked according to the memory cost. ILUK1 is predicted to be the best preconditioner as its memory cost is the lowest. Table 7.7 shows the actual best preconditioners chosen from actually solving the sparse linear systems with all the preconditioners. We can see that we make correct prediction that ILUK1 is ranked No. 1 in Table 7.7. The ranks of other preconditioners are also almost right

except that the order changes a little bit for ILUT with different parameter settings. The memory cost predicted are also close to the actual memory needed. The total relative error is only 0.93%.

Rank	Precond.	tol	filr	Memory
1	ILUK1	-	-	1944.3
2	ILUT	0.00001	5	2068.4
3	ILUT	0.001	5	2179.3
4	ILUT	0.0001	5	2347.3
5	ILUK2	-	-	2770.3
6	ILUK3	-	-	3021.9

Table 7.6: Predicted best preconditioner selection for matrix CAVITY09.

Rank	Precond.	tol	filr	Memory
1	ILUK1	-	-	2083.9
2	ILUT	0.001	5	2150.1
3	ILUT	0.0001	5	2284.1
4	ILUT	0.00001	5	2318.0
5	ILUK2	-	-	2593.4
6	ILUK3	-	-	3036.3

Table 7.7: Actual best preconditioner selection for matrix CAVITY09.

The sparse matrix in the second example is FIDAP036, one of the matrices generated by the FIDAP Package. Table 7.8 lists the prediction results of executing the Best Preconditioner Selection Algorithm. Here it is predicted that FIDAPP036 can be solved by 7 preconditioners with different parameter settings. The best preconditioner predicted is ILU0. If a sparse linear system is predicted to be able solvable by ILU0, ILU0 will definitely be the best preconditioner as it allows no fill-in elements and its memory cost can be precisely calculated. According to the prediction, ILUK1 ranks No. 2. Table 7.9 shows the actual best preconditioners chosen from actually solving the sparse linear systems with the all the preconditioners. As the sparse linear

system can actually be solved by ILU0, ILU0 is the best preconditioner for it. Just like our prediction, ILUK1 ranks No. 2. But the ranks of other preconditioners are different from our prediction except No. 6 and No. 7. The relative memory cost prediction error is 21.49%.

Rank	Precond.	tol	filr	Memory
1	ILU0	-	-	1727
2	ILUK1	-	-	2530.1
3	ILUT	0.0001	4	3039.4
4	ILUT	0.001	4	3123.3
5	ILUK2	-	-	3140.1
6	ILUT	0.001	5	3178.6
7	ILUK3	-	-	4165.8

Table 7.8: Predicted best preconditioner selection for matrix FIDAP036.

Rank	Precond.	tol	filr	Memory
1	ILU0	-	-	1727
2	ILUK1	-	-	2052.8
3	ILUK2	-	-	2555.9
4	ILUT	0.001	4	2638.2
5	ILUT	0.0001	4	2734.0
6	ILUT	0.001	5	2793.1
7	ILUK3	-	-	3010.8

Table 7.9: Actual best preconditioner selection for matrix FIDAP036.

7.4 Conclusion

In this chapter we describe our algorithm to choose the best preconditioners with parameters for a given sparse linear system, that is, to choose the preconditioner with which the sparse linear system can be solved using the least memory cost and/or time cost.

To choose the best preconditioners, we need to estimate the time and memory cost of executing the preconditioned solvers. We use SVM regression to make such predictions. The experimental results show that the relative prediction error in estimating the memory cost is about 21% for ILUK and 24% for ILUT. We also show some examples of the results of executing the Best Preconditioner Selection Algorithm.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This dissertation introduces our research work in studying and designing an Intelligent Preconditioner Recommendation System. We have conducted some research in the following directions:

- We have designed the structure of the Intelligent Preconditioner Recommendation System. This system uses data mining techniques in predicting the solving status of sparse linear systems and recommending the best preconditioners.
- We have studied and searched for useful matrix features to represent a sparse matrix. Currently we have extracted more than 60 matrix features.
- We have proposed a new prediction algorithm to estimate the condition number of a matrix. Our algorithm exceeds the traditional direct computation methods or estimation methods in response time.
- We have proposed an efficient feature selection method utilizing the difference of a matrix and its preconditioned counterparts. Using only the selected features in prediction can improve accuracy and save computational time.

- We have proposed some prediction algorithms in predicting the solving status of sparse linear systems with different preconditioners. The overall accuracy of the prediction is above 90% for the ILU0 preconditioner, above 87% for the ILUK preconditioners, and above 89% for the ILUT preconditioners.

- We have predicted the memory cost of executing PGMRES with different preconditioners. The relative prediction error is about 21% for the ILUK preconditioners and 24% for the ILUT preconditioners.

- We have proposed an algorithm to find out which preconditioners work best for a certain sparse linear system with what parameters. We use some examples to illustrate the results of the best preconditioners with parameters recommended.

8.2 Future Work

We have done some fundamental research related to the Intelligent Preconditioner Recommendation System and obtained some preliminary results. More work needs to be fulfilled to make the Intelligent Preconditioner Recommendation System a mature and useful mathematical tool. Future work may follow the directions listed below:

- The experiments in this dissertation are conducted on 319 matrices downloaded from MatrixMarket [38]. The size of the matrices varies from 27 to 23560. As the advantage of the IPRS will be shown better on large matrices, we will download and test more large matrices from, for example, the University of Florida Sparse matrix collections [12].

- One of the difficult problems in IPRS is determining the set of features that would represent the matrices in the database. The features of a matrix are directly related to the precision of the prediction system. We have used more

than 60 features in the current system. Yet there are more features that may be related to the performance of the preconditioned solvers to be explored in the future.

- We also applied some feature selection methods to find out the features that are most useful to a given task, for example, regression or classification. We may find out more efficient feature selection methods and take the cost of computing features into consideration in future work.

- The solver used in the experiments in this dissertation is PGMRES and the preconditioners are restricted to several kinds of ILU preconditioners. We may extend our work to more Krylov subspace methods like the Conjugate Gradient method and Lanczos method and so on. We may also try other kinds of preconditioners like Jacobi, Successive Overrelaxation, and Symmetric Successive Overrelaxation preconditioners, Approximate Inverse Preconditioners, block preconditioners, etc. [48].

- We proposed some algorithms to improve the accuracy of prediction of the solving status of sparse linear systems, the memory cost and some matrix features like condition number. The experiments show that the classification methods work well but the regression methods do not achieve satisfactory results. For example, around 75% of the matrices have prediction error within 10^2 in predicting condition number. We need to work out more efficient algorithms to improve the prediction accuracy.

- In the current IPRS, we use solving status, time cost and memory cost to select the best preconditioners as well as the parameters for a given matrix. The experimental results show that the absolute and relative errors for memory cost

prediction is relatively high. We may find out more efficient prediction methods to lower the absolute and relative errors.

Appendix: Features of Matrices

Name	Type	size	nnzrt	relsymm	diagdomcol	avband	frnorm	strzpiv
ADD20	RUA	2395	0.00229	1.00000	0.99958	0.52505	1.04E+01	0
ADD32	RUA	4960	0.00081	1.00000	0.94194	0.51067	1.57E+00	0
AF23560	RUA	23560	0.00083	0.99744	0.00429	0.02560	1.02E+04	0
ARC130	RUA	130	0.06136	0.78315	0.20769	0.54544	4.89E+05	1
BFW398A	RUA	398	0.02322	0.99021	0.18090	0.54978	8.38E+01	0
BFW398B	RUA	398	0.01837	1.00000	0.33166	0.54179	2.36E-04	0
BFW62A	RUA	62	0.11707	0.97333	0.48387	0.62383	3.06E+01	0
BFW62B	RUA	62	0.08897	1.00000	0.61290	0.55073	5.41E-04	0
BFW782A	RUA	782	0.01229	0.99255	0.11893	0.53558	1.29E+02	0
BFW782B	RUA	782	0.00978	1.00000	0.29540	0.53141	2.54E-04	0
BP 1000	RUA	822	0.00690	0.01030	0.00122	0.28729	1.12E+03	376
BP 1200	RUA	822	0.00699	0.01058	0.00122	0.28431	1.18E+03	366
BP 1400	RUA	822	0.00709	0.01148	0.00122	0.28560	1.13E+03	367
BP 1600	RUA	822	0.00716	0.01198	0.00122	0.27651	1.13E+03	376
BP 200	RUA	822	0.00563	0.00684	0.00122	0.26505	9.93E+02	443
BP 400	RUA	822	0.00596	0.00596	0.00122	0.27644	9.85E+02	414
BP 600	RUA	822	0.00617	0.00671	0.00122	0.28110	1.06E+03	395
BP 800	RUA	822	0.00671	0.00618	0.00122	0.28334	1.08E+03	388
BP 0	RUA	822	0.00485	0.00946	0.00122	0.24690	9.78E+02	499
BWM200	RUA	200	0.01990	1.00000	0.51000	0.50995	8.46E+03	0
BWM2000	RUA	2000	0.00200	1.00000	0.50100	0.50100	2.64E+06	0
CAVITY01	RUA	317	0.07245	0.81029	0.25552	0.22085	9.32E+01	1
CAVITY02	RUA	317	0.05894	0.81029	0.29022	0.27833	1.18E+02	0
CAVITY03	RUA	317	0.07275	0.81029	0.25552	0.22085	2.11E+02	1
CAVITY04	RUA	317	0.05894	0.81029	0.25868	0.27833	3.51E+02	0
CAVITY05	RUA	1182	0.02336	0.90747	0.13621	0.12366	1.92E+02	1
CAVITY06	RUA	1182	0.02124	0.90747	0.17428	0.13830	2.01E+02	0
CAVITY07	RUA	1182	0.02341	0.90747	0.13621	0.12366	2.40E+02	1
CAVITY08	RUA	1182	0.02124	0.90747	0.15482	0.13830	3.00E+02	0
CAVITY09	RUA	1182	0.02341	0.90747	0.13621	0.12366	3.85E+02	1
CAVITY10	RUA	2597	0.01129	0.93885	0.09280	0.08532	2.91E+02	1
CAVITY11	RUA	2597	0.01062	0.93885	0.12399	0.09184	2.97E+02	0
CAVITY12	RUA	2597	0.01131	0.93885	0.09280	0.08532	3.21E+02	1
CAVITY13	RUA	2597	0.01062	0.93885	0.11244	0.09184	3.59E+02	0
CAVITY14	RUA	2597	0.01131	0.93885	0.09280	0.08532	4.15E+02	1
CAVITY15	RUA	2597	0.01062	0.93885	0.10820	0.09184	4.79E+02	0
CAVITY16	RUA	4562	0.00663	0.95434	0.07036	0.06506	3.91E+02	1
CAVITY17	RUA	4562	0.00633	0.95434	0.10215	0.06872	3.90E+02	0
CAVITY18	RUA	4562	0.00663	0.95434	0.07036	0.06506	3.96E+02	1
CAVITY19	RUA	4562	0.00633	0.95434	0.09338	0.06872	4.01E+02	0

Cont. (Features of Matrices)

Name	Type	size	nnzrt	relsymm	diagdomcol	avband	frnorm	strzpiv
CAVITY20	RUA	4562	0.00663	0.95434	0.07036	0.06506	4.13E+02	1
CAVITY21	RUA	4562	0.00633	0.95434	0.08900	0.06872	4.24E+02	0
CAVITY22	RUA	4562	0.00663	0.95434	0.07036	0.06506	4.42E+02	1
CAVITY23	RUA	4562	0.00633	0.95434	0.08724	0.06872	4.59E+02	0
CAVITY24	RUA	4562	0.00663	0.95434	0.07036	0.06506	4.83E+02	1
CAVITY25	RUA	4562	0.00633	0.95434	0.08527	0.06872	5.05E+02	0
CAVITY26	RUA	4562	0.00663	0.95434	0.07036	0.06506	5.33E+02	1
CDDE1	RUA	961	0.00507	1.00000	0.12487	0.06354	1.37E+02	0
CDDE2	RUA	961	0.00507	1.00000	0.06348	0.06354	1.57E+02	0
CDDE3	RUA	961	0.00507	1.00000	0.12487	0.06354	1.36E+02	0
CDDE4	RUA	961	0.00507	1.00000	0.06348	0.06354	1.55E+02	0
CDDE5	RUA	961	0.00507	1.00000	0.12487	0.06354	1.31E+02	0
CDDE6	RUA	961	0.00507	1.00000	0.06348	0.06354	1.52E+02	0
CK104	RUA	104	0.09172	1.00000	0.09615	0.55473	1.11E+01	0
CK400	RUA	400	0.01788	0.99021	0.02500	0.39967	1.53E+01	0
CK656	RUA	656	0.00903	0.99279	0.01524	0.34164	1.81E+01	0
CRY10000	RUA	10000	0.00050	0.99797	0.01970	0.04910	3.42E+05	0
E05R0000	RUA	236	0.10496	1.00000	0.16949	0.29422	9.15E+01	0
E05R0100	RUA	236	0.10496	1.00000	0.09322	0.29422	9.61E+01	0
E05R0200	RUA	236	0.10496	1.00000	0.06780	0.29422	1.11E+02	0
E05R0300	RUA	236	0.10496	1.00000	0.03390	0.29422	1.44E+02	0
E05R0400	RUA	236	0.10496	1.00000	0.02119	0.29422	1.94E+02	0
E05R0500	RUA	236	0.10496	1.00000	0.01695	0.29422	2.50E+02	0
E20R0000	RUA	4241	0.00731	1.00000	0.03773	0.07000	3.89E+02	0
E20R0100	RUA	4241	0.00731	1.00000	0.03655	0.07000	3.90E+02	0
E20R0500	RUA	4241	0.00731	1.00000	0.02452	0.07000	4.23E+02	0
E20R1000	RUA	4241	0.00731	1.00000	0.01863	0.07000	5.28E+02	0
E20R2000	RUA	4241	0.00731	1.00000	0.00990	0.07000	8.81E+02	0
E20R3000	RUA	4241	0.00731	1.00000	0.00755	0.07000	1.35E+03	0
E20R4000	RUA	4241	0.00731	1.00000	0.00566	0.07000	1.86E+03	0
E20R5000	RUA	4241	0.00731	1.00000	0.00495	0.07000	2.40E+03	0
E30R0000	RUA	9661	0.00328	1.00000	0.02484	0.04628	5.87E+02	0
E30R0100	RUA	9661	0.00328	1.00000	0.02464	0.04628	5.88E+02	0
E30R0500	RUA	9661	0.00328	1.00000	0.01749	0.04628	6.10E+02	0
E30R1000	RUA	9661	0.00328	1.00000	0.01522	0.04628	6.81E+02	0
E30R2000	RUA	9661	0.00328	1.00000	0.01149	0.04628	9.38E+02	0
E30R3000	RUA	9661	0.00328	1.00000	0.00880	0.04628	1.29E+03	0
E30R4000	RUA	9661	0.00328	1.00000	0.00745	0.04628	1.70E+03	0
E30R5000	RUA	9661	0.00328	1.00000	0.00652	0.04628	2.15E+03	0
E40R0000	RUA	17281	0.00185	1.00000	0.01852	0.03456	7.85E+02	0

Cont. (Features of Matrices)

Name	Type	size	nnzrt	relsymm	diagdomcol	avband	frnorm	strzpiv
E40R0100	RUA	17281	0.00185	1.00000	0.01840	0.03456	7.86E+02	0
E40R0500	RUA	17281	0.00185	1.00000	0.01371	0.03456	8.02E+02	0
E40R1000	RUA	17281	0.00185	1.00000	0.01233	0.03456	8.57E+02	0
E40R2000	RUA	17281	0.00185	1.00000	0.01018	0.03456	1.06E+03	0
E40R3000	RUA	17281	0.00185	1.00000	0.00851	0.03456	1.36E+03	0
E40R4000	RUA	17281	0.00185	1.00000	0.00712	0.03456	1.70E+03	0
E40R5000	RUA	17281	0.00185	1.00000	0.00637	0.03456	2.08E+03	0
FIDAP001	RUA	216	0.09300	1.00000	0.45370	0.28946	6.83E-01	0
FIDAP002	RUA	441	0.13796	1.00000	0.00000	0.27490	9.76E+08	0
FIDAP003	RUA	1821	0.01588	1.00000	0.00055	0.08055	1.40E+08	0
FIDAP004	RUA	1601	0.01242	1.00000	0.00500	0.09162	1.61E+01	0
FIDAP005	RUA	27	0.38272	1.00000	0.00000	0.79424	9.09E+06	0
FIDAP006	RUA	1651	0.01796	1.00000	0.01878	0.06806	1.80E+03	0
FIDAP007	RUA	1633	0.01746	1.00000	0.02449	0.14341	1.13E+10	0
FIDAP008	RUA	3096	0.00947	1.00000	0.00065	0.05946	4.24E+08	1
FIDAP009	RUA	3363	0.00879	1.00000	0.00000	0.03670	2.96E+10	0
FIDAP010	RUA	2410	0.00944	1.00000	0.03610	0.05898	2.16E+08	0
FIDAP011	RUA	16614	0.00395	1.00000	0.00000	0.04550	3.63E+09	0
FIDAP012	RUA	3973	0.00501	1.00000	0.07903	0.03507	1.83E+03	0
FIDAP013	RUA	2568	0.01147	1.00000	0.00000	0.05385	5.13E+10	0
FIDAP014	RUA	3251	0.00622	1.00000	0.21255	0.07485	1.07E+08	0
FIDAP015	RUA	6867	0.00204	1.00000	0.03641	0.01925	6.89E+10	0
FIDAP018	RUA	5773	0.00208	1.00000	0.05889	0.04508	6.91E+08	1
FIDAP019	RUA	12005	0.00180	1.00000	0.05789	0.01983	3.25E+08	0
FIDAP020	RUA	2203	0.01389	1.00000	0.24875	0.07082	3.66E+00	0
FIDAP021	RUA	656	0.04406	1.00000	0.34299	0.16415	2.21E+00	0
FIDAP022	RUA	839	0.03175	1.00000	0.02145	0.07125	5.70E+01	0
FIDAP023	RUA	1409	0.02143	1.00000	0.01987	0.08572	1.35E+06	0
FIDAP024	RUA	2283	0.00919	1.00000	0.04205	0.06247	7.19E+02	0
FIDAP025	RUA	848	0.03374	1.00000	0.04953	0.13744	1.66E+00	0
FIDAP026	RUA	2163	0.01586	1.00000	0.04854	0.24577	3.76E+02	2
FIDAP027	RUA	974	0.03964	1.00000	0.32033	0.15399	9.51E-01	0
FIDAP028	RUA	2603	0.01135	1.00000	0.04456	0.05596	5.84E+02	0
FIDAP029	RUA	2870	0.00288	1.00000	0.26028	0.02739	1.35E+01	0
FIDAP031	RUA	3909	0.00597	1.00000	0.06754	0.04958	8.85E+01	2
FIDAP032	RUA	1159	0.00822	1.00000	0.06903	0.06013	2.98E+02	1
FIDAP033	RUA	1733	0.00676	1.00000	0.08482	0.08749	2.43E+11	0
FIDAP035	RUA	19716	0.00056	1.00000	0.03971	0.01221	3.24E+08	1
FIDAP036	RUA	3079	0.00560	1.00000	0.14745	0.06367	3.21E+03	1
FIDAP037	RUA	3565	0.00532	1.00000	0.59888	0.07984	1.47E+03	0

Cont. (Features of Matrices)

Name	Type	size	nnzrt	relsymm	diagdomcol	avband	frnorm	strzpiv
FIDAPM02	RUA	537	0.06639	1.00000	0.15642	0.25694	5.39E+01	0
FIDAPM03	RUA	2532	0.00775	1.00000	0.00000	0.07276	3.95E+01	0
FIDAPM05	RUA	42	0.28628	1.00000	0.09524	0.76190	1.81E+01	0
FIDAPM07	RUA	2065	0.01060	1.00000	0.08523	0.13257	2.54E+02	0
FIDAPM08	RUA	3876	0.00578	1.00000	0.02838	0.05485	9.08E+02	1
FIDAPM09	RUA	4683	0.00427	1.00000	0.00256	0.03294	5.44E+03	0
FIDAPM10	RUA	3046	0.00573	1.00000	0.02725	0.05398	9.78E+00	0
FIDAPM11	RUA	22294	0.00124	1.00000	0.00022	0.03979	2.60E+01	0
FIDAPM13	RUA	3549	0.00564	1.00000	0.00113	0.04864	3.24E+01	0
FIDAPM15	RUA	9287	0.00111	1.00000	0.02735	0.01664	2.82E+01	9
FIDAPM29	RUA	13668	0.00098	1.00000	0.29814	0.02586	1.78E+02	27
FIDAPM33	RUA	2353	0.00418	1.00000	0.17382	0.07658	4.94E+02	40
FIDAPM37	RUA	9152	0.00914	1.00000	0.00350	0.15220	1.05E+08	0
FS 183 1	RUA	183	0.02980	0.51731	0.55191	0.60623	1.13E+09	5
FS 183 3	RUA	183	0.03192	0.51731	0.54098	0.60623	1.15E+09	6
FS 183 4	RUA	183	0.03192	0.51731	0.56831	0.60623	1.12E+09	6
FS 183 6	RUA	183	0.02986	0.51731	0.56831	0.60623	1.18E+09	5
FS 541 1	RUA	541	0.01463	0.72275	0.99815	0.67334	7.15E+01	1
FS 541 2	RUA	541	0.01463	0.72275	0.43808	0.67334	1.70E+06	1
FS 541 3	RUA	541	0.01463	0.72275	0.37523	0.67334	1.27E+07	1
FS 541 4	RUA	541	0.01460	0.72275	0.42699	0.67334	1.58E+06	1
FS 680 1	RUA	680	0.00472	0.63643	0.94265	0.24932	1.21E+14	5
FS 680 2	RUA	680	0.00524	0.63643	1.00000	0.24932	1.21E+14	5
FS 680 3	RUA	680	0.00534	0.63643	1.00000	0.24932	1.21E+14	3
FS 760 1	RUA	760	0.00994	0.69545	0.98158	0.69779	4.54E+08	20
FS 760 2	RUA	760	0.00994	0.69545	0.50526	0.69779	4.54E+08	20
FS 760 3	RUA	760	0.01007	0.69545	0.15395	0.69779	4.54E+08	20
GEMAT11	RUA	4929	0.00136	0.00172	0.00041	0.12312	8.25E+02	2322
GEMAT12	RUA	4929	0.00136	0.00172	0.00020	0.12397	9.71E+02	2342
GRE 1107	RUA	1107	0.00462	0.19544	0.06504	0.27947	1.79E+01	0
GRE 216A	RUA	216	0.01740	0.24658	0.22222	0.15745	7.81E+00	0
GRE 216B	RUA	216	0.01740	0.24658	0.16667	0.15745	1.43E+01	0
GRE 115	RUA	115	0.03183	0.45843	0.66957	0.49921	7.25E+00	0
GRE 185	RUA	185	0.02849	0.52239	0.14595	0.32386	7.98E+00	0
GRE 343	RUA	343	0.01113	0.23902	0.16910	0.13558	9.70E+00	0
GRE 512	RUA	512	0.00754	0.23358	0.13281	0.11912	1.17E+01	0
HOR 131	RUA	434	0.02220	1.00000	0.29724	0.57611	2.10E+00	0
IMPCOL A	RUA	207	0.01335	0.03846	0.01449	0.09870	2.35E+03	121
IMPCOL B	RUA	59	0.07785	0.14423	0.16949	0.30336	1.39E+01	22
IMPCOL C	RUA	137	0.02131	0.16302	0.10219	0.13219	1.45E+02	29

Cont. (Features of Matrices)

Name	Type	size	nnzrt	relsymm	diagdomcol	avband	frnorm	strzpiv
IMPCOL D	RUA	425	0.00695	0.08140	0.00000	0.05805	1.40E+02	91
IMPCOL E	RUA	225	0.02574	0.14220	0.01333	0.13845	1.58E+04	67
JPWH 991	RUA	991	0.00614	0.94691	0.89304	0.14599	1.94E+02	58
LNS 3937	RUA	3937	0.00164	0.86862	0.07798	0.44959	1.44E+12	185
LNS 131	RUA	131	0.03123	0.76493	0.32061	0.35802	1.52E+10	20
LNS 511	RUA	511	0.01071	0.82976	0.18004	0.41503	1.06E+11	50
LNSP3937	RUA	3937	0.00164	0.86862	0.07798	0.03313	1.44E+12	200
LNSP 131	RUA	131	0.03123	0.76493	0.32061	0.19911	1.52E+10	24
LNSP 511	RUA	511	0.01071	0.82976	0.18004	0.13380	1.06E+11	59
LOP163	RUA	163	0.03519	0.54652	0.53988	0.12439	8.74E+00	0
MBEACXC	RUA	496	0.20291	0.32179	0.02823	0.95452	4.78E+00	80
MBEAFLW	RUA	496	0.20291	0.32179	0.02823	0.95452	4.09E+04	80
MBEAUSE	RUA	496	0.16691	0.22668	0.02016	0.95096	4.24E+04	118
MCCA	RUA	180	0.08207	0.66867	0.38333	0.16991	2.32E+19	38
MCFE	RUA	765	0.04166	0.70884	0.65882	0.14402	2.01E+17	40
MEMPLUS	RUA	17758	0.00031	1.00000	0.87859	0.56544	7.61E+00	0
MHD3200A	RUA	3200	0.00664	0.77172	0.00219	0.01186	1.83E+05	1
MHD3200B	RUA	3200	0.00179	1.00000	0.75438	0.00898	5.19E+00	0
MHD416A	RUA	416	0.04948	0.77190	0.01683	0.08848	2.87E+03	1
MHD416B	RUA	416	0.01336	1.00000	0.75240	0.06666	5.18E+00	0
MHD4800A	RUA	4800	0.00444	0.77189	0.00146	0.00792	4.12E+05	1
MHD4800B	RUA	4800	0.00119	1.00000	0.75458	0.00599	5.19E+00	0
NNC1374	RUA	1374	0.00455	0.83546	0.00000	0.05637	9.61E+03	0
NNC261	RUA	261	0.02202	0.84133	0.00000	0.13950	3.96E+03	0
NNC666	RUA	666	0.00909	0.83630	0.00000	0.08367	6.52E+03	0
ODEP400A	RUA	400	0.00751	0.99750	0.99000	0.01246	4.91E+01	0
ORSIRR 1	RUA	1030	0.00646	1.00000	0.54175	0.22305	1.85E+06	0
ORSIRR 2	RUA	886	0.00761	1.00000	0.54853	0.27065	1.55E+06	0
ORSREG 1	RUA	2205	0.00291	1.00000	0.53333	0.32409	8.54E+05	0
PDE225	RUA	225	0.02104	1.00000	0.30222	0.12944	7.61E+01	0
PDE2961	RUA	2961	0.00166	1.00000	0.85849	0.03159	2.23E+02	0
PDE900	RUA	900	0.00541	1.00000	0.67333	0.06563	1.46E+02	0
PORES 1	RUA	30	0.20000	0.68889	0.50000	0.42667	3.75E+07	0
PORES 2	RUA	1224	0.00642	0.66140	0.48121	0.15560	1.50E+08	0
PORES 3	RUA	532	0.01227	0.78181	0.71805	0.16377	6.63E+05	0
PSMIGR 1	RUA	3140	0.05509	0.48166	0.60159	0.92605	3.54E+06	1
PSMIGR 2	RUA	3140	0.05477	0.47865	0.00000	0.92604	1.50E+01	5
PSMIGR 3	RUA	3140	0.05509	0.48166	0.82707	0.92605	3.62E+01	1
QH1484	RUA	1484	0.00277	1.00000	0.01348	0.54609	4.71E+16	88
QH768	RUA	768	0.00497	0.94479	0.05729	0.40259	2.48E+13	84

Cont. (Features of Matrices)

Name	Type	size	nnzrt	relsymm	diagdomcol	avband	frnorm	strzpiv
QH882	RUA	882	0.00431	0.93739	0.06349	0.37531	2.26E+13	103
RDB1250	RUA	1250	0.00467	1.00000	0.57680	0.07772	2.74E+03	0
RDB1250L	RUA	1250	0.00467	1.00000	0.50000	0.07772	7.43E+02	0
RDB200	RUA	200	0.02800	1.00000	0.50000	0.18685	2.24E+02	0
RDB200L	RUA	200	0.02800	1.00000	0.50000	0.18685	1.09E+02	0
RDB2048	RUA	2048	0.00287	1.00000	0.12109	0.06109	5.61E+03	0
RDB2048L	RUA	2048	0.00287	1.00000	0.50000	0.06109	1.46E+03	0
RDB3200L	RUA	3200	0.00184	1.00000	0.50125	0.04909	2.76E+03	0
RDB450	RUA	450	0.01274	1.00000	0.50000	0.12723	6.46E+02	0
RDB450L	RUA	450	0.01274	1.00000	0.50000	0.12723	2.19E+02	0
RDB800L	RUA	800	0.00725	1.00000	0.09500	0.09649	4.19E+02	0
RW136	RUA	136	0.02590	0.44259	0.00000	0.14852	7.24E+00	2
RW496	RUA	496	0.00756	0.46907	0.00000	0.08088	1.34E+01	2
RW5151	RUA	5151	0.00076	0.49022	0.00000	0.02589	4.19E+01	2
SAYLR1	RUA	238	0.01991	1.00000	0.73950	0.11539	8.86E+08	0
SAYLR3	RUA	1000	0.00375	1.00000	0.52900	0.07049	4.31E+01	0
SAYLR4	RUA	3564	0.00176	1.00000	0.98737	0.10608	4.38E+05	0
SHERMAN1	RUA	1000	0.00375	1.00000	0.52900	0.07049	4.32E+01	0
SHERMAN2	RUA	1080	0.01980	0.68624	0.06852	0.32318	7.00E+09	133
SHERMAN3	RUA	5005	0.00080	1.00000	1.00000	0.07744	1.36E+07	0
SHERMAN4	RUA	1104	0.00311	1.00000	0.99728	0.22222	5.04E+02	0
SHERMAN5	RUA	3312	0.00190	0.78026	0.67029	0.22188	1.40E+04	72
SHL 200	RUA	663	0.00393	0.00348	0.00452	0.36746	9.43E+03	503
SHL 400	RUA	663	0.00389	0.00175	0.00151	0.35902	9.25E+03	485
SHL 0	RUA	663	0.00384	0.00296	0.00452	0.35110	9.57E+03	532
STEAM1	RUA	240	0.03903	1.00000	0.66667	0.57797	8.41E+07	0
STEAM2	RUA	600	0.01572	1.00000	0.50000	0.54658	5.27E+10	2
STEAM3	RUA	80	0.04906	1.00000	0.50000	0.55000	1.85E+10	0
STR 200	RUA	363	0.02328	0.01173	0.00275	0.10846	3.72E+01	229
STR 400	RUA	363	0.02396	0.02027	0.00275	0.10823	3.95E+01	220
STR 600	RUA	363	0.02488	0.02013	0.00275	0.12223	4.83E+01	205
STR 0	RUA	363	0.01862	0.00570	0.00275	0.08519	3.18E+01	238
TOLS1090	RUA	1090	0.00298	0.48562	0.73211	0.18138	1.23E+07	218
TOLS2000	RUA	2000	0.00130	0.54282	0.76100	0.17149	5.40E+07	400
TOLS340	RUA	340	0.01900	0.37432	0.59706	0.23309	7.97E+05	68
TOLS4000	RUA	4000	0.00055	0.59358	0.77875	0.16570	2.98E+08	800
TOLS90	RUA	90	0.21556	0.29897	0.05556	0.50556	3.96E+04	18
UTM1700a	RUA	1700	0.00737	0.59213	0.32059	0.16912	4.12E+01	96
UTM1700b	RUA	1700	0.00744	0.56097	0.21353	0.16842	4.12E+01	86
UTM300	RUA	300	0.03506	0.51601	0.28000	0.22630	1.73E+01	44

Cont. (Features of Matrices)

Name	Type	size	nnzrt	relsymm	diagdomcol	avband	frnorm	strzpiv
UTM3060	RUA	3060	0.00451	0.55914	0.17582	0.09895	5.53E+01	88
UTM5940	RUA	5940	0.00238	0.56242	0.15572	0.10125	7.71E+01	165
WATT 1	RUA	1856	0.00330	0.98873	0.55819	0.06501	1.13E+01	64
WATT 2	RUA	1856	0.00335	0.98355	0.53879	0.06724	1.38E+01	64
WEST0067	RUA	67	0.06549	0.04082	0.00000	0.54266	1.31E+01	25
WEST0132	RUA	132	0.02370	0.03382	0.00000	0.12052	3.57E+05	73
WEST0156	RUA	156	0.01488	0.00000	0.00000	0.13400	1.95E+07	82
WEST0167	RUA	167	0.01814	0.04339	0.01198	0.13819	6.39E+05	83
WEST0381	RUA	381	0.01470	0.00603	0.00000	0.62436	3.05E+03	169
WEST0479	RUA	479	0.00823	0.01780	0.00835	0.14197	7.10E+05	262
WEST0497	RUA	497	0.00697	0.01042	0.00000	0.09797	1.22E+06	262
WEST0655	RUA	655	0.00655	0.00736	0.00458	0.12015	7.10E+05	437
WEST0989	RUA	989	0.00360	0.01951	0.00000	0.09840	1.27E+06	609
WEST1505	RUA	1505	0.00239	0.00202	0.00000	0.09941	1.56E+06	967
WEST2021	RUA	2021	0.00179	0.00394	0.00000	0.09994	1.80E+06	1313
1138 BUS	RSA	1138	0.00313	1.00000	0.95958	0.06069	1.26E+05	0
494 BUS	RSA	494	0.00683	1.00000	0.92308	0.15735	5.75E+04	0
662 BUS	RSA	662	0.00565	1.00000	0.93051	0.10773	7.36E+03	0
685 BUS	RSA	685	0.00692	1.00000	0.95328	0.07078	4.01E+04	0
BCSSTK01	RSA	48	0.17361	1.00000	0.54167	0.35677	7.52E+09	0
BCSSTK02	RSA	66	1.00000	1.00000	0.50000	0.50758	5.29E+04	0
BCSSTK03	RSA	112	0.05102	1.00000	0.50000	0.05230	3.47E+11	0
BCSSTK04	RSA	132	0.20937	1.00000	0.56061	0.21625	4.19E+07	0
BCSSTK05	RSA	153	0.10351	1.00000	0.49673	0.10359	2.21E+07	0
BCSSTK06	RSA	420	0.04456	1.00000	0.63810	0.08576	2.13E+10	0
BCSSTK07	RSA	420	0.04456	1.00000	0.63810	0.08576	2.13E+10	0
BCSSTK08	RSA	1074	0.01124	1.00000	0.70112	0.16143	1.01E+11	0
BCSSTK09	RSA	1083	0.01572	1.00000	0.38504	0.05402	8.57E+08	0
BCSSTK10	RSA	1086	0.01871	1.00000	0.15101	0.03233	2.97E+08	0
BCSSTK11	RSA	1473	0.01578	1.00000	0.56280	0.06218	4.67E+09	0
BCSSTK12	RSA	1473	0.01578	1.00000	0.56280	0.06218	4.67E+09	0
BCSSTK13	RSA	2003	0.02091	1.00000	0.32152	0.11463	7.54E+12	0
BCSSTK14	RSA	1806	0.01945	1.00000	0.57807	0.05855	6.47E+10	0
BCSSTK15	RSA	3948	0.00756	1.00000	0.50152	0.06887	5.70E+10	0
BCSSTK16	RSA	4884	0.01217	1.00000	0.21785	0.02535	6.01E+10	0
BCSSTK17	RSA	10974	0.00356	1.00000	0.26654	0.02436	4.83E+10	0
BCSSTK18	RSA	11948	0.00104	1.00000	0.68806	0.04600	2.36E+11	0
BCSSTK19	RSA	817	0.01027	1.00000	0.65728	0.10831	9.62E+14	0
BCSSTK20	RSA	485	0.01333	1.00000	0.53196	0.01968	2.92E+16	0
BCSSTK21	RSA	3600	0.00205	1.00000	1.00000	0.03283	2.96E+09	0

Cont. (Features of Matrices)

Name	Type	size	nnzrt	relsymm	diagdomcol	avband	frnorm	strzpiv
BCSSTK22	RSA	138	0.03655	1.00000	0.84058	0.10744	2.06E+07	0
BCSSTK23	RSA	3134	0.00460	1.00000	0.65699	0.10874	9.11E+16	0
BCSSTK24	RSA	3562	0.01260	1.00000	0.48119	0.20894	1.39E+14	0
BCSSTK25	RSA	15439	0.00106	1.00000	0.55625	0.01102	3.15E+15	0
BCSSTK26	RSA	1922	0.00821	1.00000	0.41103	0.06356	3.95E+11	0
BCSSTK27	RSA	1224	0.03746	1.00000	0.53595	0.03368	2.07E+07	0
BCSSTK28	RSA	4410	0.01126	1.00000	0.04399	0.04172	1.05E+09	0
BCSSTM02	RSA	66	0.01515	1.00000	1.00000	0.01515	8.22E-01	0
BCSSTM05	RSA	153	0.00654	1.00000	1.00000	0.00654	4.67E+00	0
BCSSTM06	RSA	420	0.00238	1.00000	1.00000	0.00238	6.74E+04	0
BCSSTM07	RSA	420	0.04111	1.00000	0.57143	0.08573	1.80E+04	0
BCSSTM08	RSA	1074	0.00093	1.00000	1.00000	0.00093	2.29E+06	0
BCSSTM09	RSA	1083	0.00092	1.00000	1.00000	0.00092	4.92E-03	0
BCSSTM10	RSA	1086	0.01873	1.00000	0.40792	0.03233	2.96E+05	0
BCSSTM11	RSA	1473	0.00068	1.00000	1.00000	0.00068	3.01E+02	0
BCSSTM12	RSA	1473	0.00906	1.00000	0.59335	0.06178	8.24E+01	0
BCSSTM13	RSA	2003	0.00528	1.00000	0.85372	0.02938	8.94E+02	762
BCSSTM19	RSA	817	0.00122	1.00000	1.00000	0.00122	1.08E+08	0
BCSSTM20	RSA	485	0.00206	1.00000	1.00000	0.00206	1.01E+08	0
BCSSTM21	RSA	3600	0.00028	1.00000	1.00000	0.00028	6.23E-03	0
BCSSTM22	RSA	138	0.00725	1.00000	1.00000	0.00725	3.67E-02	0
BCSSTM23	RSA	3134	0.00032	1.00000	1.00000	0.00032	3.73E+07	0
BCSSTM24	RSA	3562	0.00028	1.00000	1.00000	0.00028	6.27E+06	0
BCSSTM25	RSA	15439	0.00006	1.00000	1.00000	0.00006	8.58E+09	0
BCSSTM26	RSA	1922	0.00052	1.00000	1.00000	0.00052	1.15E+01	0
BCSSTM27	RSA	1224	0.03746	1.00000	0.14052	0.03368	3.08E+04	0
GR 30 30	RSA	900	0.00956	1.00000	1.00000	0.03441	2.54E+02	0
LUND A	RSA	147	0.11333	1.00000	0.68027	0.13749	1.39E+09	0
LUND B	RSA	147	0.11296	1.00000	0.38095	0.13749	2.88E+04	0
NOS1	RSA	237	0.01811	1.00000	0.67089	0.01811	1.34E+10	0
NOS2	RSA	957	0.00452	1.00000	0.66771	0.00452	1.72E+12	0
NOS3	RSA	960	0.01719	1.00000	0.97604	0.04347	7.71E+03	0
NOS4	RSA	100	0.05940	1.00000	0.91000	0.09450	4.19E+00	0
NOS5	RSA	468	0.02361	1.00000	0.90598	0.11697	4.97E+06	0
NOS6	RSA	675	0.00714	1.00000	1.00000	0.03710	4.50E+07	0
NOS7	RSA	729	0.00869	1.00000	1.00000	0.10137	2.97E+07	0
PLAT1919	RSA	1919	0.00880	1.00000	0.66649	0.34597	2.22E+01	0
PLAT362	RSA	362	0.04415	1.00000	0.65470	0.35991	5.00E+00	0
ZENIOS	RSA	2873	0.00016	1.00000	0.92691	0.13334	9.31E+00	2873

Bibliography

[1] O. Axelsson. *Iterative Solution Methods.* Cambridge Univ. Press, Cambridge, 1994.

[2] R. Barrett, M. Berry, and *et al. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, 1993.

[3] K. P. Bennett and C. Campbell. Support vector machines: Hype or Hallelujah? *SIGKDD Explorations*, 2(2):1-3, 2000.

[4] M. Benzi. Preconditioning techniques for large linear systems: survey. *J. Comput. Phy.*, 182:418–477, 2002.

[5] M. W. Berry, Z. Drmac, and E. R. Jessup. Matrix, vector space, and information retrieval. *SIAM Rev.*, 41:335–362, 1999.

[6] P. S. Bradley and O. L. Mangasarian. Feature selection via concave minimization and support vector machines. In *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[7] C. Burges. *A tutorial on support vector machine for pattern recognition.* Kluwer Academic Publishers, 1998.

[8] C. Campbell. Kernel methods: A survey of current techniques. *Neurocomputing.*, 48:63–84, 2002.

[9] G. F. Carey, R. McLay, G. Bicken, B. Barth, S. Swift, and A. Ardelea. Parallel finite element solution of three-dimensional Rayleigh-Ḃenard-Marangoni flows. *Int. J. Numer. Meth. Fluids*, 31:37–52, 1999.

[10] E. Chow and Y. Saad. Experimental study of ILU preconditioners for indefinite matrices, *J. Comput. Appl. Math.*, 86:387–414, 1997.

[11] O. Dahl and S. Ø. Wille. An ILU preconditioner with coupled node fill in for iterative solution of the mixed finite element formulation of the 2D and 3D Navier-Stokes equations. *Int. J. Numer. Meth. Fluids*, 15:525–544, 1992.

[12] T. Davis. University of Florida sparse matrix collection, http://www.cise.ufl.edu/~davis/sparse. *NA Digest*, 97(23), June 7, 1997.

[13] E. F. D'Azevedo, F. A. Forsyth, and W. P. Tang. Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems. *SIAM J. Matrix Anal. Appl.*, 13:944–961, 1992.

[14] S. Deewester, S. Dumais, and *et al.* Indexing by latent semantic analysis, *J. Amer. Soc. Infor. Sci.*, **41**:391–407, 1990.

[15] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143-175, 2001.

[16] J. J. Dongarra, J. R. Bunch, C. B. Moler, and *et al. LINPACK Users' Guide.* SIAM, Philadelphia, PA, 1979.

[17] I. S. Duff and G.A. Meurant. The effect of reordering on preconditioned conjugate gradients. *BIT*, 29:635–657, 1989.

[18] H. C. Elman. A stability analysis of incomplete LU factorization. *Math. Comp.*, 47:191–217, 1986.

[19] J. Gao and J. Zhang. Sparsification strategies in latent semantic indexing, in *Proceedings of the 2003 Text Mining Workshop*, M. W. Berry and W. M. Pottenger, (ed.), pp. 93–103, San Francisco, CA, May 3, 2003.

[20] G. H. Golub and C. F. van Loan. *Matrix Computation.* John Hopkins Univ. Press, Baltimore, 3rd Edition, 1996.

[21] G. H. Golub and H. A. van der Vorst. Closer to the solution: iterative linear solvers. In I. S. Duff and G. A. Watson, editors, *The State of the Art in Numerical Analysis*, pages 63–92, Oxford, 1997. Clarendon Press.

[22] A. Greenbaum. *Iterative Methods for Solving Linear Systems.* SIAM, Philadelphia, PA, 1997.

[23] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[24] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer-Verlag, New York, 2001.

[25] N. J. Higham. Fortran codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. *ACM Trans. Math. Soft.*, 14:381–396, 1988.

[26] E. N. Houstis, J. R. Rice, and R. Vichnevetsky. *Intelligent Mathematical Software Systems*, Proceedings of the first IMACS/IFAC International Conference on Expert Systems for Numerical Computing, North-Holland, New York, 1990.

[27] C. W. Hsu and C. J. Lin. A comparison of methods for multi-class support vector machines. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, 2001

[28] R. Jin and H. Liu. Robust feature induction for support vector machines. In *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004.

[29] T. Joachims. Making large-scale SVM learning practical. *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf, C. Burges and A. Smola (ed.), MIT-Press, 1999.

[30] S. Kaniel. Estimates for some computational techniques in linear algebra. *Math. Comput.*, 20:369–378, 1966.

[31] N. Kang, J. Zhang, and E. S. Carlson. Parallel simulation of anisotropic diffusion with human brain DT-MRI data. *Computers and Structures*, 82(28):2389–2399, 2004.

[32] S. Karaa, J. Zhang, and C. C. Douglas. Preconditioned multigrid simulation of an axisymmetric laminar diffusion flame, *Math. Comput. Model.*, 38:269–279, 2003.

[33] D.S. Kershaw. On the problem of unstable pivots in the incomplete LU-conjugate gradient method, *J. Comput. Phys.*, 38:114–123, 1980.

[34] J. Lee, J. Zhang, and C.-C. Lu. Performance of preconditioned Krylov iterative methods for solving hybrid integral equations in electromagnetics, *J. of Applied Comput. Electromagnetics Society*, 18:54–61, 2003.

[35] Y. Li, S. Gong, and H. Liddell. Support vector regression and classification based multiview face detection and recognition. In *Proc. of the IEEE International Conference on Automatic Face and Gesture Recognition (FGR'00).*, Grenoble, France, 2000.

[36] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the 5th Symposium on Math, Statistics, and Probability.*, Univ. of California Press, p. 281-297, 1967.

[37] http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml.

[38] http://math.nist.gov/MatrixMarket.

[39] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comput.*, 31:148–162, 1977.

[40] D. Mladenić, J. Brank, M. Grobelnik, and N. Milic-Frayling. Feature selection using linear classifier weights: interaction with classification models. In *Proceedings of SIGIR'04*, Sheffield, UK, July, 2004.

[41] L. C. Molina, L. Belanche, and A. Nebot. Feature selection algorithms: A survey and experimental evaluation. In *Proceedings of 2002 IEEE International Conference on Data Mining (ICDM'02)*, Maebashi City, Japan, 2002.

[42] A. L. Pardhanani and G. F. Carey. Time-integration and iterative techniques for semiconductor diffusion modeling. *IEEE J. Technology Computer Aided Design*, SISPAD97:1–12, 1997. (in html format).

[43] J. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin DAGS for multiclass classification. *Advances in Neural Information Processing Systems*, 12 ed. S.A. Solla, T.K. Leen and K.-R. Muller, MIT Press, 2000.

[44] A. Rakotomamonjy. Variable selection using SVM-based criteria. *Journal of Machine Learning Research*, 3:1357–1370, 2003.

[45] W. B. Richardson, G. F. Carey, and B. J. Mulvaney. Modeling phosphorus diffusion in three dimensions. *IEEE Trans. Computer Aided Design*, 11(4):487–496, 1992.

[46] R. Rifkin and A. Klautau. In Defense of One-Vs-All Classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

[47] Y. Saad. ILUT: a dual threshold incomplete LU factorization, *Numer. Linear Algebra Appl.*, 14:387–402, 1994.

[48] Y. Saad. *Iterative Methods for Sparse Linear Systems*, PWS, New York, 1996.

[49] L. Shih, Y. Chang, J. Rennie, and *et al.* Not too hot, not too cold: The Bundled-SVM is just right! In *Workshop on Text Learning (TextML-2002)*, Sydney, Australia, 2002.

[50] A. J. Smola and B. Scholköpf. A tutorial on support vector regression. *NeuroCOLT Technical Report Series*, NC2-TR-1998-030, 1998.

[51] http://www-users.cs.umn.edu/šaad/software/SPARSKIT/sparskit.html

[52] T. B. Trafalis and H. Ince. Support vector machine for regression and applications to financial forecasting. In *Proceedings of IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)*, Como, Italy, 2000.

[53] V. N. Vapnik. *Statistical Learning Theory.* John Wiley and Sons, New York, 1998.

[54] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.

[55] V. Vural and J. G. Dy. A hierarchical method for multi-class support vector machines. In *Proc. of the Twenty-first International Conference on Machine Learning.*, Banff, Alberta, Canada, July 2004

[56] S. Xu, E. Lee, and J. Zhang. An interim analysis report on preconditioners and matrices. Technical Report No. 388-03, Department of Computer Science, University of Kentucky, Lexington, KY, 2003.

[57] S. Xu, E. Lee, and J. Zhang. Designing and building an intelligent preconditioner recommendation system (a progress report). In *Abstracts of the 2003 International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Scientific and Industrial Applications*, Napa, CA, 2003.

[58] S. Xu and J. Zhang. Matrix condition number prediction with SVM regression and feature selection, in *Proceedings of the Fifth SIAM International Conference on Data Mining*, Newport Beach, CA, April, 2005.

[59] S. Xu and J. Zhang. A Data Mining Approach to Matrix Preconditioning Problem, in *Proceedings of the Eighth Workshop on Mining Scientific and Engineering Datasets (MSD05), in conjunction with the Fifth SIAM International Conference on Data Mining*, Newport Beach, CA, April, 2005.

[60] H. Yang, L. Chan, and I. King. Support vector machine regression for volatile stock market prediction. In *Proceedings of the Third International Conference on Intelligent Data Engineering and Automated Learning*, 2002.

[61] J. Zhang. Preconditioned Krylov subspace methods for solving nonsymmetric matrices from CFD applications. *Comput. Methods Appl. Mech. Engrg.*, 189(3):825–840, 2000.

[62] J. Zhang. A multilevel dual reordering strategy for robust incomplete LU factorization of indefinite matrices, *SIAM J. Matrix Anal. Appl.*, 22:925–947, 2001.

[63] J. Zhang. Performance of ILU preconditioners for stationary 3D Navier-Stokes simulation and the matrix mining project. In *Proceedings of the 2001 International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Scientific and Industrial Applications*, p. 89–90, Tahoe City, CA, 2001.

[64] J. Zhang, G. F. Carey, R. McLay, and B. Barth. Performance of ILU preconditioner in stationary Navier-Stokes simulation, Preprint, Department of Computer Science, University of Kentucky, Lexington, KY, 2001.

[65] J. Zhang, A. L. Pardhanani, and G. F. Carey. Performance of adaptive dual-dropping ILUT preconditioners in semiconductor dopant diffusion simulation. *Int. J. Numer. Modeling: Electronic Networks, Devices and Fields*, 15(2):147–167, 2002.

[66] Y. Saad and J. Zhang. Enhanced multi-level block ILU preconditioning strategies for general sparse linear systems, *J. Appl. Comput. Math.*, 130:99–118, 2001.

Vita

Personal Data:

Name: Shuting Xu

Date of Birth: 05/04/1973

Place of Birth: Jinan, China

Educational Background:

- Doctor of Engineering in Computer Science and Engineering, Shanghai Jiaotong University, China, 2001.
- Bachelor of Engineering in Computer Engineering, Shandong University, China, 1994.

Professional Experience:

- Research Assistant, 01/2003 - present. Department of Computer Science, University of Kentucky.
- Teaching Assistant, 01/2002 - 01/2003. Department of Computer Science, University of Kentucky.
- Software Engineer, 07/2001 - 12/2001. Shanghai Bell Company Ltd., Shanghai, China.
- Research Assistant, 09/1996 - 07/2001. Department of Computer Science and Engineering, Shanghai Jiaotong University.
- Instructor, 09/1994 - 07/1996. Department of Computer Science, Shandong University, China.

Awards:

- International Conference On Preconditioning Techniques For Large Sparse Matrix Problems In Scientific And Industrial Applications Travel Support, 2005.
- Student Travel Support from Graduate School Fellowship of the University of Kentucky, 2004-2005.
- Thaddeus B. Curtz Memorial Scholarship Award, 2004.
- The Commonwealth Research Award of the University of Kentucky, 2004.
- SIAM Conference on Applied Linear Algebra Travel Awards, 2003.
- IEEE WIAPP'03 Travel Awards, 2003.
- Student Travel Support from Graduate School Fellowship of the University of Kentucky, 2003-2004.
- Award of Merit for a paper presentation at the 17th Annual EKU Symposium in the Mathematical, Statistical and Computer Sciences, 2003.
- Student Travel Support from Graduate School Fellowship of the University of Kentucky, 2002-2003.

Refereed Publications:

- Intelligent Preconditioner Recommendation System
  - Shuting Xu and Jun Zhang, Matrix condition number prediction with SVM regression and feature selection, in *Proceedings of the Fifth SIAM International Conference on Data Mining*, Newport Beach, CA, April, 2005.

- Shuting Xu and Jun Zhang, A data mining approach to matrix preconditioning problem, in *Proceedings of the Eighth Workshop on Mining Scientific and Engineering Datasets (MSD05), in conjunction with the Fifth SIAM International Conference on Data Mining*, Newport Beach, CA, April, 2005.
- Shuting Xu, Eun-Joo Lee, and Jun Zhang. Designing and building an intelligent preconditioner recommendation system (a progress report), in *Abstracts of the 2003 International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Scientific and Industrial Applications*, Napa, CA, 3 pages, October, 2003.

- High Performance Document Clustering
  - Shuting Xu and Jun Zhang, A hybrid parallel web document clustering algorithm and its performance study. *Journal of Supercomputing*, Vol. 30, No. 2, P. 117-131, Nov, 2004.
  - Shuting Xu and Jun Zhang, Clustering text documents with different closeness, in *Proceedings of the 2004 Workshop on Clustering High Dimensional Data and its Applications, in conjunction with the Fourth SIAM International Conference on Data Mining*, P. 34–42, Orlando, FL, April, 2004.
  - Shuting Xu and Jun Zhang, A hybrid parallel algorithm for clustering web documents, in *Proceedings of the 7th International Workshop on High Performance and Distributed Mining, in conjunction with the Fourth SIAM International Conference on Data Mining*, P. 66–73, Orlando, FL, April, 2004.

- Terrorist Analysis System with Privacy Protection

- Shuting Xu, Jun Zhang, Dianwei Han, and Jie Wang, Data distortion for privacy protection in a terrorist analysis system, in *Proceedings of IEEE Intl. Conf. on Intelligence and Security Informatics (ISI-2005)*, Atlanta, Georgia, May, 2005.

- Parallel object-relational database management system

  - Shuting Xu and Yongqiang Sun. Performance study Of real-time multiversion concurrency control protocols in parallel database systems. *Chinese Journal of Computers*, Vol. 25, No. 2, P. 173-180, 2002.

  - Shuting Xu, Chaojun Lu, Changsheng Chen, and Yongqiang Sun. A parallel transaction processing model based on BSP. *Journal of Computer Research and Development (Chinese)*, Vol. 38, No. 11, P. 1399-1404, 2001.

  - Changsheng Chen, Shuting Xu, and Yongqiang Sun. Object-relational database system and its parallel processing. *Journal of Computer Application and Software (Chinese)*, October, 2001.

  - Shuting Xu and Yongqiang Sun. An improved BSP I/O cost model. *Journal of Shanghai Jiaotong University*, September, 2001.

  - Shuting Xu and Yongqiang Sun. The comparison and analysis of multiversion access methods. *Journal of Computer Engineering (Chinese)*. Vol. 27, No. 5, P. 85-88, 2001.

  - Yongqiang Sun, Shuting Xu, Fenghua Zhu, and *et al.* PORLES: a parallel object-relational database system, in *Proceedings of ISES'01*, Wuhan, China, March, 2001.

  - Shuting Xu and Yingcai Bai. Concurrent engineering and the technol-

ogy of CAD/CAPP/CAM integration. *Journal of Computer Engineering (Chinese)*. Vol. 23, No. 12, P. 40-41, 1997.

Teaching Experience:

- Instructor, Spring 2005. Department of Computer Science, University of Kentucky.
- Instructor, Summer 2004. Department of Computer Science, University of Kentucky.
- Teaching Assistant, 01/2002 - 01/2003. Department of Computer Science, University of Kentucky.
- Instructor, 09/1994 - 07/1996. Department of Computer Science, Shandong University, China.