A RECOMMENDATION SYSTEM FOR PRECONDITIONED ITERATIVE SOLVERS

A Dissertation

by

THOMAS GEORGE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2009

Major Subject: Computer Science

A RECOMMENDATION SYSTEM FOR PRECONDITIONED ITERATIVE SOLVERS

A Dissertation

by

THOMAS GEORGE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Vivek Sarin
Committee Members,	Patrick Lynett
	Valerie Taylor
	Yoonsuck Choe
Head of Department,	Valerie Taylor

December 2009

Major Subject: Computer Science

ABSTRACT

A Recommendation System for Preconditioned Iterative Solvers. (December 2009)

Thomas George, B. Tech., Indian Institute of Technology, Madras;

M.S., Mississippi State University

Chair of Advisory Committee: Dr. Vivek Sarin

Solving linear systems of equations is an integral part of most scientific simulations. In recent years, there has been a considerable interest in large scale scientific simulation of complex physical processes. Iterative solvers are usually preferred for solving linear systems of such magnitude due to their lower computational requirements. Currently, computational scientists have access to a multitude of iterative solver options available as "plug-and-play" components in various problem solving environments. Choosing the right solver configuration from the available choices is critical for ensuring convergence and achieving good performance, especially for large complex matrices. However, identifying the "best" preconditioned iterative solver and parameters is challenging even for an expert due to issues such as the lack of a unified theoretical model, complexity of the solver configuration space, and multiple selection criteria. Therefore, it is desirable to have principled practitioner-centric strategies for identifying solver configuration(s) for solving large linear systems.

The current dissertation presents a general practitioner-centric framework for (a) problem independent retrospective analysis, and (b) problem-specific predictive modeling of performance data. Our retrospective performance analysis methodology introduces new metrics such as area under performance-profile curve and conditional variance-based fine-tuning score that facilitate a robust comparative performance evaluation as well as parameter sensitivity analysis. We present results using this analysis approach on a number of popular preconditioned iterative solvers available in packages such as PETSc, Trilinos, Hypre,

ILUPACK, and WSMP. The predictive modeling of performance data is an integral part of our multi-stage approach for solver recommendation. The key novelty of our approach lies in our modular learning based formulation that comprises of three sub problems: (a) solvability modeling, (b) performance modeling, and (c) performance optimization, which provides the flexibility to effectively target challenges such as software failure and multi-objective optimization. Our choice of a "solver trial" instance space represented in terms of the characteristics of the corresponding "linear system", "solver configuration" and their interactions, leads to a scalable and elegant formulation. Empirical evaluation of our approach on performance datasets associated with fairly large groups of solver configurations demonstrates that one can obtain high quality recommendations that are close to the ideal choices.

To my grandmother

ACKNOWLEDGMENTS

A lot of people asked me how I felt after my dissertation defense. All I remember is that I felt numb, but peaceful and content that it was finally over. Even though it was a rough ride fraught with adversity, I have been fortunate enough to have a number of friends and mentors that helped me all through the way.

First and foremost, my sincere thanks to my Ph.D. advisor, Dr. Vivek Sarin, for guiding me through this long process and helping me mature into an independent researcher. Even when I was having a lot of trouble finding a topic for my dissertation, he continued to believe in me and support me. When I first proposed the idea of a recommendation system for preconditioned iterative solvers, despite his apprehension about its feasibility, he helped me pursue it. Dr. Sarin's encouragement and support enabled me to obtain a wider exposure to the research community as well as the critical research challenges in my area. I have also been very fortunate to have had the opportunity to work with Dr. Anshul Gupta during my IBM internships. He not only helped me grow as a researcher, but has been a great friend and mentor. His feedback on my work and suggestions provided the basis for much of this dissertation. Anshul's high expectations pushed me to persevere in my work and ultimately set higher standards for myself.

There are a number of other professors that have inspired and encouraged me, especially my masters advisor, Dr. Edward Luke, who believed in me in spite of my limited computer science background, and Dr. Scott Pike, who rekindled in me the joy of learning and research when I was getting frustrated during my quest for a thesis topic. Attending Dr. Pike's course also gave me valuable insight on asking the right research questions and demonstrated the importance of effectively communicating one's ideas. I am also grateful to Dr. Teresa Leyk, who was extremely understanding, and helped me with reduced teaching assistant duties at times I needed it the most. I would also like to thank Dr. Valerie

Taylor, Dr. Patrick Lynett, and Dr. Yoonsuck Choe for agreeing to serve on my committee and for their helpful comments and constructive criticism.

I am glad to have had wonderful labmates Hemant, Kasthuri, Meiqui, Radhika, and Xue, who were always willing to lend an empathetic ear to all my research woes, in addition to being great friends. My friends have always been the most important people in my life, but especially so during the past few years of my graduate life. I am extremely fortunate that I got to meet such nice people as Jaya, Anand, Syam, Lekshmi, Raghu, Radhika, Sunita, Annie, Suju, Nitin, Pratheesh, Satish, Kums, Rupam, Ghosh, Girish, Rajith, Jayashankar, Jerry, Jolly, Alex, TP, Julie, and Gabi. Special thanks to Girija Aunty for feeding me through the last hectic months leading to my defense. I would also like to thank the Computer Science department staff for taking care of all administrative issues, and I will especially miss the love and affection of Kathy Flores and the friendly banter with Tony Okoinski.

I am deeply indebted to my family members and Ramani Chechi for their love and affection throughout my life. Nothing would have been possible without the love, support, and dedication of my grandmother. I am what I am because of her efforts. My extended family of uncles, aunts, and cousins both here in the USA and India also deserve special thanks for their advice and moral support which helped me handle the various adversities in life. Lastly, my wife, Srujana, deserves a special mention. She has been with me through thick and thin as a friend and a life partner. Without her love and encouragement throughout these years, this dissertation would not have been possible.

TABLE OF CONTENTS

Page

CHAPTER Page

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

TABLE Page

TABLE Page

TABLE Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

A fundamental step in most scientific and engineering simulations is the solution of systems of linear equations of the form $Ax = b$, where $A \in C^{m \times n}$ is typically known as the coefficient matrix, $b \in C^m$ is the right hand side vector (RHS) and $x \in C^n$ is the vector of unknowns. Large sparse linear systems involving millions and even billions of equations are becoming increasingly common in a number of problems arising from the discretization of PDEs, structural analysis, electric circuit simulations, linear and non linear programming, etc. These systems of equations can be solved using either *direct* or *iterative* methods. Direct solvers, which are based on factoring the coefficient matrix into invertible factors , are fast and robust, but are often not suitable for large three-dimensional problems due to their prohibitive computational and memory requirements. This limitation combined with the need to solve ever increasing sizes of linear systems has fueled a strong interest in iterative solvers that typically require much lesser memory and fewer computations [28]. However, iterative solvers are often plagued by failure and/or poor convergence and need to be coupled with carefully chosen and fine-tuned preconditioners for robust performance.

Decades of research has culminated in a wide range of iterative schemes and preconditioners for solving large sparse linear systems. As a result, practitioners now have access to an overwhelming number of choices of efficient implementations of various preconditioned iterative solvers in several black-box solver packages, such as PETSc [4], Trilinos [45], ILUPACK [57],Hypre [1], and many others [25]. However, for most of these solvers, there is no guarantee of success on large, complex matrices and it is extremely important to

---

The journal model is SIAM Journal of Scientific Computing.

choose the right parameters in order to achieve convergence. Often, even when the solver converges to the correct solution, there can be a wide disparity in the performance metrics such as computation time and memory usage for different choices of parameters. For example, the linear system *audikw_1* could be solved, within a memory and time limit of 16GB and 4 hours, by only 7 out of a total of 99 configurations that we experimented with for Hypre ParaSails configurations. Further, the memory and time values exceed the best values by up to a factor of 3 even among the 7 solved cases. Therefore, fine-tuning the preconditioner/solver parameters is critical both for ensuring the robustness of the solution as well as improving the performance of the preconditioned-solver in terms of the computation time, the memory resources, the accuracy of the solution and the number of iterations required.

A. Practical Challenges in Solver Selection

Choosing the appropriate scheme and fine-tuning the parameters for a particular linear system is an important, but highly challenging task even for experts in computational linear algebra and remains a blend of art and science due to variety of reasons that we discuss below.

1. Lack of Theoretical Analysis

The huge diversity among the existing preconditioners (e.g., IC(k), ICT, AMG, SAI) makes it difficult to analyze them in a unified theoretical model. There is also, often, a significant variability in the different implementations of the same preconditioner in different solver packages (e.g., PETSc IC(k), Trilinos IC(k), and Hypre IC(k)), which limits the utility of a purely theoretical analysis.

2. Complex Solver Configuration Space

A preconditioned iterative scheme, in general, has many elements — choice of package, solver algorithm, matrix re-ordering, preconditioner and various parameters specific to the preconditioner choice, resulting in a large and fairly heterogeneous solver configuration space. An exhaustive search over this space for optimal parameters using a simple trial and error strategy is non-trivial since some of the parameters could be continuous, discrete, or a nested/linear combination of both. Mutual dependencies among the solver/preconditioner parameters further exacerbate this difficulty by requiring a simultaneous exploration of the correlated parameters.

3. Multiple Solver Selection Requirements

Selection of a suitable solver is also, often, complicated by the fact that there is no universal "goodness" criteria and the appropriate solver(s) is determined by the optimization criterion specific to the application in consideration. For example, in one case, the best choice could be the fastest solver that provides the correct solution up to a certain relative error whereas in another scenario, the best choice could be the one that provides the best error within 4 hours of run time. Often, the application-specific optimization criterion is a hybrid function of multiple performance criteria, e.g., memory-time product, that can include constraints arising due to hardware limitations resulting in a large number of possible "goodness" criteria, each with its own set of suitable solvers for a given linear system.

4. Problem-specific Solver Performance

Empirical studies indicate that the effectiveness and performance of preconditioned iterative solvers is highly dependent on the linear system in consideration. For example, Table I shows the minimum and maximum values for the total time required for preconditioner cre-

ation and solving linear systems *kyushu* and *audikw_1* using 470 solver configurations spanning multiple preconditioners and packages (See Chapter III). The corresponding memory usage is also shown. The huge variations among the minimum and maximum values highlight the benefits of problem-specific fine-tuning. Therefore, simplistic solver selection strategies that disregard the linear system characteristics are of limited value. On the other hand, determining the optimal solver choices for the entire space of linear systems is non-trivial as there exists limited domain knowledge on the correlations between linear system properties and solver performance. This issue is especially critical in case of applications that require solution of a series of linear systems with the coefficient matrices changing gradually since the set of parameters that are best for the first system may not be suitable for the latter ones.

Table I. Minimum and maximum values for time taken for preconditioner creation and solving *kyushu* and *audikw_1* matrices for 470 iterative solver configurations spanning multiple preconditioners and packages. The table also shows the corresponding memory usage for storing the preconditioner as well as the linear system.

Matrix	**Metric**	**Minimum**	**Maximum**
kyushu	Time (s)	32.66	3891.68
	Memory (MB)	4547.04	6794.12
audikw_1	Time (s)	336.16	10390.40
	Memory (MB)	959.318	9327.76

5. Sparse Performance Data

The main utility in solver selection is for solving large linear systems, where one cannot afford to waste computational resources with a sub-optimal solver or multiple attempts. However, the huge requirements of memory and time resources also severely limit the collection of empirical performance data in such scenarios, which in turn limits our ability to glean insights on solver performance for different types of linear systems.

B. Motivation for a Statistical Framework for Solver Selection

In the past, there have been some attempts to analyze iterative solvers from a theoretical perspective, for example, the excellent survey articles by Benzi et al. [8] and van der Vorst et al. [73] describe the relative strengths and weaknesses of a wide range of preconditioned iterative schemes. However, the connections observed are far from being actionable by practitioners, especially given the huge variations in different package implementations of the same preconditioner, which are not discussed in these articles. This limitation of a purely theoretical approach has prompted a few empirical studies [9, 18, 34] on preconditioned iterative solvers, but these studies deal only with variants of an in-house implementation of a single preconditioner, providing little insight on the relative performance across preconditioners. So far, there has not been a thorough empirical evaluation of the readily available implementations of most common classes of preconditioners, most likely due to the extensive computational and software engineering effort involved in solving large matrices using a large number of iterative solver configurations.

On the other hand, recent technological advances have resulted in the creation of large linear systems leading to widespread adoption of iterative solvers, making it extremely important to provide guidance to practitioners, who currently rely on limited domain knowledge and ad hoc mechanisms for fine-tuning preconditioner parameters. Hence, there is an immediate need for a practitioner-centric empirical evaluation of the commonly used general purpose preconditioners available in black-box solver packages, especially for large, complex linear systems. Since such linear systems typically require significant computational resources and have a low solvability rate, the resulting data is often sparse and noisy. Analyzing such data entails a principled comparative methodology based on an informative representation of the solver configuration space and robust statistical metrics that are aligned to common practitioner decisions.

From the discussion on practical issues, it is also clear that fine-tuning solver configurations using linear system characteristics can provide significant performance improvements. Motivated by this promise, a number of recommendation systems have been proposed for software selection in scientific computing [49, 84] and in particular, for selection of sparse solvers [10, 13, 87]. However, these approaches have been demonstrated to be effective only for scenarios, where (1) one can obtain extensive performance data easily (i.e., training data is not sparse), and (2) the problems are restricted to a small homogeneous domain. Furthermore, the existing solver recommendation approaches [10, 13, 87] are based on simplistic formulations of solver selection in terms of classification and are capable only of providing highly coarse recommendations based on a single performance criterion. Since fine-grained optimization of parameters over multiple selection criteria is often required in practice, it is highly desirable to have a principled statistical methodology for choosing suitable "fully specified" solver configurations (i.e., pre-processing choices, preconditioner, iterative solver, and the associated parameters) for a specified linear system, in other words, an intelligent solver recommendation system.

C. Overview of the Dissertation

The primary goal of this dissertation is to develop methodologies and systems that can provide guidance to practitioners on choosing the best iterative solver configurations for any given linear system, while taking into account practical application requirements and constraints. The dissertation consists of two main parts, which are described below.

1. Retrospective Comparative Analysis of Solver Performance Data

The first part is an attempt to address the inadequacies in existing empirical studies and comprises of an extensive practitioner-centric comparative evaluation of a number of pop-

ular and promising general purpose preconditioners available in black-box solver packages over large linear systems chosen from a variety of domains. The typical solver selection modus operandi of practitioners is a multi-step process where the solver package and preconditioner are chosen first based on the target architecture (e.g., AIX, Linux), implementation specifics (e.g., MPI, OpenMP, C, C++), resource constraints (e.g., available memory) and available domain information on linear system properties, followed by refinement of one or more preconditioner parameters using heuristics or a trial and error policy. In order to make the optimal decision, practitioners require guidance on questions such as:

- How do the different package/preconditioners compare with each other and the direct solver with respect to time and memory?
- Does the relative performance change when parameters are fine-tuned?
- What is a good default configuration for each preconditioner in a package?
- For each package/preconditioner, what are the most sensitive parameters that can be fine-tuned for a specific linear system to improve performance?
- How do the answers to the above questions differ in a parallel setting?

To address these questions, we introduce a fairly general and rigorous methodology for retrospective analysis of performance data that allows (a) a robust comparison of solver configurations, or groups thereof, using area under performance-profile curve, (b) identification of the problem-independent best configuration within each solver configuration group for a given performance criterion, (c) estimation of the impact of fine-tuning various parameters using a suitable conditional variance measure. Based on this methodology, we developed a semi-automated system for the collection, analysis, and visualization of solver performance data that can be used to perform various comparative and sensitivity analyses

within and across pre-specified groups of solver configurations. This system was used to evaluate incomplete factorization, sparse approximate inverse, and the algebraic multilevel schemes available in packages such as PETSc, Trilinos, Hypre, ILUPACK, and WSMP along multiple dimensions such as robustness, speed and memory consumption both for serial and parallel settings, where possible. The results of this comparative analysis (presented in Chapter III) provide detailed guidance on choice of preconditioners and parameters that could be useful to practitioners, and also highlight potential areas of improvement in each solver package that could be beneficial to the software developers. Even though, the comparative methodology is targeted towards iterative solver selection, the key elements of our approach, for example metrics such as area under performance-profile curve, fine-tuning score, are readily generalizable to other software selection scenarios involving a complex space of software options.

2. Multi-stage Learning-based Solver Recommendation Approach

The second part of the dissertation focuses on linear system-specific predictive modeling of solver performance with the objective of providing recommendations on solver configurations. Though analogous to the typical user-item recommendation problem often studied in machine learning literature, the solver recommendation problem poses a unique set of challenges resulting from solver failure, huge variations in performance metrics, multiple/hybrid selection criteria, user interpretability requirements, and necessity of high quality cold start predictions, which make it impractical to directly employ existing techniques. To address these issues, we propose a novel multi-stage learning based methodology for determining the “best” solver configuration(s) given the user constraints and the desired performance behavior for any given linear system.

Our formulation of the solver recommendation problem comprises of three key subproblems: (a) solvability modeling, (b) performance modeling, and (c) performance opti-

mization. This decomposition allow one to readily address challenges arising from solver failure and multi-objective optimization. Our choice of instance space comprising of solver trials, i.e., pairs of linear systems and solver configurations, represented as vector of characteristics of linear systems, solver configuration parameters and their mutual interactions (a key distinction relative to existing approaches) also results in a fairly elegant formulation that is readily scalable with respect to the space of linear systems and solver configurations, while addressing practical concerns of users such as interpretability. Specifically, the solvability model based on an intrinsic performance criterion is used to filter out failure-prone configurations before modeling the performance statistics. Further, to accommodate optimization of multiple criteria, we separately learn models for each of the core performance statistics (e.g., time/memory/error). Standard classification and regression techniques augmented with latent factors are used to learn the solvability and performance models after suitable transformation of performance data. The optimization step involves combining the learned performance models to identify the top solver choices for the specified performance criteria. For the case, where generalized linear regression is employed to model the core performance statistics, we also propose an efficient methodology for identifying the top-k solver choices for multiplicative combinations of the core performance statistics using monotonic rank aggregation techniques.

We implemented the proposed methodology as a modular self-learning solver recommender system with specialized components dedicated to data collection, feature generation, offline learning and online recommendation unit. Using this system, we evaluated the key aspects of the proposed approach on different subsets of solver performance data using models at different levels of granularity. The resulting solver recommendations demonstrate the efficacy and potential promise of our approach. As in the case of the first study, the proposed solver recommendation methodology can also be readily adapted to other software selection scenarios when there is sufficient domain knowledge on the informative

problem and software characteristics.

3. Organization

The rest of the dissertation is organized as follows: Chapter II provides a survey of the state of the art in iterative solvers, preconditioners, black-box solver packages as well as a brief review of existing approaches for using machine learning techniques for scientific software selection. It also includes some background material on classification, regression and feature selection techniques that are later employed in our solver recommendation approach. Chapter III presents various components of our novel practitioner-centric methodology for retrospective performance analysis, as well as a description of the performance analysis infrastructure. The results obtained by analyzing performance data from a wide range of preconditioned iterative schemes on a set of benchmark problems are also presented in this chapter. Chapter IV discusses the key desiderata for a solver recommendation system and presents details of our multi-step learning based methodology as well as prototype system for identifying the best solver configurations given a linear system and user requirements. Chapter V provides an empirical evaluation of various aspects of the proposed solver recommendation system for two scenarios that differ in the granularity of the solver configuration and linear system space. Chapter VI summarizes the main contributions of the dissertation and discusses possible extensions involving active collection of performance data, optimization over a continuous solver configuration space and applications to broader software selection scenarios.

CHAPTER II

RELATED WORK AND BACKGROUND

In this section we describe the state of the art in iterative solvers, preconditioners, black-box solver packages for solving linear systems, and also discuss the various machine learning techniques used in expert systems for scientific software.

A. Iterative Solvers

Over the years, a number of iterative solvers have been developed and these are currently the best known methods to deal with very large sparse matrices. The methods range from simple stationary iterative methods to more complex methods such as Krylov subspace methods and multigrid techniques. The applicability and effectiveness of a few of the iterative solvers in these main classes is described in more detail in the following sections.

1. Stationary Iterative Methods

Stationary iterative methods such as Jacobi, SOR involve some form of splitting of the coefficient matrix and the solution at the j^{th} iteration is represented as a linear combination of the solution of the $j-1^{th}$ iteration. These methods can be written in the general form

$$Cd^{i+1} = r^i, \tag{2.1}$$

$$x^{i+1} = x^i + d^{i+1},\ i = 0, 1, 2 \cdots \tag{2.2}$$

where $r^i = b - Ax^i$ is typically known as the residual, d^{i+1} is the correction at the i^{th} iteration, and C is a matrix which depends on A and the choice of the specific method.

When $A = C - R$, the above equations can also be written in the alternative form :

$$x^{i+1} = C^{-1}Rx^i + C^{-1}b. \tag{2.3}$$

Let the coefficient matrix $A = L + D + U$, where D is the diagonal (assumed to be without any zero elements), L is the strict lower triangular part, and U is the strict upper triangular part of A. Jacobi, Gauss-Seidel, and SOR methods differ in the choice of the splitting matrix C and are described in more detail below.

a. Jacobi

The Jacobi method corresponds to the simple form where the matrix $C = C_J$ is the diagonal D of the coefficient matrix A. From equation (2.3), we observe that the $(i+1)^{th}$ iteration involves inverting a diagonal matrix. Even though there might exist other methods that provide faster convergence in a serial environment, Jacobi methods might be preferable for massively parallel systems (for example, electronic structure simulation) either as a solver or preconditioner mainly due to the embarrassingly parallel nature of the algorithm.

b. Gauss-Seidel

Gauss-Seidel method (GS) improves on the Jacobi method by including all the new values generated till the $(i-1)^{th}$ iteration in correcting the i^{th} component of the residual vector. For GS iterations, the matrix $C = C_{GS}$ is either the lower triangular portion of A, i.e.,$L + D$ (*forward GS*) or the upper triangular portion of A, i.e., $D + U$ (*backward GS*). There also exist symmetric GS iterative schemes where each iteration consists of a forward GS followed by a backward GS. In any implementation of the GS method, one needs to keep track of the new and old variables. Different implementations differ in the order in which the variables are updated. Two most commonly used orderings are *red-black* and *natural* ordering. The ordering chosen for implementation is important since it affects the

convergence behavior.

c. Successive Over Relaxation (SOR)

SOR method is an improvement on the GS method since it uses a weighted average of new and old values for updating. SOR is based on the splitting of the matrix $A = ((D + \omega L) + (\omega U - (1-\omega)D))/\omega,\ 0 < \omega < 2$. If $\omega < 1$, then it is called *under relaxation* and if $\omega > 1$, then it is called *over relaxation*. In general, it is not easy to compute the optimal value of ω that maximizes the convergence. In practice, adaptive methods are used to converge to the optimal value starting from an initial guess either using simple heuristics or using knowledge of the underlying problem domain.

d. Applicability Conditions & Convergence Analysis

(a) If the matrix A is *strictly diagonally dominant* (i.e., each diagonal element is larger than the sum of the magnitudes of off-diagonal elements in its row), the Jacobi and GS iterations are guaranteed to converge. Writing $C^{-1}R$ as G, we can say that in fact $||G_{GS}||_\infty \leq ||G_J||_\infty < 1$. A similar statement can be made even for column diagonally dominant matrix. The lower the value of the $||.||$, the faster the convergence.

(b) If the matrix A is *irreducible and weakly diagonally dominant*, the Jacobi and GS iterations converge and $\rho(R_{GS}) < \rho(R_J) < 1$. A smaller value for the spectral radius ρ indicates faster convergence, a value close to 1 implies poor convergence and a value greater than 1 indicates divergence.

(c) GS and Jacobi methods are found to converge even for matrices that do not satisfy conditions (a) and (b). However, it is necessary that the diagonal terms in A are greater in magnitude than the off-diagonal terms which can be achieved by reordering.

(d) Even though for special cases (a) and (b), the GS method is faster than the Jacobi method, this is not true in general. There exist non-symmetric matrices for which the Jacobi method converges while GS method diverges and vice versa [82].

(e) If A is symmetric positive definite (SPD), SOR converges for all ω in the range $0 < \omega < 2$. If A is not SPD, then the above condition is just necessary and *not* a sufficient condition for convergence. The rate of convergence depends on the choice of ω and the reordering scheme used. Since GS is a special case of SOR where $\omega = 1$, GS also converges for SPD matrices.

In the past, stationary iterative methods were used for most computational simulations. However, in recent years they are slowly being replaced by preconditioned Krylov subspace methods due to their superior convergence rate [6]. Nowadays, these stationary methods are commonly used as preconditioners for Krylov subspace methods or as smoothers for multigrid based solvers since they are relatively inexpensive.

2. Krylov Subspace Methods

Krylov subspace methods such as CG and GMRES are characterized by the subspaces in which the solution iterates x_j lie. The m^{th} Krylov subspace for a given matrix A and vector r_0 is given by the following:

$$\mathcal{K}_m(A, r_0) = span\{r_0, Ar_0, A^2r_0, \cdots A^{m-1}r_0\} \tag{2.4}$$

Various Krylov subspace methods differ in the criteria they use in selecting a vector in the subspace.

a. Conjugate Gradients Method (CG)

The Conjugate Gradients (CG) method picks the j^{th} solution iterate x_j to be the vector that minimizes the A-norm (i.e., $||x - x_i||_A = (x - x_i), A(x - x_i))$ of the error over the Krylov subspace for any SPD coefficient matrix A. The new residual is orthogonal to the space of previously generated residuals or some related space. The main advantage of CG is its low computational and memory requirements. The CG method has also been applied to general unsymmetric matrices by applying it to the normal equations (CGNE) or normal residual (CGNR) formulation of the linear system. CG can be applied in this case because even though the matrix A is unsymmetric, the product AA^T or A^TA is SPD.

b. Generalized Minimal Residual Method (GMRES)

The GMRES method picks x_j to be the vector that minimizes the two norm of the residual over the Krylov subspace. At each iteration, GMRES generates one dimension of an orthonormal basis for the Krylov subspace $K_i(A, r_0)$. The residual norm, $||ri||_2$, is minimized via a least squares problem. A major drawback for this method is that the computation cost increases linearly with iteration count since all the basis vectors of the Krylov subspace have to be stored to guarantee convergence. To alleviate the problem with storage requirements, restarted versions of GMRES are used in practice.

c. Applicability Conditions & Convergence Analysis

CG: If A is SPD then, CG is guaranteed to converge in a finite number of iterations. Specifically, there exists a pessimistic bound on the number of iterations required for reducing the error by a fixed factor which is proportional to the square root of the condition number. However, in practice CG converges to the pre-specified tolerance in much fewer iterations. This practical behavior is explained based on the eigen value distribution of the

coefficient matrix. A general rule of thumb is that if the largest and smallest eigen values of A are clustered closely together, then the CG method would converge quickly. Convergence of CG method, in practice, might differ from that in exact arithmetic [23]. The hope is that as long as the matrix is not too ill-conditioned, the floating point result would ultimately converge to the desired solution. The CG method is best suited for applications producing SPD matrices. Target applications include matrices from structural engineering simulations of offshore platform, suspension bridges, buildings, etc.

GMRES: For any positive definite matrix (i.e., the symmetric part of $(A + A^T)/2$ is SPD), GMRES will eventually converge. For PD matrices that are normal (i.e., $AA^T = A^TA$) there exists bounds based on the condition number or eigen values for the residual after m GMRES iterations. There does not exist any simple analysis in terms of eigen values of A or condition number of the eigen vector matrix for general unsymmetric matrices. Greenbaum et al. [35] established that for highly non-normal matrices with the same spectrum *any* convergence behavior is possible. In practice, the restarted version of GMRES is often used and there does not exist any theoretical explanation for its convergence behavior. The convergence properties vary greatly with the choice of the restart value [18]. Preconditioning techniques that reduces the number of iterations could become really useful for using low values for restart. There are a number of applications that produces unsymmetric matrices and GMRES is a natural choice for these domains such as circuit physics modeling, chemical kinetics, oil reservoir simulation, economic modeling etc.

3. Multigrid Methods

Multigrid methods, even though originally developed for the solution of discretized elliptic PDEs, are now applicable to a more general class of problems. The main idea in this approach is to represent the residuals as composed of low and high frequency modes. The use of an iterative method (smoothing) reduces the high frequency components quickly

and produces an approximate solution. This solution on the fine grid is then restricted to a coarse grid where the previous low frequency components of the residual in the fine grid becomes high frequency components in the coarse grid (restriction). This is recursively applied to a few more coarser levels and then interpolated back to the fine grid (prolongation). An algebraic version of multigrid method has been developed for use in black box solvers, which uses the properties in the coefficient matrix alone and does not assume the existence of underlying meshes. Multigrid methods may be used as a solver or as a preconditioner with Krylov subspace methods.

a. Applicability Conditions & Convergence Analysis

The fundamental assumption for geometric multigrid methods is the availability of an underlying mesh. Algebraic Multigrid (AMG) methods address this limitation by defining the interpolation and prolongation operators in an algebraic way. In practice, AMG has been shown to be applicable for a wide range of matrices. However, careful tuning of the different parameters (restriction, prolongation operators, smoothers, number of levels, etc.) are often needed to improve the convergence rate for most real life applications.

Convergence results exists for multigrid methods for problems ranging from simple differential operators such as Poisson operators on structured grids to self adjoint elliptic PDEs on arbitrary domains. For this class of problems, the convergence rate of multigrid methods is independent of the problem size which is almost optimal except for the hidden constant, i.e., the work required to solve the system is proportional to the number of unknowns and the number of iterations required to solve remains almost constant. Some of the applications where multigrid techniques have been successfully employed include image segmentation, quantum chemistry, structured grid generation, and VLSI design.

For all these iterative solvers, there exists theoretical knowledge on the convergence behavior only for a narrow class of problems and generalization to general SPD and unsym-

metric matrices is non-trivial. Since there is no single best method that is suited for all the problems [61], researchers have developed different preconditioners/pre-processing steps to improve convergence while using minimal computational and memory resources. If the problem under consideration belongs to a class for which there exist theoretical bounds on convergence, then one could make a judicious choice on the iterative method needed to solve. However, in the case of general SPD and unsymmetric linear systems, there are a number of methods that are possible candidates and whether a method is applicable or not is usually determined from experience gained in performing empirical studies.

B. Preconditioners

A preconditioner broadly refers to an explicit or implicit scheme that modifies the original linear system such that it is easier to solve using an iterative method. For example, solving $Ax = b$ is equivalent to solving $M^{-1}Ax = M^{-1}b$ for a non-singular matrix M. If $M^{-1} \approx A^{-1}$, then the preconditioned linear system has better spectral properties, thus achieving faster convergence rate for iterative methods, while ensuring that the original solution remains unchanged. An ideal general purpose preconditioner should have the following characteristics for successful deployment in a black-box iterative solver package.

- **Effectiveness:** The use of a preconditioner should result in a reduction in the cost of computing the solution which is often measured as the number of iterations. However, this is an effective indicator only if it is accompanied by a reduction in the total time needed for obtaining the solution, because preconditioned iterations tend to be more expensive than un-preconditioned ones.

- **Robustness:** It is important that the preconditioner provide the computational improvements while retaining applicability over a wide range of problems and not be susceptible to numerical instabilities.

- **Parallelizability:** Preconditioning a linear system should not cause any additional serial components to be introduced or reduce the parallel efficiency that could be achieved in its absence. If such serial bottle necks are unavoidable, then the increase in time for the parallel case should be small relative to the total savings due to preconditioning.

- **Parameter Predictability/Adaptability:** Current general purpose preconditioners for iterative solvers require the user to specify a number of parameters. These parameters often have a significant impact on the solver performance and usually have to be carefully fine-tuned for a given problem using trial and error methods to obtain good performance. Hence, it is highly desirable to have parameters that are known to affect the convergence properties in a predictable manner so that they can be chosen with less effort. An alternate option would be to have a preconditioner accompanied by an adaptive scheme that chooses the appropriate parameters for a given linear system [39].

The above conflicting requirements have challenged researchers for a long time and substantial work has been done in developing preconditioners that addresses most of these requirements if not all. Even though simple preconditioners like Jacobi are cheap to apply and have good parallel efficiency, they are limited in their applicability [8]. We now describe a few promising general purpose preconditioners available for SPD linear systems.

1. Level-based Incomplete Cholesky Factorization (IC(k))

An important class of preconditioners for SPD systems is based on incomplete Cholesky factorization of the coefficient matrix. There are several variations of incomplete factorization that differ in the rules for dropping entries to compute the incomplete factors. One strictly positional criterion for dropping is based on what is known as the *level of fill*, which

is a measure of "closeness" of a fill entry to the original entries in the coefficient matrix (please refer to the book by Saad [71] or the survey by Benzi [8] for a formal definition). In IC(k) factorization, all fill-in entries at levels exceeding k are dropped. An important advantage of a pure IC(k) preconditioner is that the sparsity pattern can be determined a-priori by a symbolic factorization step and the cost of constructing the preconditioner is amortized when solving multiple linear systems with the same sparsity pattern. Often, depending on the implementation, when the parameter $k > 0$, it is supplemented with additional parameters to handle fill levels higher than 0. Thus, many implementations of the IC(k) preconditioner use a combination of dropping based on both the position and magnitude of the nonzeros. The following parameters are used in the implementations of IC(k) that we study in this paper.

1. *Highest fill level,* k, is the fundamental parameter in all implementations of the IC(k) preconditioner and denotes the level beyond which all fill-ins are dropped.

2. *Fill factor* specifies an upper bound on the amount of memory used by the preconditioner for levels of fill greater than zero. A fill factor f denotes that the preconditioner would not use more that f times the number of nonzeros in the original matrix.

Careful partitioning and ordering of sub-domains has been shown to be effective in obtaining scalable parallel implementations of the IC(k) preconditioner [51]. One approach for parallelization of IC(k) is to use the sequential IC(k) algorithm within each sub domain, popularly known as the Block Jacobi based IC(k). In general, it has been observed that the Block Jacobi version of IC(k) is more scalable than a true parallel implementation of IC(k).

2. Threshold-based Incomplete Cholesky (ICT)

The threshold based incomplete Cholesky or ICT preconditioners control fill-in by means of a dual dropping strategy based on a numerical threshold τ and an upper limit f on the

number of fill-ins in each row or column. Typically, any new fill-in whose magnitude is below τ times a chosen metric is dropped. In addition, if the number of nonzeros in a row or column of the factor exceeds f times the number of nonzeros in that row or column in the original coefficient matrix, then the excess entries with the smallest magnitude are also dropped. Lower values for drop tolerance lead to more accurate preconditioners but result in higher memory consumptions, and vice versa.

There exists variants of ICT that use a multi-level incomplete factorization strategy (MLICT) combined with static pre-ordering and a dropping criterion that attempts to minimize the norms of the inverses of the triangular factors [57]. Another variation in BlockSolve95 [53] incorporates a hybrid strategy that uses ideas from both level-based and threshold based incomplete factorization preconditioners. The sparsity of the factors are guaranteed by retaining only the largest elements such that the memory usage is no larger than that required by a ILU(0) preconditioner. BlockSolve95 has the added advantage that no parameters need to be specified.

3. Sparse Approximate Inverse (SAI)

The problems inherent in using incomplete factorization based variants are partially addressed by preconditioners based on sparse approximate inverses [9]. Depending on the algorithms used for finding the sparse inverse, approximate inverse based preconditioners could be fairly robust in practice and easily parallelizable. However, these preconditioners usually incur a high initial setup cost and the efficacy and the cost of applying the preconditioner depends on the choice of the sparsity pattern. We study only a single implementation of approximate inverse preconditioner which uses *a-priori* knowledge of sparsity patterns and Frobenius norm minimization to generate an approximate inverse (ParaSails [17, 19]). For SPD matrices, a symmetric factored approximate preconditioner is generated. ParaSails uses three main parameters for controlling the accuracy and the cost of the preconditioner,

and these are described below.

1. *Threshold* controls the sparsification of the coefficient matrix such that it can be used to generate the *a-priori* sparsity pattern by dropping elements that are below the specified value. The range of values for *threshold* is [0,1]. One can also specify a negative value for *threshold* such that its absolute value dictates the percentage of nonzeros that must be dropped. The exact value for *threshold* is determined automatically in this case.

2. *Number of levels* controls the memory usage of the resulting preconditioner. ParaSails uses *a-priori* sparsity patterns that are powers of sparsified matrices. For example, if a value of 2 is used for the number of levels, then the sparsity pattern corresponds to the power of 3 of the sparsified matrix. Typical values are 0, 1 and 2 with the default value being 1.

3. *Filter* is a numerical threshold used to reduce the cost of applying the preconditioner by further dropping elements from the computed approximate inverse. This parameter works similar to *threshold* and one could also specify a negative value if it has to be determined automatically based on a percentage of sparsity that is desired. For example, if $filter = -0.9$, then the threshold is calculated such that 90% of the non zeros in the computed preconditioner are dropped.

4. Algebraic Multigrid (AMG)

Algebraic multigrid or multilevel methods are currently enjoying a lot of popularity as black-box solvers. The basic idea of an algebraic multilevel solver is to construct a hierarchy of coarser graphs, where each node in a coarse level represents multiple nodes of the previous finer level. At each coarse level, an iterative solver or a smoother is used to compute an approximate solution to system corresponding to that level and then project

this solution to the next finer level. The entire scheme can be viewed as a preconditioner for an iterative solver at the finest level.

We study two implementations of AMG, namely BoomerAMG [43], which is a parallel implementation of the classical AMG [70] available in Hypre, and Trilinos ML [30, 79], which includes a parallel implementation of the smoothed aggregation approach [80] for AMG. Trilinos provides default sets of parameters for three main preconditioner types for problems arising from specific domains; classical smoothed aggregation for SPD or nearly SPD systems (SA), classical smoothed aggregation based 2-level domain decomposition (DD), and 3-level algebraic domain decomposition (DD-ML).

Implementations of AMG typically have a large number of user tunable parameters of which the most important[1] ones are listed below.

- *Smoothers* - Hybrid symmetric Gauss-Seidel/Jacobi [1] is the default smoother of choice since CG is the default solver used for SPD matrices. For Trilinos ML, symmetric Gauss-Seidel, Chebyshev polynomial, and IFPACK smoothers are some of the influential ones.

- *Coarsening schemes:* There are multiple coarsening schemes available in Hypre BoomerAMG, of which the important ones are Falgout (FALG), Parallel Modified Independent Set (PMIS) and Hybrid Modified Independent Set (HMIS). Similarly, in case of Trilinos ML, the popular coarsening schemes for classical smooth aggregation (SA) include Uncoupled, MIS, hybrid Uncoupled-MIS, and ParMETIS.

- *Number of Smoother Sweeps:* This parameter gives users of Trilinos ML another means of controlling the trade-off between the cost per iteration and the number of iterations required. Typical choices are values of 2 and 3 for symmetric Gauss-Seidel

[1] Based on personal communication with authors of BoomerAMG and Trilinos ML.

and Chebyshev polynomial smoothing.

- *Number of levels for aggressive coarsening (AGG):* The value given to this parameter in BoomerAMG sets the number of levels for which aggressive coarsening must be applied starting from the finest level.

- *Strong threshold:* The value specified for this BoomerAMG parameter serves as a threshold to determine whether two points in a graph are strongly or weakly connected. High values of strong threshold lead to cheaper, but less effective preconditioners whereas low values result in expensive preconditioners with better convergence properties.

C. Scientific Software Selection

In this section, we describe existing work on scientific software recommendation systems and also some of the adaptive techniques that have been proposed to improve the performance and robustness of preconditioned iterative solvers.

1. Expert Systems for Recommending Scientific Software

The use of data mining techniques for knowledge discovery is not a recent development in the scientific community [67]. One of the early influential works in this area is the recommendation portal PYTHIA-II [49], which provides users with the data management infrastructure to make suitable software choices. Data mining techniques have also been used for recommending specialized applications such as graph partitioning software [83] and solvers for elliptic PDE problems [68] using domain knowledge and empirical performance data.

Problem solving environments (PSE) are becoming commonplace and are touted as the future of scientific computing [50], especially, for use in remote computing technolo-

gies. PSEs allow the user to compose applications without knowing the details of the underlying algorithms and were introduced into the world of iterative solvers by the Linear System Analyzer [14]. Using the LSA, a user can compose a solution strategy by making choices on the pre-processing steps, the preconditioner, and the solver. Typically, efficient implementations of promising research preconditioners and iterative solvers are available as components of a PSE. Therefore, the number of choices that have to be made for solving a sparse linear system is huge, and numerous intelligent systems have been proposed for recommending solvers/preconditioners to alleviate the burden on the application developer [10, 13, 86]. Most of these existing approaches, however, involve a simplistic formulation of the solver selection problem that focuses on the solvability of a linear system, or in case of [10] achieving a fixed improvement over a default method. Such a formulation readily translates to a binary classification problem, which is then addressed using off-the-shelf association rule and classification algorithms. A recent research effort [55] attempts to use reinforcement learning for solvability prediction, however, with limited success in obtaining good results in comparison to more expensive supervised learning techniques.

2. Learning-based Recommendation Systems

Statistical techniques for estimating dyadic response functions, in particular, the preference ratings of users for products, form another large body of research that is relevant to our current work. Tuzhilin et al. [2] provide a detailed survey of machine learning techniques for recommendation systems. These include unsupervised techniques such as [5, 44, 69] that rely only on the local structure of the preference ratings, supervised approaches that make use of user demographic and product content attributes, as well as hybrid approaches [3, 63] that leverage both the correlations in the ratings as well as the user-product attributes. These approaches almost entirely focus on improving the accuracy of *all* the preference ratings, without specifically considering the additional sensitivity required for the desirable range,

or the algorithmic aspects of efficiently generating the top k recommendations. Though effective for product recommendation systems, these techniques do not consider typically practical aspects such as the variability in the performance values, feature selection as well as the final application goals that are critical for solver selection.

3. Adaptive Preconditioned Iterative Solvers

Over the past few years, there have been a number of empirical studies [18, 33, 34] on iterative solvers that highlight the importance of problem specific fine tuning for improving solver performance and robustness. In order to address the high failure rate of iterative solvers, the idea of *poly-iterative* solvers was proposed in [7] to use variants of the CG method (CGS, B-CGStab, QMR) simultaneously on a single problem in parallel. The iterations are stopped as soon as a single method converges. Use of poly-iterative solvers results in higher solve times in comparison to the best method, but has a reduced failure rate. This approach is predominantly suited for distributed memory machines and the authors also present optimization steps to amortize the communication overhead of the various methods. Another related approach to improve the robustness of iterative solvers is to use the idea of *composite* and adaptive solvers [11]. Unlike poly-iterative solvers, the constituent base solvers are applied in sequence and not in parallel. The key idea is a combinatorial scheme which uses metrics such as solve time and mean failure rate to construct a composite solver that is more reliable than the base methods. Adaptive solvers [11, 39] where the solver parameters are dynamically chosen at the beginning of each iteration based on the characteristics of the linear system as it changes during the iterative solution process are other approaches aimed at improving the performance as well as robustness of iterative solvers. Recent empirical studies [33] indicate that adaptive solutions are highly effective.

D. Machine Learning Techniques

This section gives an introduction to the machine learning techniques that will be used in our recommendation approach.

1. Classification

Classification refers to the task of assigning a set of input objects into a predefined set of target classes. In an inductive machine learning context, this typically consists of (i) a training phase that involves learning a classifier from a set of example objects labeled with the corresponding classes, and (ii) a prediction phase where the learned classifier is deployed to label new objects. It can be formally defined as follows. Given training data with n labeled examples $(x_1, y_1), (x_2, y_2), \cdots (x_n, y_n)$, $x_i \in \mathcal{X}, y_i \in \mathcal{Y}$, the goal of supervised classification techniques is to identify a function $h : \mathcal{X} \rightarrow \mathcal{Y}$ from a hypothesis class $\mathcal{H}$ that maps any object $x \in \mathcal{X}$ to its target label $y \in \mathcal{Y}$ such that the quality of predictions on the training examples is optimized

$$\max_{h \in H} \sum_{i=1}^{n} Q(h(x_i), y_i).$$

In most commonly used classification methods such as decision trees [65], Naive Bayes, neural networks [60], support vector machines [15], the input objects in $\mathcal{X}$ are represented as vectors of predictive features and the classifier is chosen to optimize quality criteria such as misclassification error, data likelihood or margin. The choice of the hypothesis class $\mathcal{H}$ and the learning algorithm also result in further diversity in the performance with the best technique depending on the data distribution to some extent. However, previous research on modeling solver performance [10, 87, 41] has shown that the choice of feature representation is often more important than the choice of the classifier itself. We will now describe some of the supervised classification algorithms that have been used in the current

work.

a. Decision Trees

Decision tree-based classification [64] is one of the most widely used effective methods for inductive inference. A decision tree classifies an example by asking a series of questions, each pertaining to the value of a single feature of the input item. The questions are associated with the internal nodes of the tree and the response to question determines the appropriate child node to consider next. These internal nodes are called splitting attributes or predictors and the questions contained within these nodes are called the splitting predicates. The leaf nodes contain the class information and an item is assigned a class label based on the path of the query from the root to a leaf. There are many variations of decision tree classifiers and they differ primarily in the order of assignment of the interior node, splitting attributes, and splitting predicates. Depending on the training data, one might have to perform *pruning* to avoid the risk of over-fitting. The state-of-the-art decision tree algorithms can handle categorical, continuous valued features, multi-class classification problems, and are robust to outliers. Techniques such as boosted decision trees [66] and alternating decision trees [29] that combine output of multiple decision trees have been shown to provide even superior performance.

b. k-Nearest Neighbors (k-NN)

k-NN is a supervised algorithm for classifying an object based on a majority vote by its "closest" neighboring examples in the feature space. The notion of "closest" depends on the training data and the distance metrics suitable for the application. When the training examples are multi-dimensional vectors in a feature space, Euclidean distance is typically used as the metric provided the values are numerical. When a new example has to be classified, the distances to all the training examples are calculated and the class labels

corresponding to the k closest training examples are used for the majority vote. The ideal value of k is dependent on the data and is typically chosen via cross-validation. A simple majority voting has the drawback that it biases the predictions towards classes with more frequent training examples. However, this drawback can be addressed by inverse weighting the votes with the true distance measures. If the training data is noisy, then a pre-processing of the features in the form of scaling or feature selection is often needed to improve the accuracy of predictions. Although the naive k-NN algorithm requires the computation of pairs of distances over the entire training data, a number of variations that optimize this step by pruning the candidate neighbors have been proposed.

c. Support Vector Classification

Support vector machine-based classification (SVM) was developed by Vapnik et. al. [12] and has gained widespread popularity and acceptance due to its strong theoretical foundations based on structural risk minimization, its ability to generalize to unseen data, and promising empirical performance on problems from various domains. SVM tries to find a linear separating hyperplane that has the maximum distance with examples of either class, also known as the maximum-margin. Often, there does not exist a hyperplane that clearly separates the data points belonging to the positive and negative classes. To address such scenarios, the notion of a "soft margin" was introduced [20]. The key idea is to allow certain data points to be correctly classified by allowing a tolerance while still maximizing the distance from the hyperplane to the cleanly split examples. In certain cases, even soft margins are not adequate for obtaining an acceptable separation of the two classes and it is necessary to map the training vectors to a higher dimensional space by means of a kernel function. A good kernel function will result in the data being (nearly) linear separable in the higher dimension space.

Given a set of n training instance-label pairs $\{x_i, y_i\}_{i=1}^{n}$, where $\mathbf{x}_i \in R^d$ and $y_i \in$

$\{-1, 1\}$, the support vector classification requires the solution of the following optimization problem:

$$\begin{aligned} & \min_{\mathbf{w},b,\xi} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle + b) \geq 1 - \xi_i, \;\; 1 \leq i \leq n \\ & \xi_i \geq 0 \end{aligned} \tag{2.5}$$

where $\mathbf{w}$ is a normal vector to the hyperplane, ξ is the slack variable that measures the degree of mis-classification of $\mathbf{x}_i$, $C > 0$ is the penalty parameter for the error term, and ϕ is the function used to map the training vectors x_i into a higher dimensional space via a positive definite kernel $K(\cdot, \cdot)$, such that $K(\mathbf{x}_1, \mathbf{x}_2) = \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2)\rangle$. The optimization problem attempts to achieve a trade-off between the maximal margin and a small error penalty. Typically, the optimization is solved using a dual formulation that associates each data point x_i with a weight α_i where a non-zero α_i indicates a support vector. The learned model can then be used to obtain predictions on any input object $\mathbf{x}$ using

$$f(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \phi(\mathbf{x})\rangle + b) = \text{sign}(\sum_i \alpha_i y_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x})\rangle + b) = \text{sign}(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b).$$

Though the original SVM classification algorithm [12] is computationally expensive, in recent years, fast variants [52] that only require linear time (with respect to the number of training examples) have been proposed. SVM-based classification, however, does suffer from one main limitation namely the inability to provide class assignment probabilities. The original SVM formulation is also restricted to binary classification though there have been extensions to multi-class scenarios.

2. Regression

Regression techniques are used to model the behavior of a real or integral valued target property of an object in terms of other predictive characteristics. As in the case of classification techniques, there is a training phase and a prediction phase. The training phase involves identifying the best functional mapping $h : \mathcal{X} \mapsto \mathcal{Y}$ from the input space to the output space that optimizes the fit on the training data. However, unlike in case of classification techniques where the target property takes only nominal values, there is an ordering among the values of the target property. Hence, the quality criteria used for regression techniques tend to be based on a suitable distortion measure associated with the target property, for example, mean squared error. We now discuss two commonly used regression techniques namely linear regression [59] and support vector regression [26].

a. Linear Regression

Linear regression [59] refers to fitting a linear model to predict the target response variables. Formally, the linear model can be expressed as $y = \beta^t x + \epsilon$ where β denotes the vector of regression parameters and x is a suitable representation of the input objects as a vector of the predictive features. The most common choice of linear models is linear least squares regression which is applicable to real-valued target variables with the error term ϵ assumed to belong to a normal distribution $\mathcal{N}(0, \sigma^2)$ with zero mean and constant variance (assumption of homoscedasticity). However, when the training data contains outliers or the assumption of homoscedasticity is not valid, robust regression [47] is preferred. Extensions to generalized linear models [59] that allow the linear model to be related to the response variable via a link function and the noise term ϵ to be drawn from any exponential family distribution, can be used for modeling a much wider variety of target variables.

b. Support Vector Regression

Support vector regression [26, 77] is an extension of support vector machines for approximating real-valued functions. The key idea is to find a function $f(x)$ that has at most ϵ deviation from the actual target property y_i for all the training data, while remaining as flat as possible. In case of linear models where $f(x) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b$, the notion of flat translates to minimizing the norm of $\mathbf{w}$ as in the case of support vector classification. This formulation also ensures that we can disregard errors that are less than ϵ and attempt to optimize the model over the support vectors that exceed this limit. Since it might not always be feasible to obtain such a function $f(x)$ for a given ϵ, the formulation also allows for slack variables ξ_i^l, ξ_i^u that are analogous to the "soft margin" in case of support vector classification. Using the notation for the training data as before, the resulting optimization problem is given by

$$\begin{aligned} & \min_{\mathbf{w},b,\xi} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i^u + \xi_i^l \\ \text{subject to} \quad & y_i - \langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle + b \leq \epsilon + \xi_i^u, \;\; 1 \leq i \leq n \\ & \langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle + b - y_i \leq \epsilon + \xi_i^l, \\ & \xi_i^l, \mathbf{x}_i^u \geq 0, \end{aligned} \tag{2.6}$$

where ϕ is the function used to map the training vectors x_i into a higher dimensional space via a positive definite kernel $K(\cdot, \cdot)$, such that $K(\mathbf{x}_1, \mathbf{x}_2) = \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle$ and the positive constant C determines the trade-off between the "flatness" of the model and the extent to which additional errors (over ϵ) are tolerated. As in the case of support vector classification, the extension to non-linear kernels is readily facilitated via a dual formulation posed in terms of weights α_i associated with each of the input data points. The weight α_i is non-zero only for support vectors, i.e., the cases where the error incurred is greater than ϵ, which determine the learned model. The model prediction for the target property of any

input object $\mathbf{x}$ is given by

$$f(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b = \sum_i \alpha_i y_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle + b = \sum_i \alpha_i y_i \langle K(\mathbf{x}_i, \mathbf{x}) \rangle + b.$$

3. Feature Engineering

Feature engineering [40] refers to the process of identifying an informative representation of the input space that facilitates high quality predictions for the desired property. It is an extremely critical component in most machine learning tasks such as classification/regression as it can help in alleviating problems resulting from high sparsity and high dimensionality, improve the generalization error of the learned models as well as reduce the computational time and storage resources. The choice of feature representation has often been shown to be more important for improving prediction accuracy than the choice of the learning algorithm and the size of training data, especially in case of a large feature space.

In most practical learning scenarios, feature engineering involves a combination of feature design, selection, grouping and transformation that map an object in the input space into a multi-dimensional vector. Of these tasks, the design or initial extraction of features is often specific to the application and has to be performed by a domain expert, for example, solvability of a trial can be assumed to be a function of certain numerical properties of the linear system and the parameters of the solver configuration being used. However, the optimal selection of informative features, latent factor identification via clustering/dimensionality reduction, and normalization of feature values are dependent only on the distributional properties of the original features and the target response, and are often performed using domain-independent statistical techniques, which we briefly discuss below.

Feature selection involves choosing a highly informative subset of the original features in order to prevent model over-fitting and to reduce the computational effort. Since

finding the optimal feature subset requires exponential time, most practical techniques either rely on feature ranking or sequential search. In feature ranking techniques, the predictive power of the individual features with respect to the target response is computed using various criteria such as mutual information and Pearson correlation, which is then used to sort the available features and the top k are chosen, with k often being determined by cross-validation. Though highly efficient, these ranking techniques do not account for dependence among features and often result in redundant features. Sequential search techniques such as forward-selection and backward-selection, on the other hand, progressively construct a subset of features by choosing or eliminating a single feature from the available pool based on their relative predictive power *given the current chosen set*. There also exist a number of other selection techniques based on wrappers, filters and embedded methods specific to various learning algorithms [40].

Creation of latent features by projecting the original features into a low dimensional space or grouping them into a small number of feature clusters is another effective approach that is often known to result in improved performance especially in case of high sparsity. This approach includes techniques such as principal component analysis [62] and non-negative matrix factorization [56], and feature clustering using the k-means or similar algorithms [60]. In applications, where the input objects can be represented as dyads (e.g., trials can be represented as pairs of linear systems and solver configurations), it has been shown that simultaneous clustering of the two dyadic dimensions [58] can yield highly predictive latent factors though their applicability is restricted to new objects over the known dyadic dimensions (i.e., known linear systems and solver configurations in case of trials). In contrast to feature selection techniques, latent factor methods often take into account the dependencies between the various features in a holistic fashion. However, they often operate in an unsupervised fashion independent of the target response and might occasionally result in latent factors that are not necessarily informative of the desired target property.

Suitable normalization of feature values [62] is another simple, yet critical task in obtaining a good feature representation. When the different feature values are not commensurate with each other, the parameter estimation steps in most learning algorithms require solving ill-conditioned matrices, resulting in significant numerical errors, and consequently, sub-optimal models. Hence, it is vital to ensure that the values of the various features are comparable and this is achieved using techniques such as linear scaling to unit range(linear transformation so that the maximum maps to 1 and the minimum to 0), z-score normalization (linear transformation to achieve zero mean, unit variance) and inverse logit transformation (non-linear transformation to map any real value into the range 0 to 1). There exist a number of normalization techniques in literature and the appropriate choice is often determined both by the data domain as well as the learning algorithm.

CHAPTER III

COMPARATIVE SOLVER PERFORMANCE EVALUATION

In this chapter, we describe various components of our practitioner-centric methodology for retrospective performance analysis and as well as an implementation of the performance analysis infrastructure. We also provide results of our performance evaluation on most of the popular general purpose preconditioners such as incomplete factorization, sparse approximate inverse, and algebraic multilevel schemes available in various black-box solver packages, which can be valuable to practitioners as well as software developers.

The remainder of this chapter is organized as follows: Section A provides an overview of the main contributions of the current work. Section B provides details of our experimental set up, including matrix collection, solver packages, and the preconditioners included in the study, and the hardware used. Section C gives an overview of the various metrics that are used to rank the performance of individual solver configurations as well as that of package-preconditioner combinations. In Section D, we present the detailed empirical evaluation methodology and results. We describe our performance analysis framework and a prototype implementation in Section E and provide concluding remarks in Section F.

A. Key Contributions

The main contributions of the current work are as follows:

1. Benchmarking Methodology

We introduce a methodology for a rigorous comparative evaluation of various preconditioners, including the use of some relatively simple but powerful metrics to facilitate a credible ranking of solver configurations (combinations of solver package, preconditioners, itera-

tive method, and solver and preconditioner parameters). Notable among these metrics are memory-time product and area under the curve for performance profiles (Section C).

2. Performance Analysis Infrastructure

We developed a semi-automated system for the collection, analysis, and visualization of relative performance data. It runs experiments and collects performance data (time, memory, error norm, etc.) for all combinations generated from a user specified set of linear systems, a set of hardware configurations (number of CPUs and memory limits), and sets of values of various solver and preconditioner parameters. This is achieved via a data collection unit composed of both serial and parallel driver programs and associated scripts for some widely used solver packages. Subsequently, the analysis and reporting unit of the system performs various comparative and sensitivity analyses within and across pre-specified groups of solver configurations using the collected performance data.

3. Extensive Empirical Evaluation

Using the above system, we evaluate a suite of preconditioners based on the incomplete factorization, sparse approximate inverse, and the algebraic multilevel schemes available in packages such as PETSc, Trilinos, Hypre, ILUPACK, and WSMP. We compare the robustness, speed, and memory consumption of these preconditioners on a set of benchmark problems and present results that can serve as guidance to practitioners. For packages that provide support for parallel execution, we collect and present performance data on multiple processors.

4. Good Default Configurations

For each combination of solver package and preconditioner, we identify the best overall choice of solver and preconditioner parameters on a suite of diverse problems. These obser-

vations can be used for choosing good *default* configurations for each package-preconditioner combination.

5. Fine-tuning of Parameters

In addition to determining good default configurations for each preconditioner implementation, we also study how sensitive the performance of a certain preconditioner is to parameter choices and which parameters have the greatest impact on performance. This analysis sheds light on the reliability of the default configuration and provides guidance for fine tuning the parameters to a specific problem or class of problems.

6. Choice of Package-preconditioner Combination

We simultaneously project the performance of all package-preconditioner combinations included in this study along three carefully chosen dimensions involving time, memory, and robustness to allow a ready comparison of the relative strengths of various implementations. We perform this comparison for the overall best set of parameters as well as for problem specific best set of parameters for each preconditioner implementation because their relative rankings can be different under the two scenarios.

7. Parallel Performance

We extend our empirical comparison of various preconditioner implementations to up to 64 CPUs. In addition to traditional performance metrics like parallel efficiency and speedup, we also study impact of parallelism on the choice of parameters.

B. Empirical Setup

In this section, we present the details of our experimental setup. These include an introduction to the solver packages, preconditioners and their parameters, descriptions of the test matrices and the hardware platform, and the specifics of our experimental approach.

1. Software Packages

We included the following packages in our study, which we believe are likely to be of most value to researchers and practitioners. These include well-established packages that include most commonly used preconditioners, as well as research packages with recently published general purpose preconditioners.

a. PETSc - Release Version 2.3.3-p0

PETSc [4], developed at Argonne National Laboratory, is implemented in C and has extensive documentation available for a new user with a plethora of informative examples demonstrating all the important aspects of the software package. The main goal of the PETSc project is to equip a user with the tools necessary for building scalable scientific applications. PETSc provides efficient implementations for all the commonly used Krylov subspace methods as well as fixed pattern and threshold based incomplete factorization preconditioners. Even though a wide range of preconditioning schemes are available via interfaces to external packages, we were not able to configure PETSc to use external packages (except BlockSolve95 [53]) due to lack of support for 64-bit compilation.

b. Trilinos - Release Version 8.0.3

Trilinos [45] was developed at Sandia National laboratories and its main focus is to provide parallel solvers and libraries in an object oriented framework. Trilinos is composed

of a number of self contained independently developed packages that could be used as a stand-alone application or in conjunction with other packages that support a minimal set of prerequisites for the interfaces. A suite of object oriented preconditioners are available in the Ifpack [74] and ML [30] packages. The AztecOO package, provides an object oriented interface to the popular Aztec solver library which contains implementations of the Krylov subspace methods. Ifpack supports a suite of Jacobi-style and incomplete factorization-based preconditioners whereas the ML package provides variants of algebraic multigrid type of preconditioners based on smoothed aggregation.

c. Hypre - Release Version 2.0.0

Hypre [1], developed at Lawrence Livermore National Laboratory, is designed primarily for the solution of large, sparse linear systems of equations on massively parallel computers. Hypre provides four different logical interfaces, namely, structured, semi-structured, finite element and linear algebraic. In addition to incomplete factorization based preconditioners (Euclid [51]), Hypre also has parallel implementations for approximate inverse based (ParaSails [17, 19]) and algebraic multigrid based (BoomerAMG [43]) preconditioners.

d. ILUPACK - Dev. Version 2.2

ILUPACK [57] was developed at Technische Universit$\ddot{a}$t Berlin and it contains implementations of inverse-based multilevel ILU preconditioners that controls the growth of the inverse triangular factors for both real and complex matrices. In addition to the standard static reordering schemes, it also includes the ARMS ordering schemes such as INDSET and ddPQ [72].

e. WSMP - Dev. Version 8.7

The Watson Sparse Matrix Package (WSMP) [37, 38], developed at IBM, contains serial and parallel sparse direct solvers as well as CG and GMRES solvers with new incomplete factorization based preconditioners [39].

2. Matrix Reordering

Previous research has shown that a suitable reordering of the coefficient matrix can reduce the memory requirement and potentially have a significant impact on the performance of many preconditioners [71]. Hence, wherever applicable, the matrices were first re-ordered using either the Reverse Cuthill Mckee ordering (RCM) [21] or the Nested Dissection ordering (ND) [31, 36]. In the case of ILUPACK, we used five built-in reordering schemes, namely, Nested dissection (ND), RCM, Approximate Minimum Fill (AMF), Independent set (INDSET), and permutation for diagonal dominance (ddPQ) [72].

3. Test Matrices

Since our objective is to detect general performance trends among different preconditioners and our analysis is purely empirical, it is imperative that the test matrices represent a spectrum of the problems for which computational simulations are extensively used. To this effect, we chose the matrices from a wide range of applications spanning fluid dynamics, sheet metal forming, electric circuit simulation, chemical process simulation, optimization etc. In order to narrow the scope of this empirical study, we consider only symmetric positive definite systems(SPD). The details of these SPD matrices are shown in Table II. Most of the matrices are obtained from the University of Florida collection [22] and the remaining ones are obtained from some of the applications that use WSMP [37].

Table II. SPD test matrices with their order (N), number of non-zeros (NNZ) and the application area of origin

Matrix	N	NNZ	Application
90153	90153	5629095	Sheet metal forming
af_shell7	504855	17588875	Sheet metal forming
Autodesk-big	1073724	84317460	Static stress analysis
audikw_1	943695	77651847	Automotive crankshaft modeling
bmwcra_1	148770	10644002	Automotive crankshaft modeling
ctu-1	1017397	74144859	Structural analysis
ctu-2	384012	28069776	Structural analysis
cfd1	70656	1828364	C.F.D. pressure matrix
cfd2	123440	3087898	C.F.D. pressure matrix
conti20	20341	1854361	Structural analysis
garybig	42459173	238142243	Circuit simulation
G3_circuit	1585478	7660826	Circuit simulation
hood	220542	10768436	Automotive
inline_1	503712	36816342	Structural engineering
kyushu	990692	26268136	Structural engineering
ldoor	952203	46522475	Structural analysis
msdoor	415863	20240935	Structural analysis
mstamp-2c	902289	70925391	Metal stamping
nastran-b	1508088	111614436	Structural analysis
nd24k	72000	28715634	3D mesh problems (ND problem set)
oilpan	73752	3597188	Structural analysis
parabolic_fem	525825	3674625	C.F.D. convection-diffusion
pga-rem-1	5978665	29640547	Power network analysis
pga-rem-2	1480825	7223497	Power network analysis
qa8fk	66127	1660579	F.E.M. stiffness matrix for 3D acoustic problem
qa8fm	66127	1660579	F.E.M. mass matrix for 3D acoustic problem
ship_003	121728	8086034	Structural analysis - ship structure
shipsec5	179860	10113096	Structural analysis - ship section
thermal2	1228045	8580313	Steady state thermal problem
torso	201142	3161120	Human torso modeling

4. Solvers, Preconditioners, and Parameters

We now describe the specific parameters that are used for the solvers and preconditioners used in our study. The preconditioners can primarily be classified into three broad classes, namely incomplete factorization, sparse approximate inverse, and algebraic multigrid. Although the current study's scope is limited to symmetric positive definite systems, we do use the GMRES solver if the resulting preconditioner is non SPD. Table III lists the specific preconditioners and the values of the tunable parameters that were experimented with. In all, 470 different combinations of solvers, preconditioners, and parameters were tried for the single processor case. The total number of solver configurations including all the serial and parallel cases added up to 2156.

Table III. Description of the package specific preconditioner parameters.

Package	Solver	Preconditioner	Orderings	Parameters
PETSc	CG	BlockSolve95	RCM, ND	-
		IC(k)	RCM, ND	*Level of fill:* 0, 1, 2 *Fill factor:* 3, 5, 8, 10
HYPRE	CG	IC(k)	RCM, ND	*Level of fill:* 0, 1, 2, 4, 6, 8
		ParaSails	RCM, ND NONE	*Number of levels:* 0, 1, 2 *Threshold:* 0, 0.01, 0.1, -0.75, -0.9 *Filter:* 0, 0.001, 0.05, -0.9
		BoomerAMG	RCM, ND NONE	*Maximum number of levels:* 25 *Number of aggressive coarsening levels:* 0, 10 *Coarsening schemes:* Falgout, HMIS, PMIS *Strong threshold:* 0.25, 0.5, 0.8. 0.9
Trilinos	CG	IC(k)	RCM, ND	*Level of fill:* 0, 1, 2, 4, 6, 8
		ML-SA	RCM, ND NONE	*Smoothers:* Symmetric Gauss-Seidel Chebyshev Polynomial, Incomplete Factorization *Smoother sweeps:* 1, 2, 3 *Coarsening Schemes:* Uncoupled, MIS Hybrid Uncoupled-MIS, ParMETIS
		ML-DD ML-DD-ML	RCM, ND NONE	- -
ILUPACK	CG	Multilevel ICT	RCM, ND, AMF INDSET, DDPQ	*Drop Tolerance:* 0.03, 0.01, 0.003, 0.001, 0.0005 *Inverse Norm Estimate:* 10, 25, 50, 75, 100
WSMP	Auto-select CG/GMRES	ICT	RCM, ND	*Drop Tolerance:* 0.01, 0.003, 0.001,0.0003 *Diagonal Perturbation:* OFF, ON (0.001) *Fill factor:* 2.5, 3.3, 4.1, 4.9

5. Hardware Specifics

The experiments were conducted on up to 64 processors on an IBM HPC cluster 1600, based on the Power5+ IBM processor running the 64-bit version of AIX (version 5.3). Each of the p5-575 nodes on the cluster has sixteen 1.9 Ghz Power5+ processors. A memory limit of 24 GB per node and a wall time limit of 4 hours was used for each empirical trial involving a single matrix and a solver configuration.

All the packages were compiled using IBM compilers xlf (Fortran), xlc (C) or xlC (C++) in 64-bit mode with the -O3 optimization flag. The Engineering Scientific Subroutine library (ESSL) was linked in to provide BLAS routines. The page size for text and data was set to 64 KB.

6. Experimentation Methodology

We now describe our methodology for conducting the experiments and collecting the performance data. In order to make the evaluation as uniform as possible, we adhered to the following rules for all the experiments.

- Diagonal scaling is performed on the linear system before starting the solution process.
- A right hand side vector of all 1's is used and the initial guess of the solution for the iterative process is always the zero vector.
- We use right preconditioning since it is the default for all the packages except PETSc and it allows us to have a uniform convergence criteria based on the true residual for all the experiments. This choice of right preconditioning was also influenced by a similar study conducted on ILU preconditioners [18].

- The iterations are stopped when the number of iterations reaches 1000 or when the relative residual norm drops below 10^{-5}.
- When using more than one processor, ParMETIS [54] is used to partition the rows of the matrix and distribute them appropriately. We then reorder the local matrix on each processor according to the specified reordering scheme.
- A trial is considered to have failed if the total time is above 4 hours, or the memory consumption per node exceeds 16 GB while using up to 8 processors per node and 24 GB when using all 16 CPUs in a node, or the final relative error norm exceeds 0.02.
- A trial is also considered to have failed if its performance on a desired metric is more than one order of magnitude worse than that of the best performing trial. More details on this can be found in Section C.4.

7. Performance Metrics

For each successful trial, we measured and recorded the following performance metrics.

a. Time Taken

This is the total time in seconds required for both creating the preconditioner and actually solving the linear system. We measure this using timing calls before and after the appropriate routines. In the case of multiple processor runs, the reported time is the maximum among all the processors.

b. Memory Usage

This is the amount of memory in bytes allocated on the heap during the preconditioner creation phase. In the case of multiple processor runs, we compute the *cumulative* memory

usage across all MPI processes. If GMRES is used as the iterative solver, we also add the memory required for storing the subspace vectors to the total observed memory.

c. Relative Error Norm

This is measured as the ratio of L2 norm of the final error to that of the initial error. For computing the error, we use an approximation of the actual solution obtained using the WSMP direct solver.

d. Memory Time Product

In order to determine good solver configurations that perform best over all the problems we introduce a simple intuitive performance criterion. Specifically, we use the product of total solution time and the memory required for storing the coefficient matrix and preconditioner as our primary performance criteria. Henceforth, we will refer to this quantity as the *Memory-Time-Product* (MTP). The choice of MTP is motivated by our observation that both computation time and memory use appear to be inadequate measures of the quality of a preconditioner, when considered individually. For most preconditioners, there is a range of parameter choices in which there is a trade-off between solution time and memory consumption, although it is possible to make parameter choices that increase or decrease both time and memory simultaneously. The optimum operating point of a preconditioner for a given problem lies in a trade-off zone. As reported in literature [34] and confirmed by our own experiments, direct solvers can result in the overall fastest time, albeit at the cost of a significantly high memory consumption (Section D.4). Therefore, a preconditioner could simply emulate the direct solver and emerge as the fastest preconditioner. At the other extreme, preconditioners such as Jacobi, Gauss-Seidel, or IC(0), consume very little memory, but can take an impractically large number of iterations to converge. As a result, judging the quality of preconditioners based solely on their time or memory requirements simply

yields winners that are extreme cases and are of little practical interest.

C. Benchmarking Methodology

In this section, we present the benchmarking methodology that we use to evaluate the preconditioners and the relative performance of their various configurations resulting from different choices of parameters and other user selectable options. Our methodology is based on performance profiles [24], which we augment with some other metrics described later in this section.

1. Solver Configurations and Performance Data

A solver configuration is a solver and preconditioner implementation with a given set of values for all parameters and user selectable options. For example, Hypre's CG solver and its ParaSails preconditioner, with RCM ordering, 2 levels of fill, a threshold of 0.01, and a filter value of 0.05 is one solver configuration, PETSc's CG solver and BlockSolve95 preconditioner with ND ordering is another. We denote the set of all solver configurations by $\mathcal{S}$. Let $|\mathcal{S}| = m$. The set $\mathcal{S}$ used in our study was constructed by using all feasible combinations in Section 4, with $m = 470$ for the single processor case. We denote by $\mathcal{P}$ the set of linear systems/problems to be solved with $|\mathcal{P}| = n = 30$ in our study. A trial is the application of a solver configuration to a problem. We performed $m \times n = 14100$ trials for the single processor case.

Let μ represent any performance measure that takes a specific value for each evaluation trial. Examples of performance measures include time taken, memory usage, memory-time product, etc. The $n \times m$ trials result in an $n \times m$ matrix μ of performance data for each performance metric, where the element (p, s) corresponds to the performance $\mu_{p,s}$ of solver configuration s with respect to problem p. The performance values $\mu_{p,s}$ may not always be

well-defined due to solver configuration failure and other practical limitations. Without loss of generality, we assume that lower values of performance values are desirable and therefore, we represent ill-defined values corresponding to solver configuration failures with a very high value (∞).

The solver configurations are partitioned into groups to facilitate the analysis of the performance data collected through the trials. Each solver configuration belongs to one or more (possibly overlapping) groups. For example, all solver configurations for the Hypre package can be considered to belong to configuration group $\mathcal{C}_1$, all solver configurations using the BoomerAMG preconditioner can be considered to belong to another configuration group $\mathcal{C}_2$, and all solver configurations resulting from various choices of ordering and coarsening schemes for Hypre BoomerAMG can be considered to belong to the configuration group $\mathcal{C}_3$.

2. Performance Ratios

Given the data for a particular performance measure, it is straightforward to compare the effectiveness of the methods with respect to a single problem. Specifically, we assume that methods with lower performance values are better. However, comparing methods based on their collective performance requires calibration across the problems. A natural way to compare the solver configurations in a configuration group $\mathcal{C}$ would be to consider the normalized performance values $r_{p,s}(\mathcal{C})$, otherwise known as the performance ratios of the methods for each problem:

$$r^{\mu}_{p,s}(\mathcal{C}) = \frac{\mu_{p,s}}{\min\limits_{s' \in \mathcal{C}} \mu_{p,s'}} \ ,$$

which is the ratio of the actual performance value of solver configuration s to the best (least) value over all solver configurations for the problem p. Note that $r_{p,s}(\mathcal{C}) \geq 1$ for all (p, s) and is equal to 1 for at least one solver configuration $s \in \mathcal{C}$ for each problem p, as

long as at least one $s \in \mathcal{C}$ is able to solve the problem p.

It seems reasonable that the average performance ratio $\eta_s(\mathcal{C})$ of a configuration $s \in \mathcal{C}$ would be a fair indicator of the effectiveness of the configuration s with respect to that performance metric μ, where

$$\eta_s^\mu(\mathcal{C}) = \frac{1}{n} \sum_{p=1}^{n} r_{p,s}^\mu(\mathcal{C}).$$

In practice, however, η_s is often not very useful since a single failure for a solver configuration s can make its average performance ratio η_s ill-defined, making it difficult to compare the different methods. One simplistic solution for handling this issue is to only consider problems that have well defined performance ratios, but this would not be fair to methods that actually solve the harder problems not solved by all the methods. A more principled approach is to compare the performance of the methods both in terms of the number of problems solved as well as average performance ratio directly using the distribution of the performance ratios. To achieve this, we use the notion of performance profiles, which we now describe.

3. Performance Profile

A performance profile [24] is a plot of the cumulative distribution of the performance ratios. Let $\rho_s^\mu(\tau)$ denote the cumulative distribution of the performance ratios of a solver configuration s with respect to the measure μ:

$$\rho_s^\mu(\tau) = \frac{1}{n} |r_{p,s}^\mu(\mathcal{C}) \leq \tau|.$$

$\rho_s^\mu(\tau)$, therefore, denotes the fraction of the problems that the configuration $s \in \mathcal{C}$ can solve with performance that is within a factor of τ of the best performance for each problem.

Table IV shows hypothetical performance data (say, run time in seconds) for a set of

Table IV. Hypothetical performance data with three solver configurations (s_1, s_2, s_3), three configuration groups ($\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$), and three problems (p_1, p_2, p_3). Solver configuration failures are represented with ∞.

Solver Configurations	Configuration Groups	$\mu_{p,s}$			$r^{\mu}_{p,s}(\mathcal{C}_1)$		
		p_1	p_2	p_3	p_1	p_2	p_3
s_1	$\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$	7	2	3	3.5	1	1
s_2	$\mathcal{C}_1, \mathcal{C}_2$	3	4	6	1.5	2	2
s_3	$\mathcal{C}_1, \mathcal{C}_3$	2	∞	5	1	∞	5/3

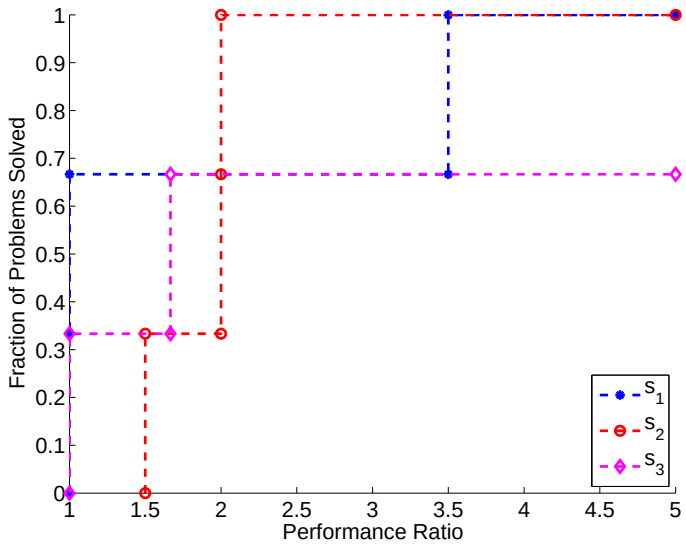


Fig. 1. Performance profile curves for solver configurations in group $\mathcal{C}_1$.

three solver configurations and three problems. Figure 1 shows the performance profiles for the configuration group $\mathcal{C}_1$ shown in Table IV. The performance profile plot readily reveals information that may not be apparent from average performance ratios even when they are well defined. For example, the point (1.67, 0.67) on the plot for configuration s_3 denotes that this configuration was able to solve 67% of the problem within 1.67 times the best running time for any configuration for these problems. Similarly, the point (2.0, 1.0) on the plot for configuration s_2 denotes that this configuration was able to solve all (100%) of the problems consuming at most twice the best running time for each problem. Additionally, performance profiles enable one to compare easily methods under circumstances where the user specifies an additional success threshold θ on the performance ratio; i.e., any solver configuration that results in performance value that is θ times greater than that of the best value is considered a failure. In the example in Table IV and Figure 1, s_1 is clearly the best method for $\theta = 1.3$, followed by s_3 and s_2. On the other hand, for $\theta = 4$, both s_1 and s_2 are equally good since they both solve all the three problems and that too with identical average performance ratio of $11/6$. Comparing with s_3 is somewhat tricky since it solves only two out of the three problems, and has a lower average performance ratio of $8/6$ for the solved problems.

4. Solver Quality Measure

When comparing a large number of performance profile curves, it may not be possible to visually determine the relative ordering or to pick the best solver configuration. The situation is usually exacerbated further due to different failure rates of the various configurations. To address this issue and to eliminate human intervention in the case of a large number of solver configurations, we propose to use the area under the performance profile curves for ranking the various solver configurations. The area under the curve (*AUC*) provides a longitudinal summary of multiple assessments at all possible performance ratios of

interest. For a performance ratio threshold of θ, the only τ values of interest are those from 1 to θ. Therefore, the relevant area under the performance profile curve for a configuration s with respect to the configuration group $\mathcal{C}$ $AUC_s^\mu(\mathcal{C}, \theta)$ is the area under the cumulative distribution curve up to θ, which is given by

$$AUC_s^\mu(\mathcal{C}, \theta) = \theta - \frac{1}{n}\sum_{p \in \mathcal{P}} \min(\theta, r_{p,s}^\mu(\mathcal{C})).$$

This formulation, ignores all the performance ratios of solver-configurations outside the range of the threshold θ and effectively considers those as failures. For the example in the performance profile figure, $AUC_{s_1}^\mu(\mathcal{C}_1, 5.0)$ = 3.17, $AUC_{s_2}^\mu(\mathcal{C}_1, 5.0)$ = 3.17, and $AUC_{s_3}^\mu(\mathcal{C}_1, 5.0)$ = 2.44. We also notice that *AUC* of s_3 is comparable to that of s_1 and s_2 for smaller values of θ, but becomes progressively worse as the threshold increases.

Note that for the special case where there are no solver configuration failures and all performance ratios are less than or equal to θ, the area under curves is directly related to average performance ratios. A choice of θ is critical in the *AUC*-based comparison since this can significantly affect the *AUC* and thereby, the relative ranking of the different methods. In the current study, we choose this threshold to be equal to 10. In other words, we assume that a trial that results in performance that is more than an order of magnitude worse than the best performance for a given problem is effectively a failure. This is in addition to the failure criteria described in Section B.6.

5. Configuration Group Quality Measures

So far in this section, we have discussed metrics for comparing the relative performance of individual solver configurations. It is often desirable to compare different configuration groups. For example, for a given problem set $\mathcal{P}$, it would be interesting to be able to objectively compare the implementations of different preconditioners in different packages.

In our study, we measure and compare the following three metrics for configuration groups comprised of various configurations of a given package-preconditioner combination.

a. Problem Independent Best Performance

For a configuration group $\mathcal{C}$, we define $PIB^{\mu}(\mathcal{C})$ as the solver configuration $s' \in \mathcal{C}$ such that $AUC^{\mu}_{s'}(\mathcal{C}, \theta)$ is the maximum among all $AUC^{\mu}_{s}(\mathcal{C}, \theta)$ for all $s \in \mathcal{C}$. In other words, $PIB^{\mu}(\mathcal{C})$ is the configuration that results in overall best performance with respect to metric μ for a given problem set $\mathcal{P}$. Therefore, if $\mathcal{C}$ represents a package-preconditioner combination, then the values of the various parameters and user selectable options corresponding to $PIB^{\mu}(\mathcal{C})$ are logical choices for default parameters for $\mathcal{P}$. The performance of the configuration $PIB^{\mu}(\mathcal{C})$ can be considered representative of the performance of configuration group $\mathcal{C}$, and can be used to compare different groups by using the performance profiles and the *AUC* metric.

b. Problem Specific Best Performance

An alternative to using the performance of $PIB^{\mu}(\mathcal{C})$ to represent the performance of configuration group $\mathcal{C}$ is to represent it by its problem-specific best performance. Formally, we define the problem-specific best performance $\mu^{PSB}_{p,\mathcal{C}}$ of configuration group $\mathcal{C}$ for problem p as the best performance value among all the solver configurations in $\mathcal{C}$; i.e.,

$$\mu^{PSB}_{p,\mathcal{C}} = \min_{s \in \mathcal{C}} \mu_{p,s}.$$

Note that $\mu_{p,\mathcal{C}}$ is an aggregation of the performance values of the member solver configurations. This is in contrast to $\mu_{p,PIB^{\mu}(\mathcal{C})}$, which directly considers the performance values of a particular winning member solver configuration.

6. Hardware Configurations

The performance metrics $\mu_{p,s}$ obtained for each problem p and solver configuration s is also a function of the hardware configuration. Let $\mathcal{H} = \{h\}$ denote the set of all hardware configurations on which performance data is obtained from. For the current study, $h \in \{1, 2, 4, 8, 16, 32, 64\}$ where each number consists of the number of processors used for each trial. We study the performance with respect to one value of h at a time.

7. Parallel Performance

Most solver packages included in this study are designed to solve large sparse systems in highly parallel environments. In the parallel case, users may be interested in additional performance metrics other than those studied in the context of a single processor. For example, a user might be interested in knowing how the relative performance of the solvers observed in a serial environment changes in various parallel settings. We consider each multi-processor run to be part of a different hardware group. The various solvers are evaluated in each of these hardware groups separately and the *AUC* of performance profiles are used to study the behavior of the solvers under various hardware configurations. An important performance metric in a parallel scenario is the efficiency of the respective parallel implementations. Efficiency is computed as $\epsilon = Time^{sp}/(np \times Time^{np})$ where $Time^{sp}$ is the best sequential time and $Time^{np}$ represents the time observed for np processors. A relatively high efficiency for large processors could either suggest that the solver can be parallelized efficiently or that the serial implementation is not optimal. Similarly, a low value of efficiency could suggest the existence of expensive sequential components or a poor parallel implementation.

D. Results

In this section, we present the results of our empirical evaluation. In Section D.1, we analyze the performance of various solver configurations within configuration groups comprised of package-preconditioner combinations and report the best configuration for each group over all the problems with respect to time, memory, and memory-time product (MTP). In addition, we also discuss the effect of certain important parameters on the memory and time performance and analyze the variation in performance of the overall best parameter combinations in a multi-processor scenario with increasing number of processors. Section D.2 presents the effect of parameter fine-tuning on memory and time within each configuration group in a multi-processor scenario. This is followed by a comparison of the default and problem-specific best performance of the various configuration groups along with the direct solver in Section D.4. Finally, in Section D.5, we look at the parallel efficiency trends of the various package-preconditioner combinations.

1. Performance Within Configuration Groups

For the purpose of the analysis in this section, we divided the set of all solver configurations into configuration groups, where each group represents a package-preconditioner combination. For each configuration group and hardware configuration, we identified the solver configurations that resulted in the best overall performance with respect to time, memory, and memory-time product (MTP) over all the problems in the test suite. The best performance was determined using the area (AUC) under the performance profile (PP) curves, as discussed in Section B.7. We also provide detailed analyses of the effects of various parameters for suitable subsets of the serial configurations by means of PP curves. We chose these figures on a case-by-case basis depending on the interesting performance trends that we found for individual preconditioner implementations.

For the sake of brevity, we use acronyms to describe the parameter choices in both the tables and the legends of the figures in this section. A complete list of these acronyms and their expansions is shown in Table V, which the readers might find useful to refer to while interpreting the subsequent tables and figures with experimental results.

Table V. List of acronyms used to denote various parameter choices.

Parameter Name	**Values**	**Acronyms**
Level of fill	0, 1, 2, 4, 6 ,8	LF0, LF1, LF2, LF4, LF6, LF8
IC(K) fill factor	1, 3, 5, 8, 10	F1, F3, F5, F8, F10
Max. additional nonzeros/row	∞	NzINF
Drop tolerance	0.03, 0.01, 0.003, 0.001, 0.0003, 0.0005	DT3e-2, DT1e-2, DT3e-3, DT1e-3, DT3e-4, DT5e-4
Inverse norm estimate	10, 25, 50, 75, 100	IE10, IE25, IE50, IE75, IE100
Number of ParaSails levels	0, 1, 2	PLev0, PLev1, PLev2
Threshold	0, 0.01, 0.1, -0.75, -0.9	Th0, Th.01, Th.1, Th-.75, Th-.9
Filter	0, 0.001, 0.05, -0.9	Flt0, Flt.001, Flt.05, Flt-.9
BoomerAMG coarsening schemes	Falgout, Hybrid MIS, Parallel MIS	FALG, HMIS, PMIS
Strong threshold	0.25, 0.5, 0.7, 0.9	ST.25, ST.5, ST.7, ST.9
ML preconditioner type	Classical SA, SA based 2-level domain decomp, 3-level algebraic domain decomp.	SA, DD, DD-ML
Smoothers	Symmetric Gauss-Seidel, Chebyshev, Incomplete factorization	SGS, CBY, IFPACK
Smoother sweeps	1, 2, 3	SS1, SS2, SS3
ML coarsening schemes	Uncoupled, MIS, Hybrid uncoupled-MIS, ParMETIS	UC, MIS, UCMIS, PMETIS
WSMP fill factor	2.5, 3.3, 4.1, 4.9	F2.5, F3.3, F4.1, F4.9
WSMP diagonal shift	-1, 0.001	SHIFT-OFF, SHIFT-ON

a. Level-based Incomplete Factorization IC(k)

The PETSc, Trilinos, and Hypre packages include implementations of the IC(k) preconditioner. We have also included the BlockSolve95 preconditioner in our study. In the case of PETSc, experiments were conducted with values of k = 0, 1, and 2. Hypre and Trilinos IC(k) implementations consume reasonable time and memory for levels of fill higher than 2; and therefore, we included values of k = 4, 6, and 8 in our experiments. For nonzero values of k, we experimented with fill factors of 3, 5, 8, and 10 for PETSc. Trilinos IC(k) does not provide a user controlled parameter for controlling the fill factor and the default setting in Hypre IC(k) is ∞, i.e., no limit. Table VI shows the overall best configurations

Table VI. Solver configurations that resulted in the best memory, time, and MTP performance profile area for IC(k) preconditioners in PETSc, BlockSolve95, Hypre, and Trilinos for various numbers of processors (shown in parenthesis). Expansions of the parameter acronyms can be found in Table V.

Preconditioner	**Memory Winner**	**Time Winner**	**MTP Winner**
PETSc IC(k)	CG, RCM, LF0, F1 (1 2 8 16 32 64) CG, ND, LF0, F1 (4)	CG, RCM, LF0, F1 (1 2 4 8 16 32 64)	CG, RCM, LF0, F1 (1 2 4 8 16 32 64)
PETSc BlockSolve	CG, RCM (2 4 8 16 32 64) CG, ND (1)	CG, RCM (1 2 16 32 64) CG, ND (4 8)	CG, RCM (1 2 4 16 32 64) CG, ND (8)
Trilinos IC(k)	CG, RCM, LF4 (1 2) CG, RCM, LF6 (4 8 16 32 64)	CG, RCM, LF8 (1 2 4 8 16 32 64)	CG, RCM, LF4 (2) CG, RCM, LF6 (32 64) CG, RCM, LF8 (1 4 8 16)
Hypre IC(k)	CG, RCM, LF1, NzINF (1 64) CG, RCM, LF2, NzINF (2 4 16 32) CG, ND, LF1, NzINF (8)	CG, RCM, LF1, NzINF (1) CG, RCM, LF2, NzINF (2 64) CG, ND, LF1, NzINF (8) CG, ND, LF2, NzINF (4 16) CG, ND, LF6, NzINF (32)	CG, RCM, LF1, NzINF (1 2 16 32 64) CG, ND, LF1, NzINF (4 8)

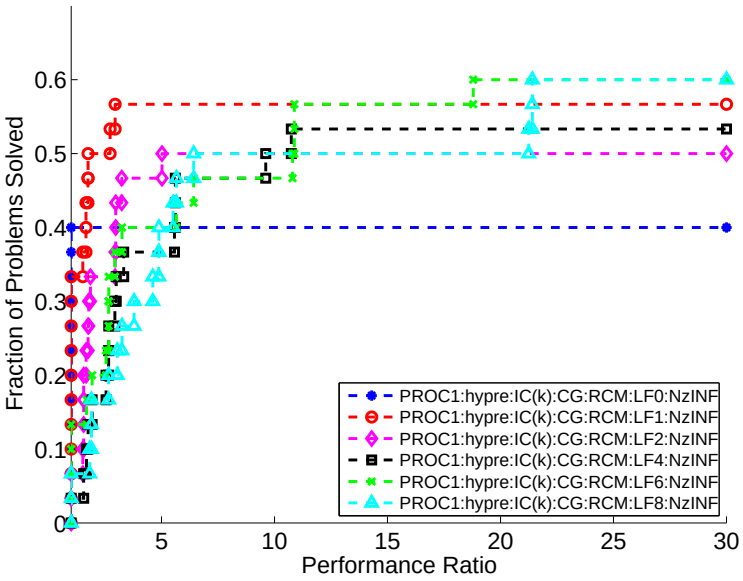
with respect to time, memory, and MTP for various hardware configurations (number of processors) for all the IC(k) implementations. As is often the case with preconditioners, we noticed that the different implementations of even the relatively straightforward IC(k) had very different performance characteristics and responses to values of their parameters.

Hypre IC(k): Our experimentally determined overall best parameters for the Hypre IC(k) preconditioner for different numbers of processors are shown in Table VI. We also investigated the impact of levels of fill on performance. Figure 2 shows the memory and time profile curves for various levels of fill with RCM ordering. As expected, increasing the number of levels of fill from 0 to 1 results in increased robustness and improved run times. However, the performance drops as the number of levels is increased beyond one because the improvement in quality of the preconditioner cannot compensate for the increased memory and time required for higher levels of fill.

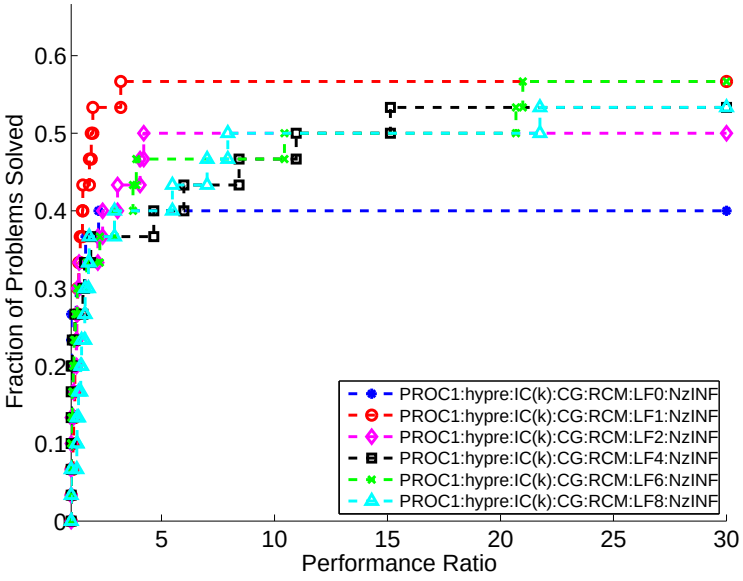
PETSc IC(k): The overall best parameter configurations shown in Table VI indicate that for PETSc IC(k), level of fill $k = 0$ with RCM ordering resulted in the best overall performance. The performance showed little variation in response to changing the fill factor for low values of fill factor. High fill factors resulted in excessive memory consumption. Figure 3 shows the time and memory profiles for various levels of fill with RCM ordering and a fill factor of 5. Unlike Hypre IC(k), both the memory and the time performance of PETSc IC(k) deteriorates rapidly with even a modest increase in level of fill beyond 0.

PETSc BlockSolve95: Ordering is the only user controlled parameter in BlockSolve95; however, we observed almost no difference in its performance between RCM and ND orderings.

Trilinos IC(k): Figure 4 shows the time and memory profiles for various levels of fill with RCM ordering for the Trilinos IC(k) preconditioner. The memory profile indicates that higher levels of fill solve more problems at the expense of using slightly more memory. However, the time profiles reveal that the higher levels of fill result in more effective

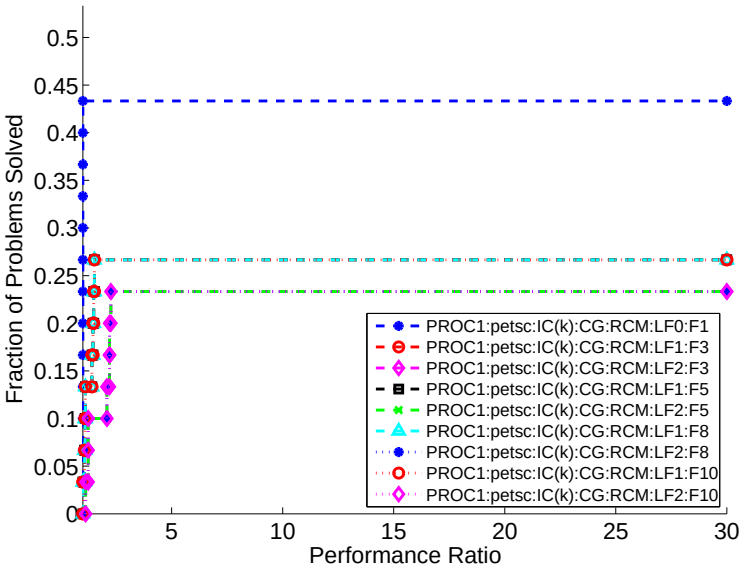


(a) Memory performance profile

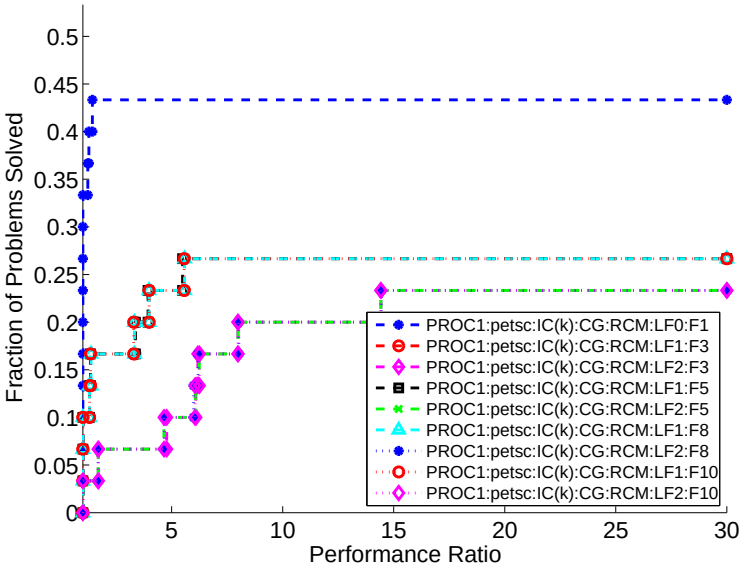


(b) Time performance profile

Fig. 2. Serial memory and time profile curves for the Hypre IC(k) solver configurations with RCM ordering, an unlimited number of additional nonzeros per row, and various levels of fill.



(a) Memory performance profile



(b) Time performance profile

Fig. 3. Serial memory and time profile curves for the PETSc IC(k) solver configurations with RCM ordering, fill factor of 5, and various level of fill parameters.

preconditioners that result in faster solution times. This is in stark contrast to the IC(k) implementations of PETSc and Hypre.

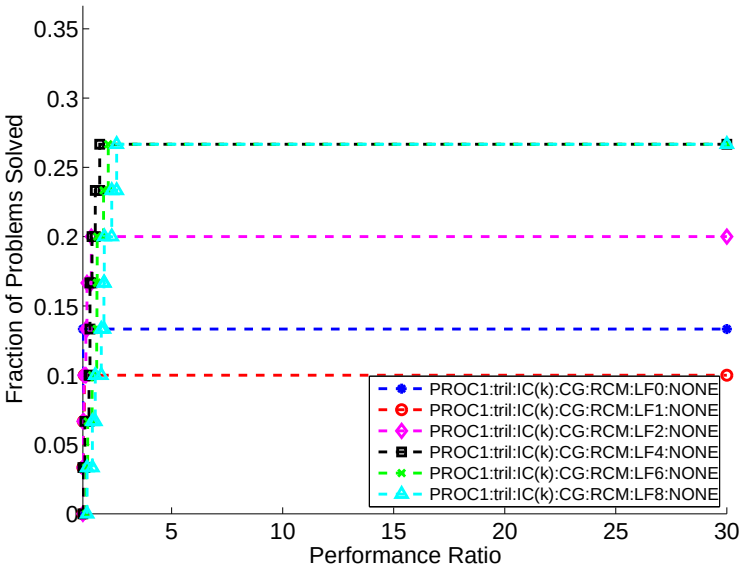
Comparison of Hypre, PETSc, and Trilinos IC(k): Figure 5 shows a comparison of BlockSolve95 and the configurations of Hypre, PETSc, and Trilinos IC(k) that resulted in the best overall MTP performance as measured by the AUC metric. PETSc IC(k) is most memory efficient because its best configuration has $k = 0$; however, it is less robust than Hypre. For the problems that they both can solve, the overall best configurations of PETSc and Hypre are equally fast. However, the best Hypre IC(k) configuration is able to solve more problems than the best PETSc IC(k) configuration.

b. Threshold-based incomplete Cholesky

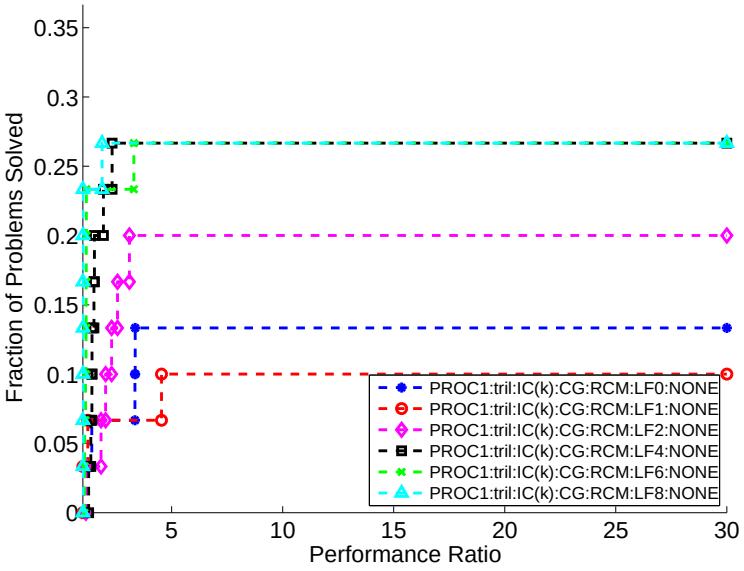
Table VII. Configurations that resulted in the best memory, time, and MTP performance profile area for the ILUPACK MLICT and WSMP ICT preconditioners.

Preconditioner	Memory Winner	Time Winner	MTP Winner
ILUPACK MLICT	CG, AMF DT1e-2, IE10	CG, AMF DT3e-3, IE75	CG, RCM DT3e-02, IE75
WSMP ICT	AUTO, ND, DT3e-3 F3.3, SHIFT-ON	AUTO, RCM, DT1e-3 F4.9, SHIFT-OFF	AUTO, RCM, DT3e-3 F4.9, SHIFT-ON

We studied the ICT preconditioners of WSMP and ILUPACK in detail. We do not report the results of ICT preconditioner implementations of other packages due to their serious performance and robustness problems. For ILUPACK MLICT, we tried five different built-in reordering schemes (RCM, AMF, INDSET, PQ, and METISN), five different values for drop tolerance (0.03, 0.01, 0.003, 0.001, 0.0005), and five different values of the norm of inverse estimate (10, 25, 50, 75, 100). In the case of WSMP ICT, we tried two ordering schemes (RCM, ND), four values of drop tolerance (0.01, 0.003, 0.001, 0.0003), four values of fill factor (2.5, 3.3, 4.1, 4.9), with and without diagonal perturbation. Table VII shows the solver configurations that resulted in the best memory, time, and MTP

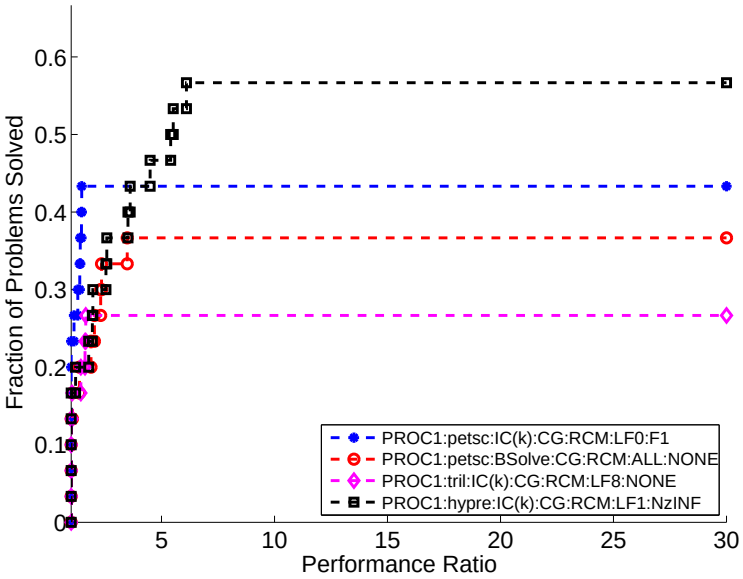


(a) Memory performance profile

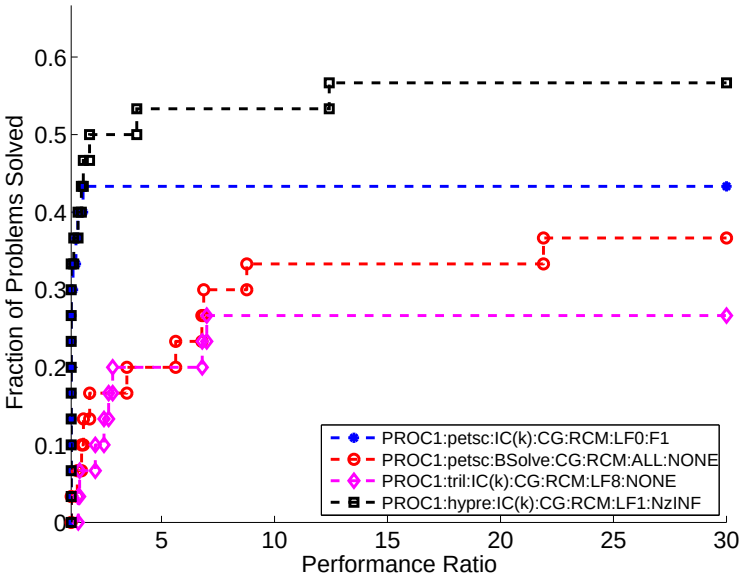


(b) Time performance profile

Fig. 4. Serial memory and time profile curves for the Trilinos IC(k) solver configurations with RCM ordering and various level of fill parameters.



(a) Memory performance profile



(b) Time performance profile

Fig. 5. Memory and time performance profile curves for the overall best configurations for Trilinos IC(k), PETSc IC(k), PETSc BlockSolve95, and Hypre IC(k) in the serial case.

profile areas for ILUPACK MLICT and WSMP ICT.

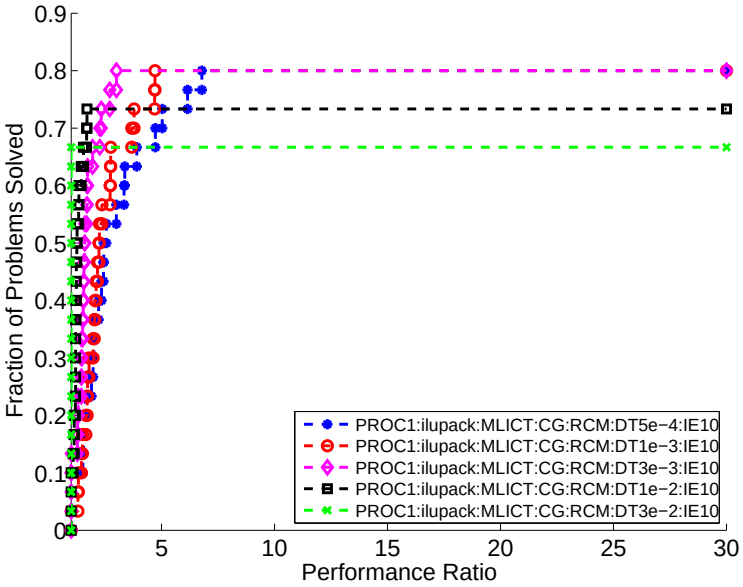
ILUPACK MLICT: In Figure 6, we study the effect of drop tolerance for RCM ordering and a low inverse norm estimate value of 10. We observe that the configuration corresponding to moderately low drop tolerance value of 0.003 is the most robust but requires significantly more memory resources than higher drop tolerance values. The same drop tolerance has the best time profile area too. Figure 7 shows that even for a high value of inverse norm estimate such as 100, the best results are obtained with the same drop tolerance value of 0.003.

WSMP ICT: Figure 8 shows the performance profile curves corresponding to the various drop tolerance values for RCM ordering, diagonal perturbation of 0.001, and fill factor of 4.9 for WSMP ICT. Drop tolerance values of 0.001 and 0.003 seem to offer the best balance between robustness and memory and time consumption. Figure 9 shows the effect of varying the ordering and diagonal perturbation for the best MTP values of drop tolerance and fill factor (0.003 and 4.9, respectively). The use of diagonal perturbation results in more robust solver configurations. While using diagonal perturbation, ND performed slightly better than RCM, and without it, RCM performed better.

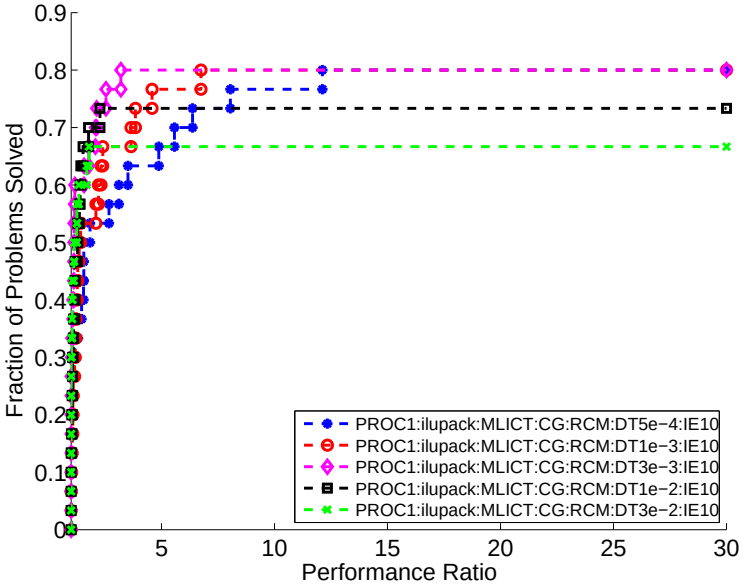
c. Algebraic Multigrid Methods

Multigrid preconditioners typically have a large number of parameters that need to be fine tuned. We used the default values suggested in the user manuals [1, 30] for a majority of the parameters, and varied a few key ones, based on the suggestions from the authors of Hypre and Trilinos. Table VIII shows the best configurations for this class of preconditioners.

Hypre BoomerAMG: We experimented with the ordering, coarsening scheme, maximum number of levels for aggressive coarsening, and strong threshold for the BoomerAMG preconditioner. In Figures 10 and 11, we show how the coarsening scheme and aggressive coarsening levels affect the performance. These figures show the results with the RCM

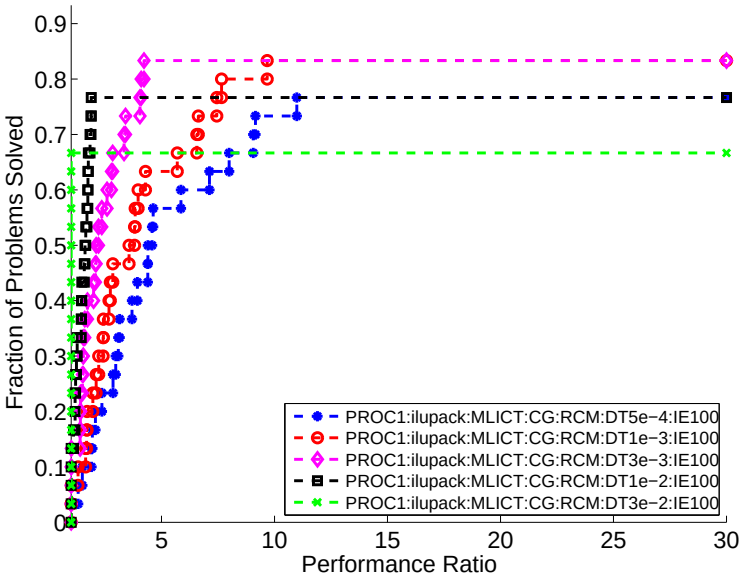


(a) Memory performance profile

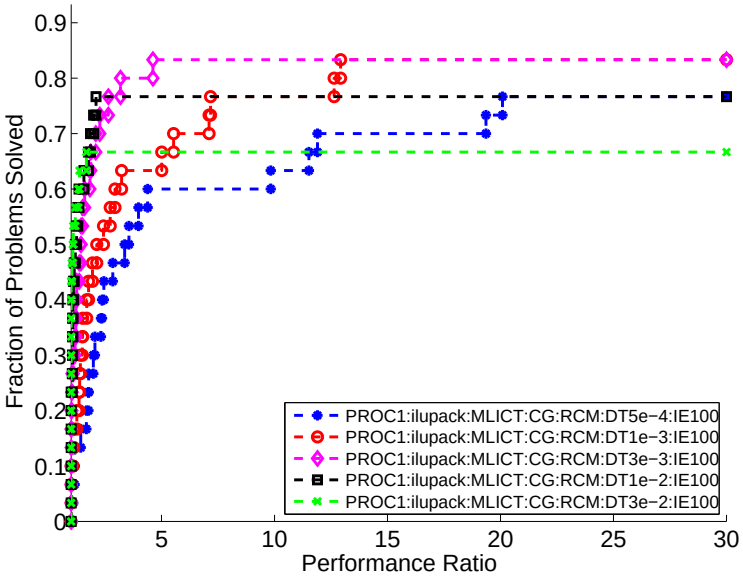


(b) Time performance profile

Fig. 6. Memory and time performance profile variations for varying values of drop tolerance (DT) in the case of ILUPACK MLICT with RCM ordering and inverse norm estimate value of 10.

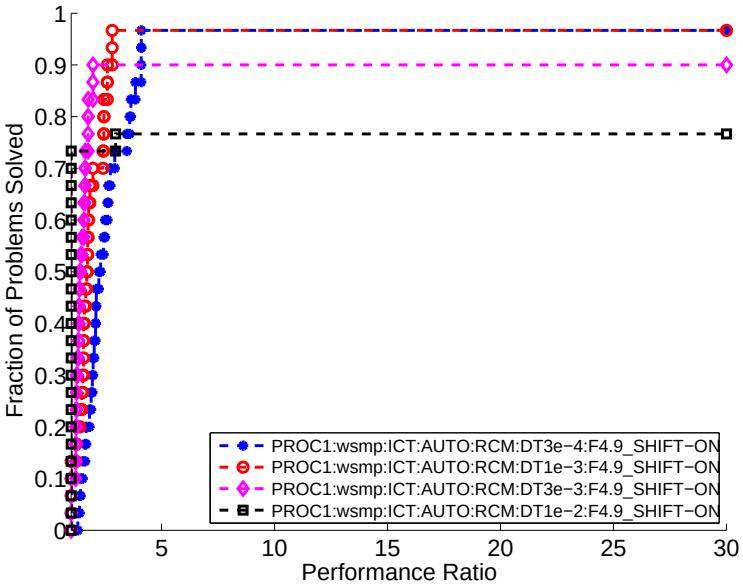


(a) Memory performance profile

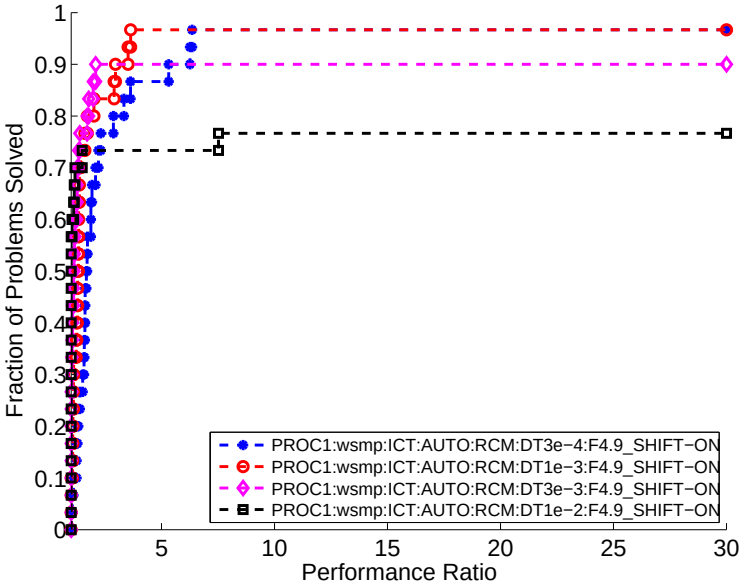


(b) Time performance profile

Fig. 7. Memory and time performance profile variations for varying values of drop tolerance (DT) in the case of ILUPACK MLICT with RCM ordering and inverse norm estimate value of 100.

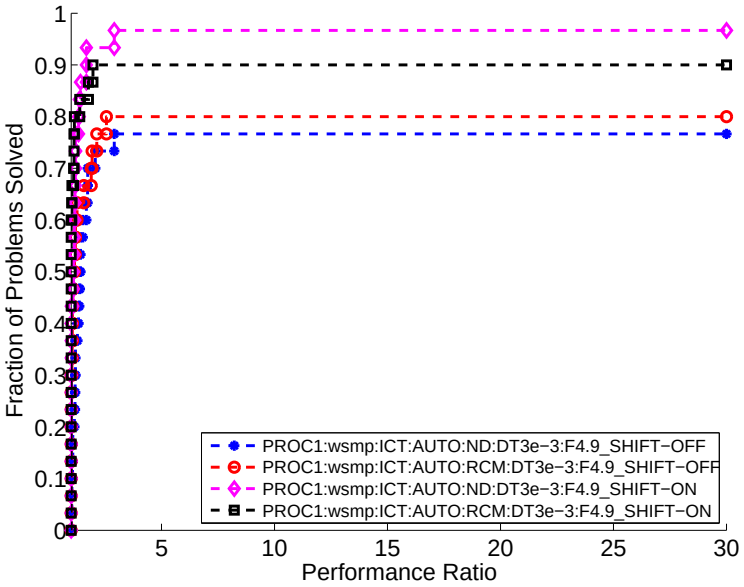


(a) Memory performance profile

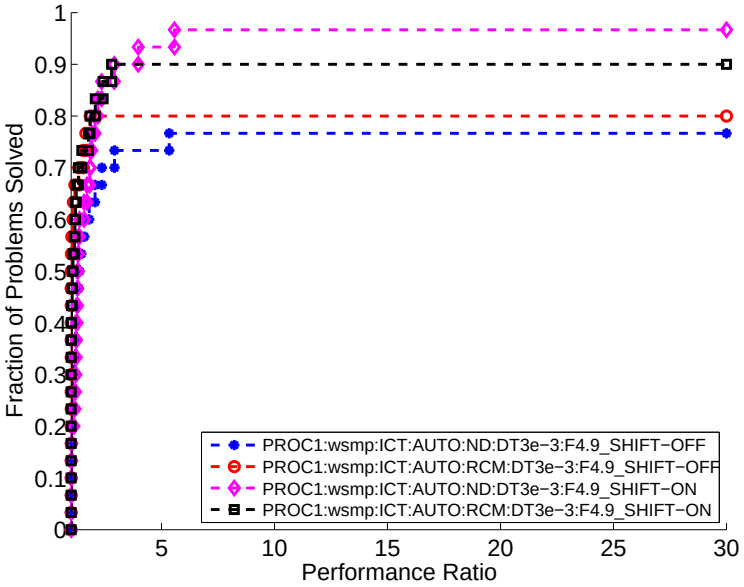


(b) Time performance profile

Fig. 8. Memory and time performance profile variations for varying values of drop tolerance (DT) in the case of WSMP ICT with RCM ordering, diagonal perturbation, and fill factor value of 4.9.



(a) Memory performance profile

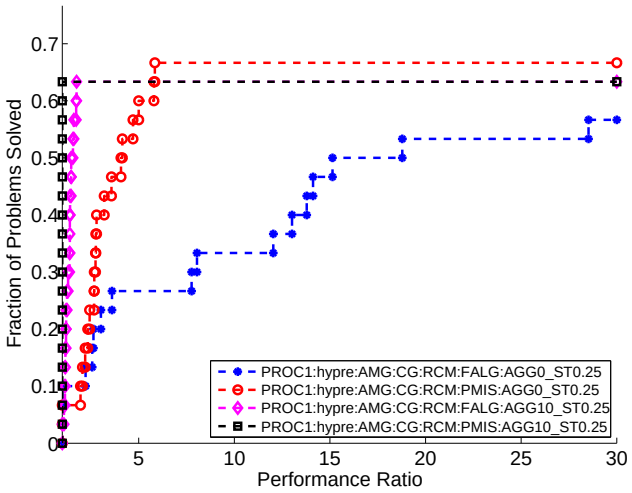


(b) Time performance profile

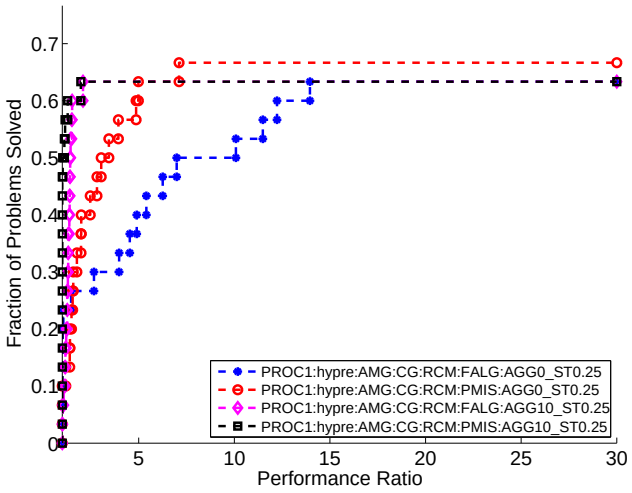
Fig. 9. Memory and time performance profile variations for various ordering schemes with and without diagonal perturbation in the case of WSMP ICT with drop tolerance 0.003 and fill factor 4.9.

Table VIII. Iterative solver configurations that resulted in the best memory, time, and MTP performance profile area for the AMG preconditioners in Hypre and Trilinos. The numbers enclosed by parenthesis denote the number of processors.

Preconditioner	Memory Winner	Time Winner	MTP Winner
Hypre BoomerAMG	CG, RCM, HMIS AGG10, ST0.9 (32) CG, RCM, PMIS AGG0, ST0.9 (64) CG, RCM, PMIS AGG10, ST0.9 (1 2 4 8) CG, NONE, PMIS AGG10, ST0.9 (16)	CG, RCM, HMIS AGG10, ST0.9 (32) CG, RCM, PMIS AGG0, ST0.7 (4 64) CG, RCM, PMIS AGG10, ST0.9 (1) CG, ND, FALG AGG0, ST0.9 (8) CG, NONE, FALG AGG0, ST0.9 (2) CG, NONE, PMIS AGG0, ST0.7 (16)	CG, RCM, HMIS AGG10, ST0.9 (32) CG, RCM, PMIS AGG10, ST0.9 (1 2 4 8) CG, NONE, PMIS AGG10, ST0.7 (64) CG, NONE, PMIS AGG10, ST0.9 (16)
Trilinos ML	CG, RCM, ML-SA SGS, SS2, UCMIS (8) CG, RCM, ML-SA SGS, SS3, UCMIS (4) CG, NONE, ML-SA SGS, SS2, UCMIS (1 2) CG, NONE, ML-SA SGS, SS2, MIS (32) CG, NONE, ML-SA SGS, SS3, UCMIS (16) CG, NONE, ML-SA SGS, SS3, MIS (64)	CG, RCM, ML-SA SGS, SS2, UC (2 8) CG, RCM, ML-SA SGS, SS2, PMETIS (32) CG, RCM, ML-SA SGS, SS3, UC (4) CG, RCM, ML-SA SGS, SS3, PMETIS (1) CG, NONE, ML-SA SGS, SS2, UC (16) CG, NONE, ML-SA SGS, SS3, UCMIS (64)	CG, RCM, ML-SA SGS, SS2, UCMIS (2) CG, RCM, ML-SA SGS, SS2, UC (8) CG, RCM, ML-SA SGS, SS2, PMETIS (32) CG, RCM, ML-SA SGS, SS3, UC (4) CG, RCM, ML-SA SGS, SS3, PMETIS (1) CG, NONE, ML-SA SGS, SS2, UC (16) CG, NONE, ML-SA, SGS SGS, SS3, UCMIS (64)



(a) Memory performance profile



(b) Time performance profile

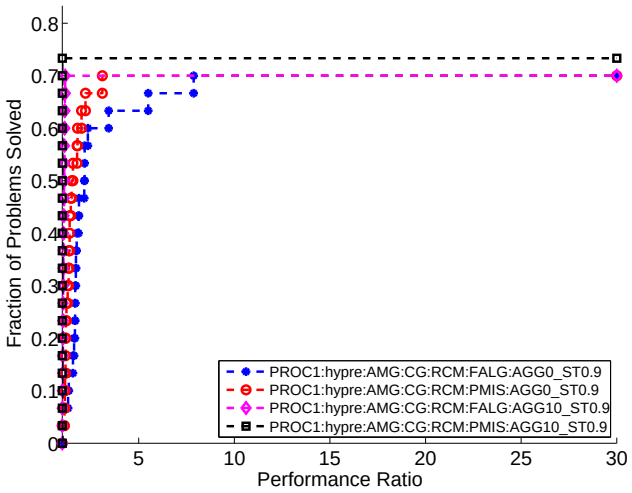
Fig. 10. Memory and time performance profile curves for Hypre BoomerAMG solver configurations for a strong threshold value of 0.25 in the serial case. The legends also provide details on the solver (CG), ordering (RCM), number of levels of aggressive coarsening (AGG), and coarsening schemes (FALG, PMIS).

ordering, which resulted in the best MTP performance in the serial case. We found the performance of the HMIS coarsening scheme to be quite similar to that of the PMIS scheme, so we have included only PMIS and Falgout schemes in these figures. Figure 10 shows the results for a relatively small value of 0.25 for the strong threshold and Figure 11 for a high value of 0.9. Figure 10 shows that Falgout coarsening scheme results in heavy memory and time usage for most problems when used without aggressive coarsening. PMIS appears to be the better coarsening scheme for our test suite with low strong threshold values. This observation is different from what is suggested in the user manual [1], which recommends Falgout coarsening scheme as the default. The memory and time profiles in Figure 11 indicate that the performance difference between the various coarsening schemes is not as significant for a high strong threshold value, especially when aggressive coarsening is used. Note that the authors recommend a high value of strong threshold for 3D problems, which constitute about 50% of our test suite.

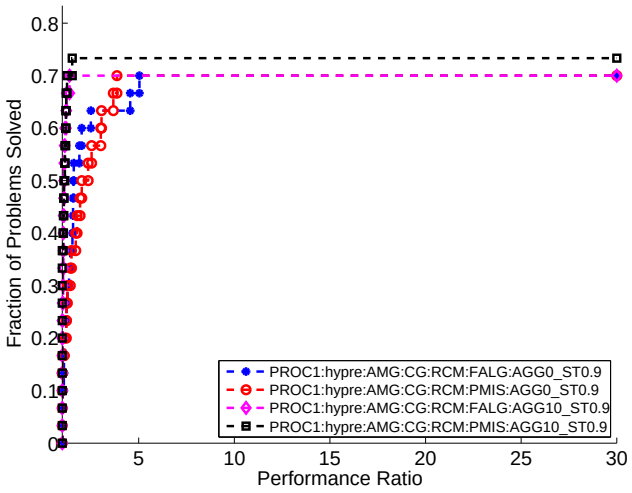
Trilinos ML: For the ML preconditioner in Trilinos, we compared the performance of classical smoothed aggregation (SA), two level SA based domain decomposition (DD), and three level algebraic domain decomposition (DD-ML) with their predefined default set of parameters as described in [30]. In addition, we also experimented with multiple coarsening schemes, smoothers, and the number of smoother sweeps for the SA preconditioner.

Figure 12 shows the time profiles for varying the number of smoother sweeps for the symmetric Gauss-Seidel and Chebyshev smoother. We observe that increasing the number of sweeps from two to three significantly improves the robustness of Chebyshev polynomial smoother. For the symmetric Gauss-Seidel smoother, increasing the smoother sweeps causes only a slight change in the number of problems solved. We plot only the time profiles since the memory usage is not affected by the number of smoother sweeps. Overall, the symmetric Gauss-Seidel smoother is faster and solves more problems.

Figure 13 shows the effect of various coarsening schemes on the performance of



(a) Memory performance profile



(b) Time performance profile

Fig. 11. Memory and time performance profile curves for Hypre BoomerAMG solver configurations for a strong threshold value of 0.9 in the serial case. The legends provide details on the solver (CG), ordering (RCM), number of levels of aggressive coarsening (AGG), and coarsening scheme (FALG, PMIS).

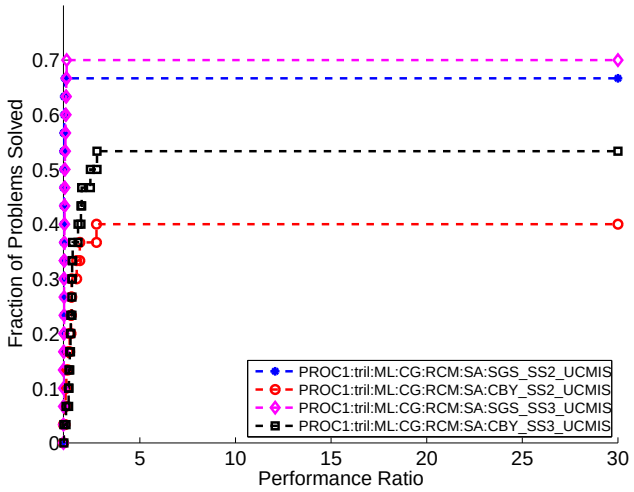


Fig. 12. Time performance profile curves for Trilinos ML solver configurations for different smoothing sweeps of symmetric Gauss-Seidel and Chebyshev smoothers in the serial case. The legends provide details on the solver (CG), ordering (RCM), and coarsening scheme (UCMIS).

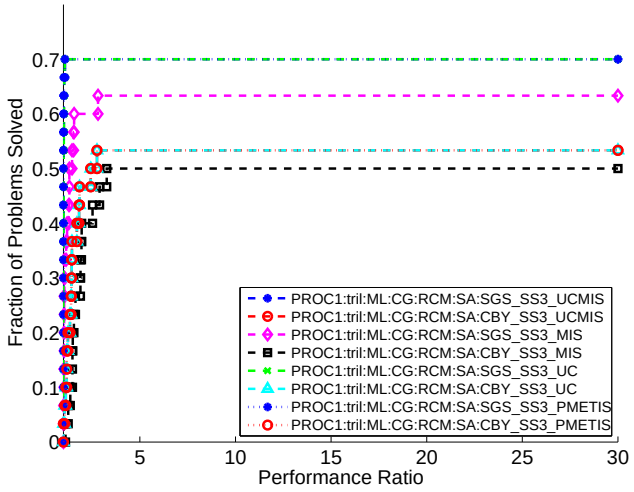


Fig. 13. Time performance profile curves for Trilinos ML configurations using the Chebyshev and symmetric Gauss-Seidel smoothers with various coarsening schemes in the single processor case. The legends provide details on the solver (CG), ordering (RCM), and smoother sweeps (SS3).

Chebyshev and symmetric Gauss-Seidel smoothers. Once again, we do not show the memory profiles since they are very similar for all the coarsening schemes. The time profiles indicate that the performance of all coarsening schemes except MIS is nearly identical in Trilinos ML with classical smoothed aggregation is nearly identical.

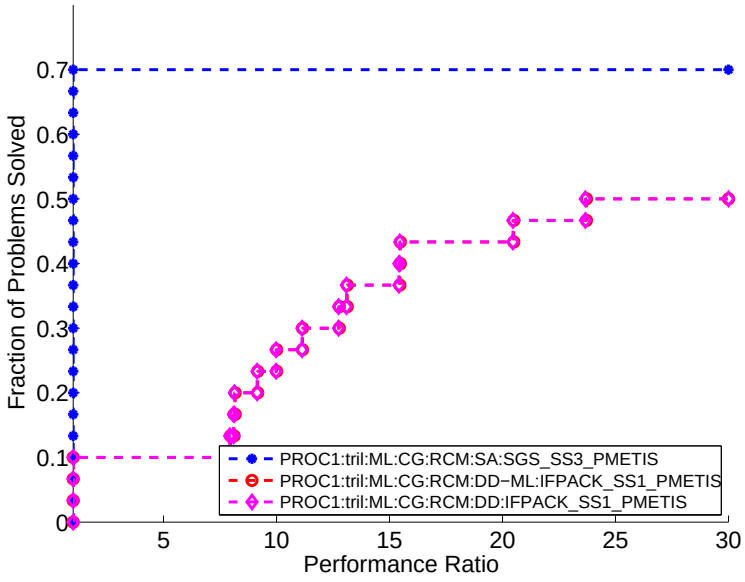
Figure 14 shows a comparison of the time and memory usage of the DD, DD-ML configurations, and the overall best SA configuration while using the ParMETIS coarsening scheme. We observe that the smoothed aggregation approach is very efficient with respect to memory; however, the difference in time performance is not so dramatic.

Comparison of Hypre BoomerAMG and Trilinos ML: Figure 15 shows a comparison of the overall best configurations of Trilinos ML and Hypre BoomerAMG in the single processor scenario. We observe that the memory profiles of Trilinos ML and Hypre BoomerAMG are quite close, although BoomerAMG solves more problems. The time profiles indicate that Hypre BoomerAMG solves a large fraction of problems using much less time than Trilinos ML.

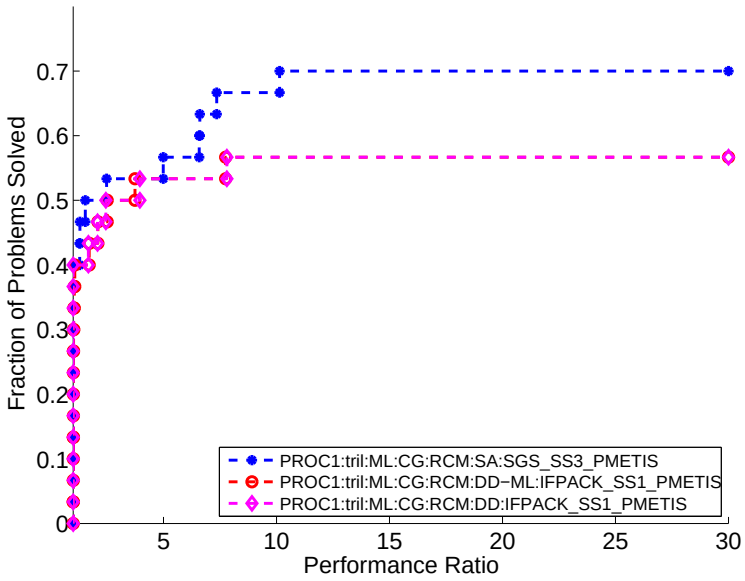
d. Sparse Approximate Inverse

Table IX. Iterative solver configurations that resulted in the best memory, time, and MTP performance profile area for the ParaSails preconditioner in Hypre. The numbers enclosed by parenthesis denote the processor number corresponding to the overall best solver configurations. The configuration names provide details on parameters such as number of levels (Lev), threshold (Th), and filter (Flt).

Preconditioner	Memory Winner	Time Winner	MTP Winner
Hypre ParaSails	CG, ND, PLev1 Th0, Flt0.05 (2) CG, ND, PLev2 Th0.01, Flt0.001 (32 64) CG, ND, PLev2 Th0.1, Flt0.001 (1 4 8 16)	CG, ND, PLev1 Th0.1, Flt0 (32) CG, ND, PLev2 Th0.1, Flt0.001 (1 2 4 8 16) CG, NONE, PLev2 Th0.1, Flt0 (64)	CG, ND, PLev1 Th0.1, Flt0 (32) CG, ND, PLev1 Th0.1, Flt0.001 (1 2 4 8 16 64)

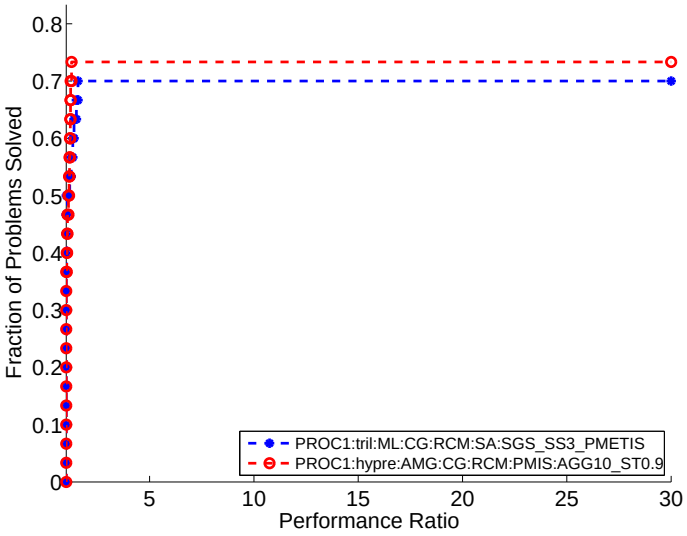


(a) Memory performance profile

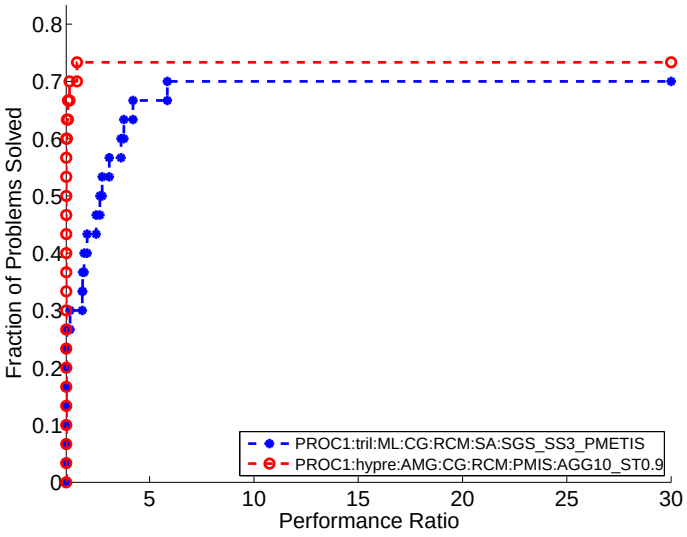


(b) Time performance profile

Fig. 14. Memory and time performance profile curves for the best Trilinos ML SA, DD, and DD-ML configurations for RCM ordering in the single processor case.

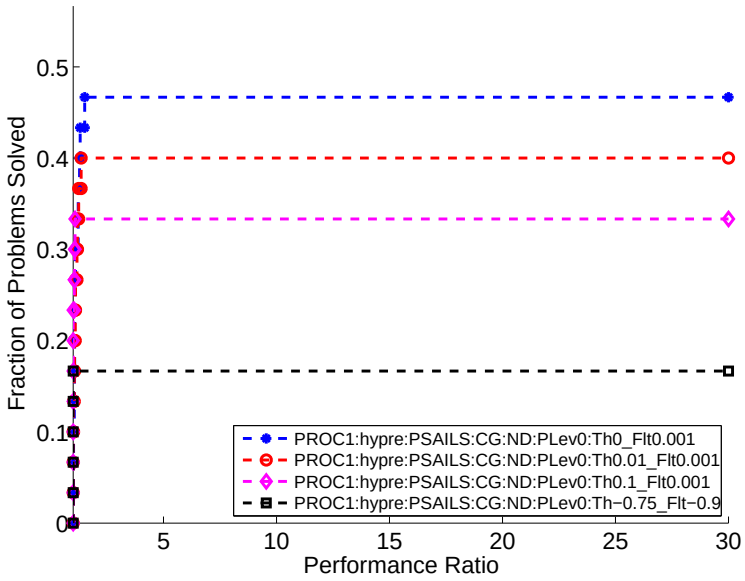


(a) Memory performance profile

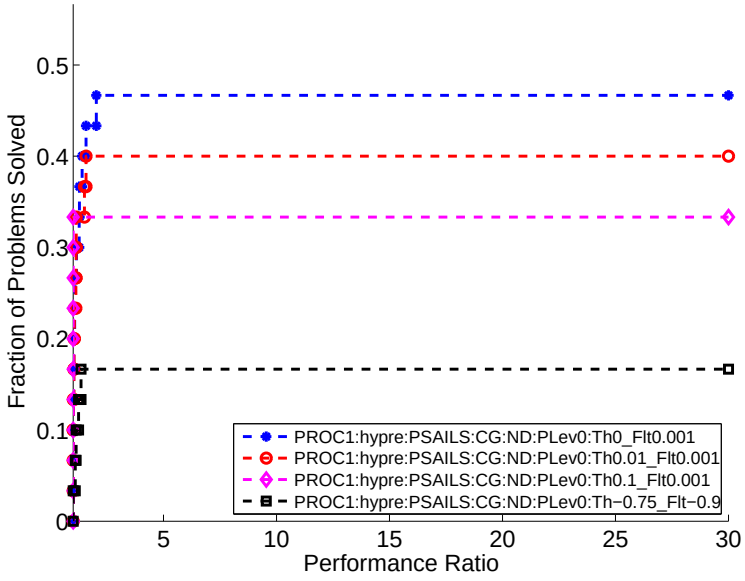


(b) Time performance profile

Fig. 15. Memory and time performance profile curves for the overall best Trilinos ML and Hypre BoomerAMG solver configurations. The legends provide details on the solver (CG), ordering (RCM), coarsening scheme (PMIS, PMETIS), number of levels of aggressive coarsening (10), and strong threshold values (0.9), ML preconditioner type (SA), and the number of smoother sweeps (3).

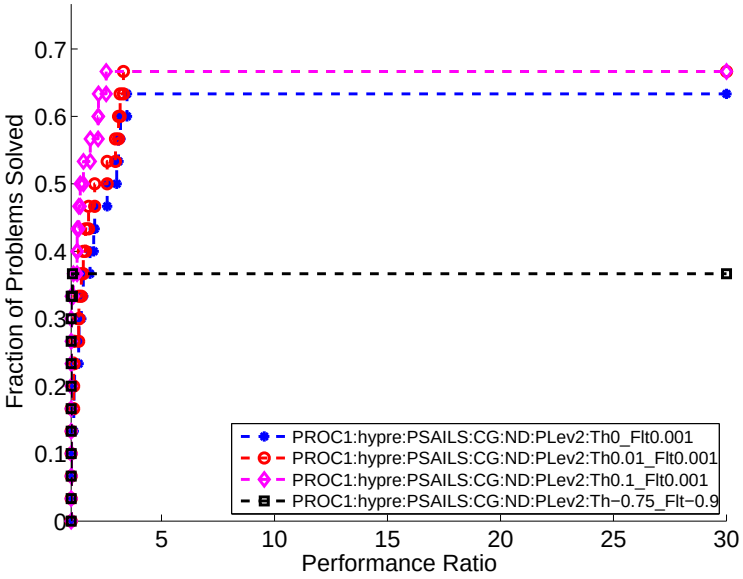


(a) Memory performance profile

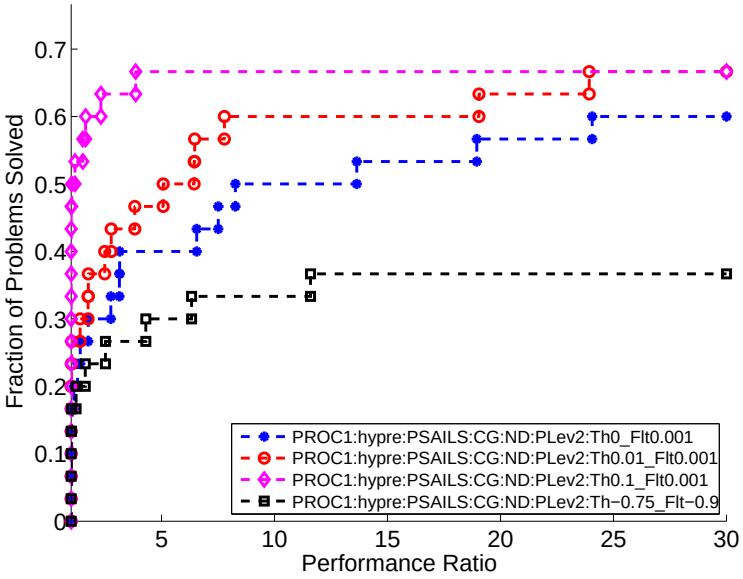


(b) Time performance profile

Fig. 16. Memory and time performance profile curves for Hypre ParaSails solver configurations corresponding to various threshold and filter values for a fixed number of levels (PLev0) and best MTP ordering (ND).



(a) Memory performance profile



(b) Time performance profile

Fig. 17. Memory and time performance profile curves for Hypre ParaSails solver configurations corresponding to various threshold and filter values for a fixed number of levels (PLev2) and best MTP ordering (ND).

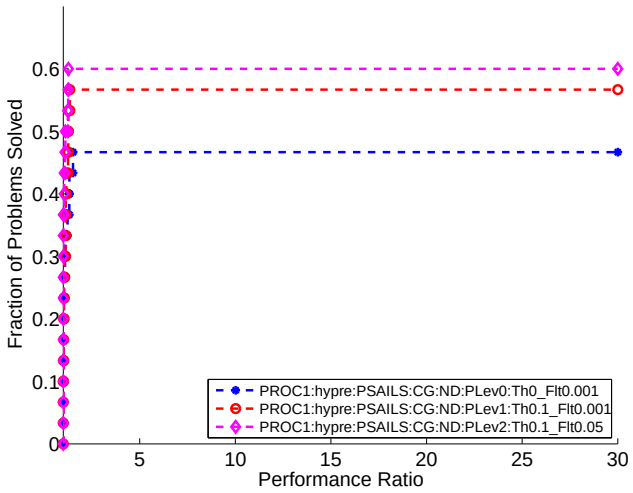
For the ParaSails preconditioner in Hypre, we experimented with multiple threshold values (0, 0.01, 0.1, -0.75, -0.9) and filter values (0, 0.001, 0.05, -0.9) for three different levels as suggested by the user manual [1]. Table IX summarizes the configurations that resulted in the best performance profiles in our experiments.

Figures 16 and 17 show the time and memory profiles for different threshold and filter values with nested dissection ordering for 0 and 2 levels, respectively. When the number of levels is 0, the memory and time requirements of all the configurations are similar. However, their robustness varies a lot and the configuration with threshold 0.0 and filter 0.001 solves the most problems. In the case of two levels, the memory and time profile curves for different configurations are well separated and the threshold value of 0.1 appears to work the best. The negative threshold values suggested by the authors in the user manual, which have a different interpretation from non negative values, solved the fewest problems.

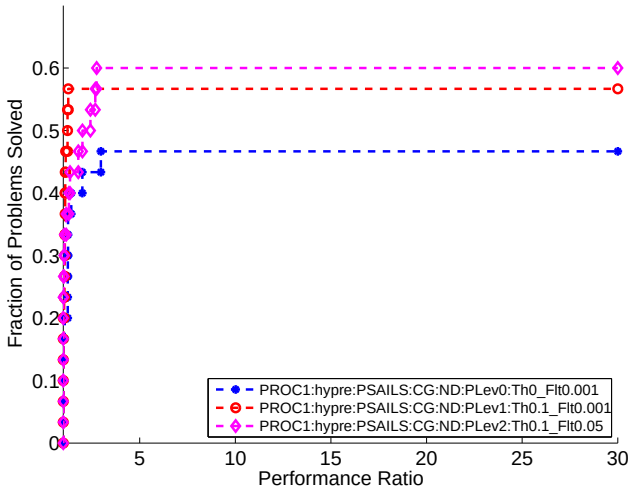
Figure 18 shows the performance profile curves for the best threshold and filter combination for 0, 1, and 2 levels. We observe that the best configuration for 2 levels solves the maximum number of problems, but is considerably slower than the best configuration for 1 level. An interesting observation is that the memory consumption actually declines with increasing number of levels. This is because the best configurations of higher levels include higher values of threshold and filter to drop more entries.

e. Variation of Overall Best Configurations with Number of Processors

From the tables in the previous section, we note that the best overall configurations of most preconditioner implementations are different for different number of processes. To determine if these processor-specific overall best configurations are substantially different in their performance, we plotted the MTP performance profile curves for each of these configurations for a fixed number of processors. Between 1 and 64 processors, we observed very small differences between the performance for the 7 sets of best configurations (for 1,



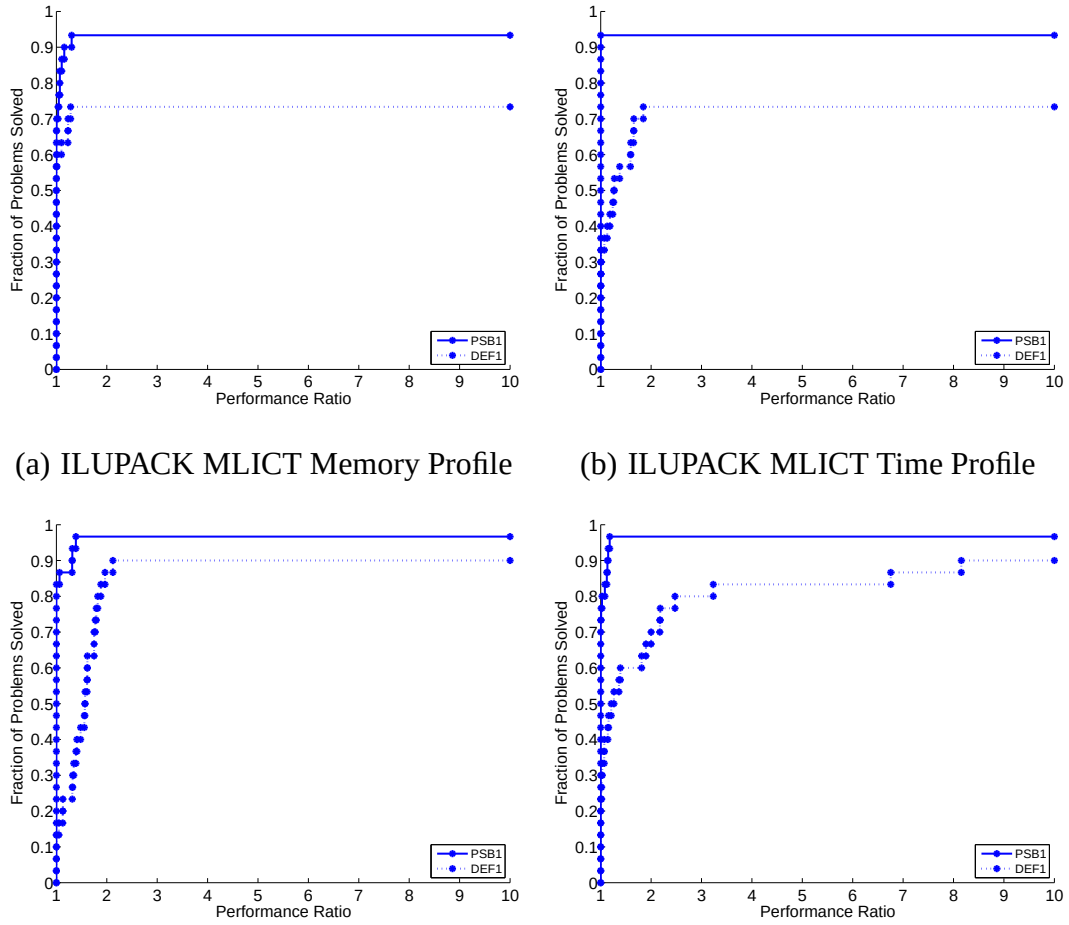
(a) Memory performance profile



(b) Time performance profile

Fig. 18. Memory and time performance profile curves for Hypre ParaSails solver configurations that resulted in the best MTP profile area for a fixed ordering (ND) and level combination. The legends provide details on the solver (CG), number of levels (Lev), threshold (Th), and filter (Flt).

2, 4, 8, 16, 32, and 64 processors). The differences are usually tied to different scalabilities of the preconditioner generation and the solution phases, which we discuss in detail in Section D.5. If there is a considerable difference in the scalability of the two phases, then parameters that shift more computation to the more scalable phase would be favored as the number of processors is increased.



(a) ILUPACK MLICT Memory Profile (b) ILUPACK MLICT Time Profile

(c) WSMP ICT Memory Profile (d) WSMP ICT Time Profile

Fig. 19. Memory and time performance profile curves for the problem independent best (PIB) and the problem-specific best (PSB) configurations of ILUPACK MLICT and WSMP ICT.

2. Performance Benefits of Fine Tuning

The users fine-tune the parameters of preconditioned iterative solvers to optimize the performance for a particular application, instead of using the default values. In this section, we discuss the effect of problem-specific fine-tuning of parameters on the performance of various preconditioner implementations. For each solver configuration group and processor configuration, we compare two sets of performance values. The first set of performance values corresponds to the overall best configuration based on the MTP metric. The second set corresponds to problem-specific best (PSB) performance values; i.e., the performance of the configurations with the least MTP value within the configuration group for each problem. These two sets of performance values are compared for 1, 2, 4, 8, 16, 32, and 64 processors for the solvers whose parallel implementations are available.

Figure 19 shows the memory and time performance profile curves for the overall best and problem-specific best configurations for WSMP ICT and ILUPACK MLICT. These preconditioners currently have only a serial implementation available, so only single processor results are shown. Figure 19 shows that both solvers show considerable improvement in performance due to problem-specific best parameter selection. The improvement is more significant for ILUPACK MLICT than for WSMP ICT.

Figures 20–25 show the performance variation between the PSB and overall best for all the preconditioners with parallel implementations for which such a comparison . Instead of showing separate performance profile curves for memory and time separately for each processor setting, we combine the information contained in separate memory and time plots in a single figure. Each figure has two circles for each processor configuration. The empty circles correspond to the overall best parameter configurations and the filled circles correspond to the problem-specific best performance. The x- and y-coordinates of each circle are respectively the areas of time and memory profiles obtained by considering the

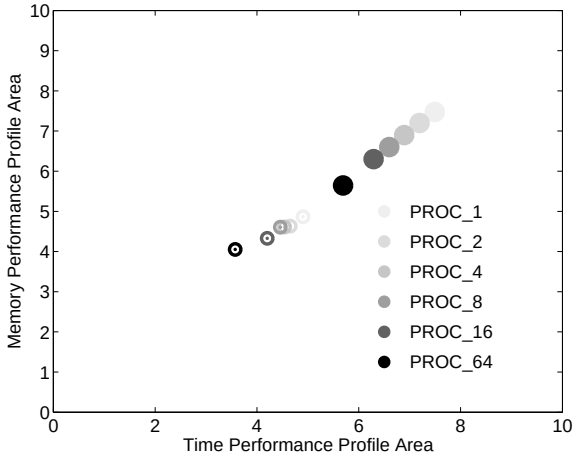


Fig. 20. Memory and time profile areas for the overall best (empty circles) and the problem-specific best configurations (filled circles) of Hypre IC(k) preconditioner for multiple processors.

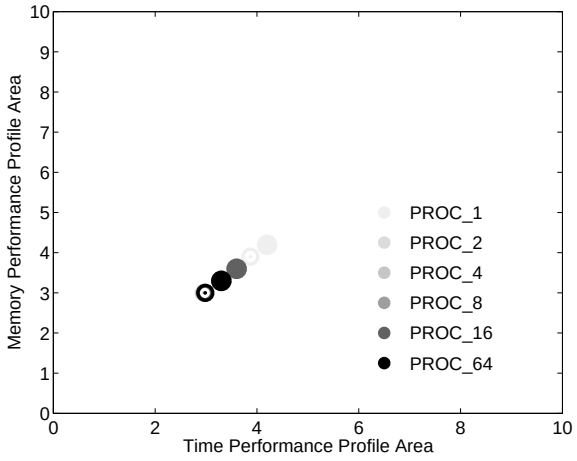


Fig. 21. Memory and time profile areas for the overall best (empty circles) and the problem-specific best configurations (filled circles) of PETSc IC(k) preconditioner for multiple processors.

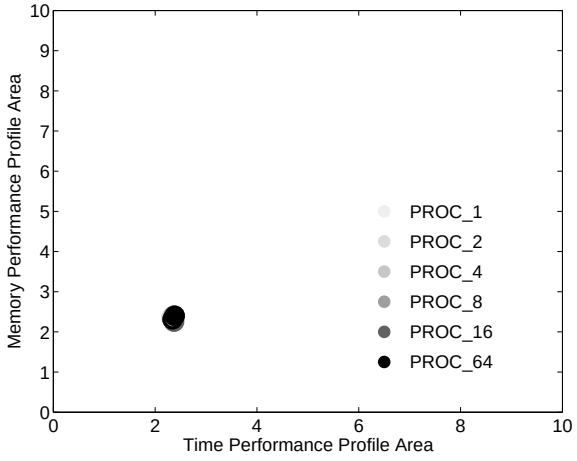


Fig. 22. Memory and time profile areas for the overall best (empty circles) and the problem-specific best configurations (filled circles) of Trilinos IC(k) preconditioner for multiple processors.

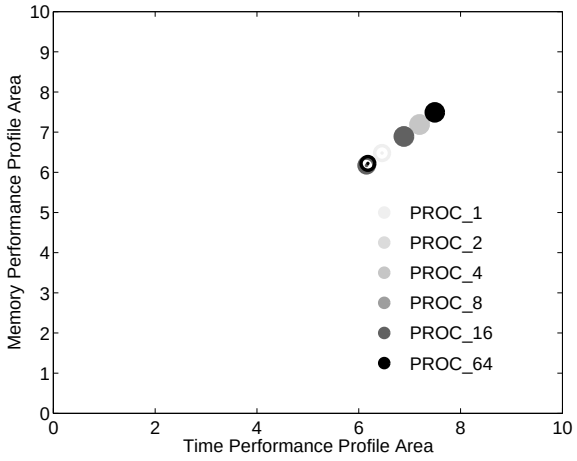


Fig. 23. Memory and time profile areas for the overall best (empty circles) and the problem-specific best configurations (filled circles) of Hypre BoomerAMG preconditioner for multiple processors.

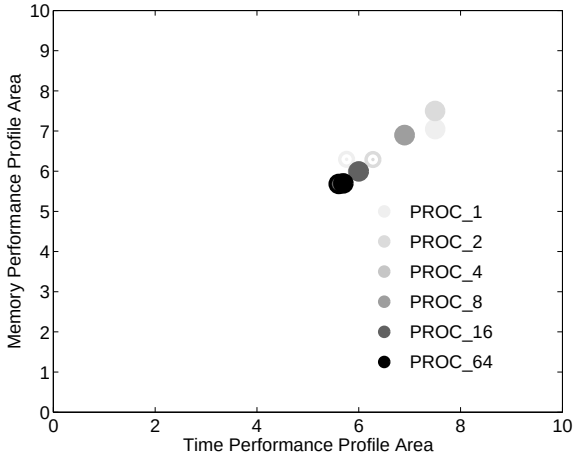


Fig. 24. Memory and time profile areas for the overall best (empty circles) and the problem-specific best configurations (filled circles) of Trilinos ML preconditioner for multiple processors.

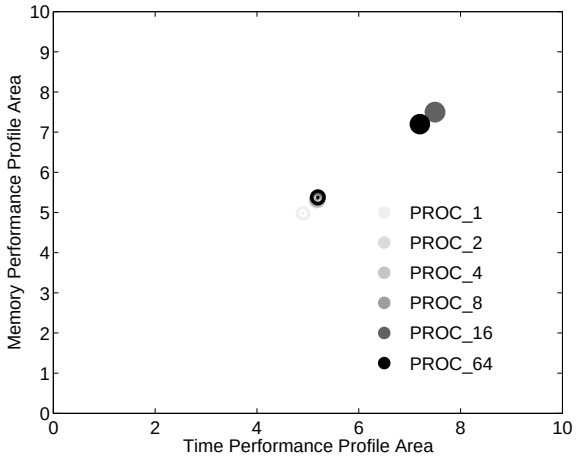


Fig. 25. Memory and time profile areas for the overall best (empty circles) and the problem-specific best configurations (filled circles) of Hypre ParaSails preconditioner for multiple processors.

PIB and PSB performance of a single configuration group together. The size of each circle is proportional to the number of problems solved. In our experiments, the PSB and overall best profile areas for PETSc BlockSolve95 and Trilinos IC(k) were nearly identical and are, therefore, not shown. However, in the case of Hypre's IC(k), ParaSails, and Trilinos ML in Figures 20 – 24, we observe that there is considerable performance benefit to fine-tuning for both memory and time as indicated by the separation of the PIB and PSB circles. The gap between PSB and PIB profile areas is seen to reduce as we increase the number of processors suggesting that fine tuning is less important for higher number of processors. In the case of PETSc IC(k) and Hypre BoomerAMG in Figures 21 and 23, the gap between the PIB and PSB profiles is minor for both memory and time.

3. Influence of Parameters on Solver Performance

In this section, we analyze the relative importance of the various preconditioner parameter choices on the time and memory performance within each solver configuration group. In Section D.2, we observed the impact on performance due to collective fine-tuning of the parameters for each solver configuration group. However, it does not provide any information on the individual effects of the various parameters that were experimented with. In order to capture the relative importance of parameters, there are two main approaches described in global sensitivity analysis literature [75] based on linear regression parameters and conditional variance respectively. We also propose a new fine-tuning score based on variance conditioned on complementary factors.

a. Regression-based Sensitivity

The first approach is to perform a linear regression [75] with the performance values as the target variable and the various parameters as the dependent variables with the categorical parameters being converted to multiple binary variables. For the current scenario, one could

choose the performance ratios (normalized performance with respect to the overall best configuration $PIB(\mathcal{C})$) as the target variable in order to adjust for effects of the individual matrices. The coefficients of the regression model indicate the degree and direction of change in the performance ratio for each unit change in a parameter and can be interpreted as a measure of sensitivity of the performance with respect to the influencing parameters. However, this approach is often not suitable in case of non-linear behavior or large number of outliers as is often the case. Furthermore, this approach does not adequately address the fact that the parameters tend to have different distributions. These drawbacks limit the applicability of this regression based approach for our domain.

b. Conditional Variance-based Sensitivity

Another common approach in sensitivity analysis is to capture the relative importance of a parameter in terms of the reduction in variance of the performance metric conditioned on the parameter value [76]. Let F_i be the i^{th} parameter and μ be the performance metric of interest normalized for each matrix to reduce variance. Let $V(\mu)$ be the global variance of the performance metric, and let $V_{F_{-i}}(\mu|F_i = f_i)$ denote the conditional variance of μ when the parameter F_i takes the value f_i and the variation is over F_{-i}(i.e., all factors except F_i). The key idea is that freezing one potential source of variation results in a conditional variance $V_{F_{-i}}(\mu|F_i = f_i)$ that is lower than the unconditional global variance $V(\mu)$ and is determined only by parameters other than F_i. Since we desire a sensitivity measure independent of the parameter values f_i, we consider the expectation of the conditional variance over all possible values of the factor F_i, i.e., $E_{F_i}(V_{F_{-i}}(\mu|F_i))$. This sensitivity measure is always lower or equal to the global variance $V(\mu)$ and it can be shown that

$$E_{F_i}(V_{F_{-i}}(\mu|F_i)) + V_{F_i}(E_{F_{-i}}(\mu|F_i)) = V(\mu),$$

where the second term denotes the variance in the expected performance metric conditioned on F_i, which is always non-negative. A small value of $E_{F_i}(V_{F_{-i}}(\mu|F_i))$ or, in other words, a large value of $V_{F_i}(E_{F_{-i}}(\mu|F_i))$ implies that most of the variance in μ can be explained by the parameter F_i indicating that it is an important factor. The conditional variance $V_{F_i}(E_{F_{-i}}(\mu|F_i))$ is typically normalized by the global variance $V(\mu)$ to give the importance measure

$$S_i = \frac{V_{F_i}(E_{F_{-i}}(\mu|F_i))}{V(\mu)}.$$

Since the behavior tends to vary across matrices, we also aggregate both the conditional and global variance across the matrices.

Figure 26 shows the relative importance of the various factors for different preconditioners in the serial case. The height of the bars corresponding to each parameter indicates the reduction in variance attained with respect to time and memory due to fixing the parameter to a particular value. A high value of reduction indicates that there is very little variance that is not explained by this parameter. For example, in Figure 26, we observe that the level of fill parameter is the most important parameter whereas ordering is the least important one. For Hypre ParaSails, the memory usage is much more sensitive to the filter parameter than the threshold, whereas the opposite behavior is observed in the case of time. In the case of Hypre BoomerAMG, number of aggressive coarsening levels impact memory the most, whereas the strong threshold is the most important factor with respect to time. For Trilinos ML, both smoother as well as the smoother sweeps seems to have the maximum influence. This is just an artifact of the correlation between the parameters used in this study. We analyze this case in more detail later in section d. For both ILUPACK MLICT and WSMP ICT, drop tolerance is the most important parameter with respect to both time and memory.

Figure 27 shows the relative importance of the various parameters in a preconditioner

with respect to memory and time as the number of processors is increased. The relative importance of the parameters remain the same irrespective of the number of processors. In the case of PETSc BlockSolve95, there is only a single parameter and hence, we omit it.

c. Variance-based Fine-tuning Score

The variance-based sensitivity score discussed in Section b is used commonly in statistical literature for retrospective analysis since a high value of sensitivity with respect to a parameter indicates that the response variable can be confined to a small interval (i.e., has low variance), which correlates to the explanatory and predictive power of the parameter. From the variance decomposition relation in Section b, we note that the conditional variance $V_{F_i}(E_{F_{-i}}(\mu|F_i))$ indicates the spread between the average performance values obtained for different fixed values of a parameter. However, this measure does not capture the effect of modifying a single parameter *while keeping the rest fixed* as is common in a practical fine-tuning scenario. Hence, we consider an alternate fine-tuning sensitivity measure for a parameter F_i defined as the expectation of variance of μ for different values of F_i for fixed values on the rest of the parameters, i.e., $E_{F_{-i}}(V_{F_i}(\mu|F_{-i}))$. This variance is a better indication of the change in the performance one can expect by fine-tuning a single parameter keeping others fixed and we define this value normalized by the global variance as the *fine-tuning sensitivity*. Note that the two measures $V_{F_i}(E_{F_{-i}}(\mu|F_i))$ and $E_{F_{-i}}(V_{F_i}(\mu|F_{-i}))$ are closely related to each other and involve applying the aggregation and variance is different orders. When the parameters are all uncorrelated and μ exhibits linear dependence on the parameters, the two measures are identical.

Figure 28 shows the average normalized variation with respect to both time and memory for each of the fine-tuneable parameters for different preconditioners in the serial case. The height of the bars corresponding to each parameter corresponds to the normalized variance of the change in performance one can expect by fine-tuning that parameter keeping

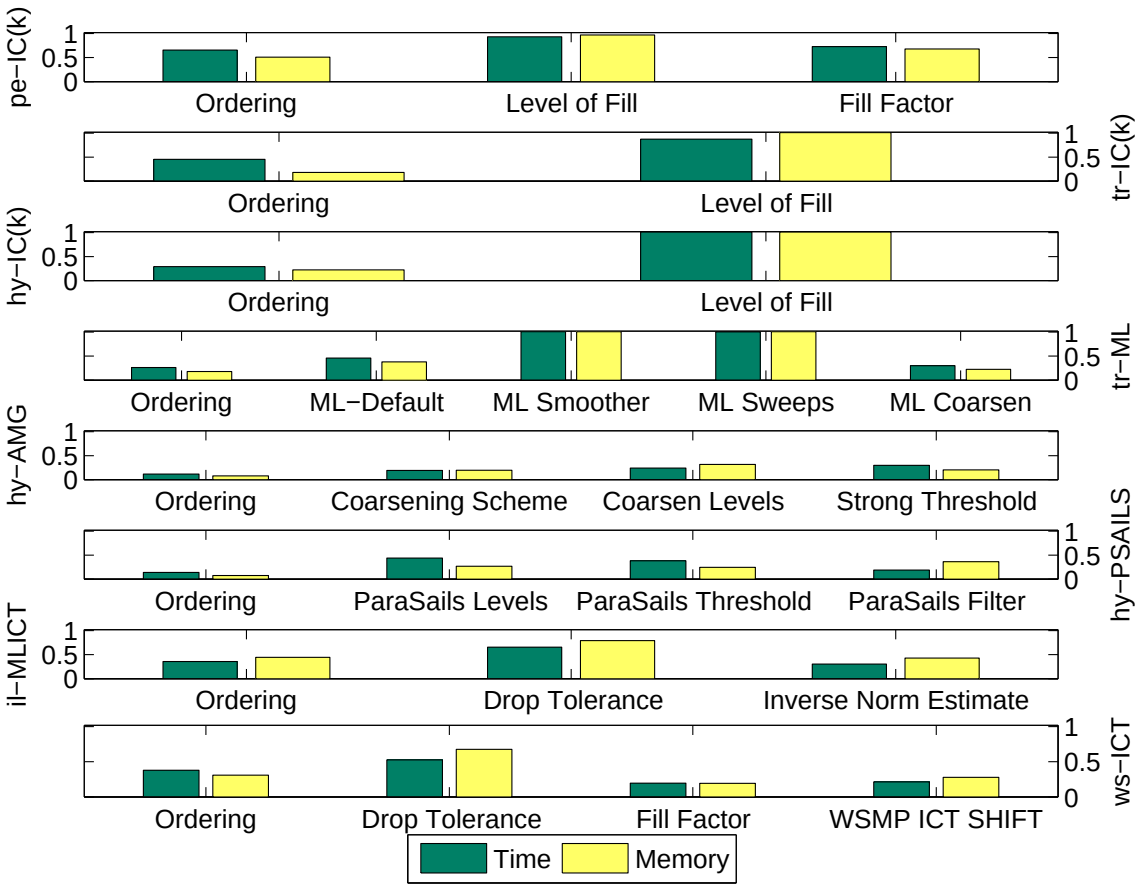
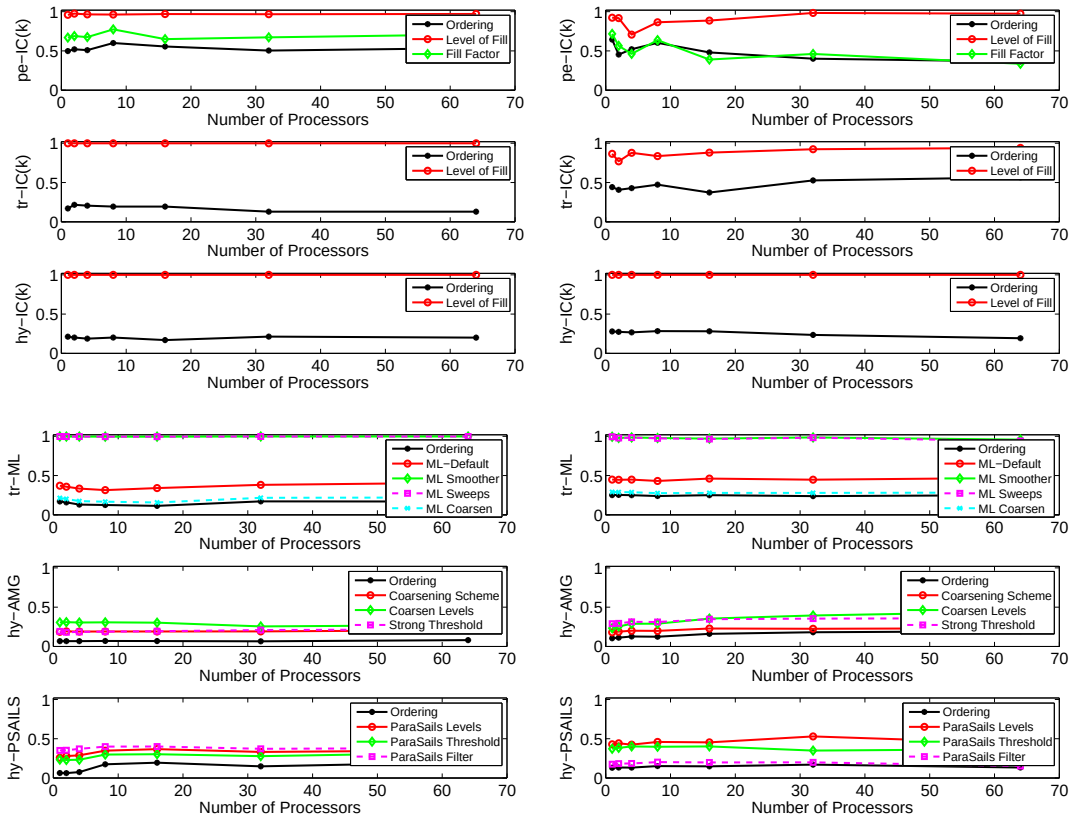


Fig. 26. Conditional variance based sensitivity scores of the parameters in a preconditioner with respect to time and memory in the serial case.



Relative importance of parameters with respect to memory.

Relative importance of parameters with respect to time.

Fig. 27. Relative importance with respect to memory and time of the various parameters for the different preconditioners in the parallel case. Each curve in the subplots corresponds to a parameter that is varied in our study.

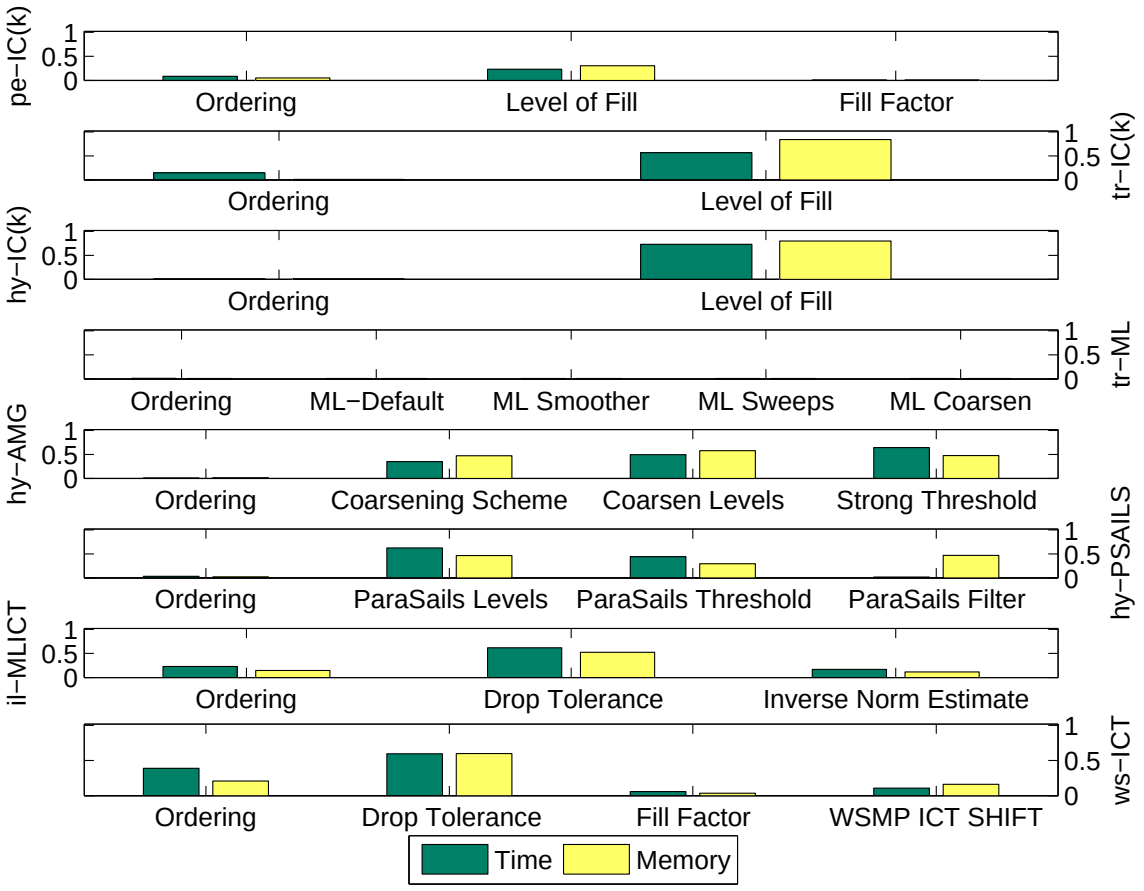
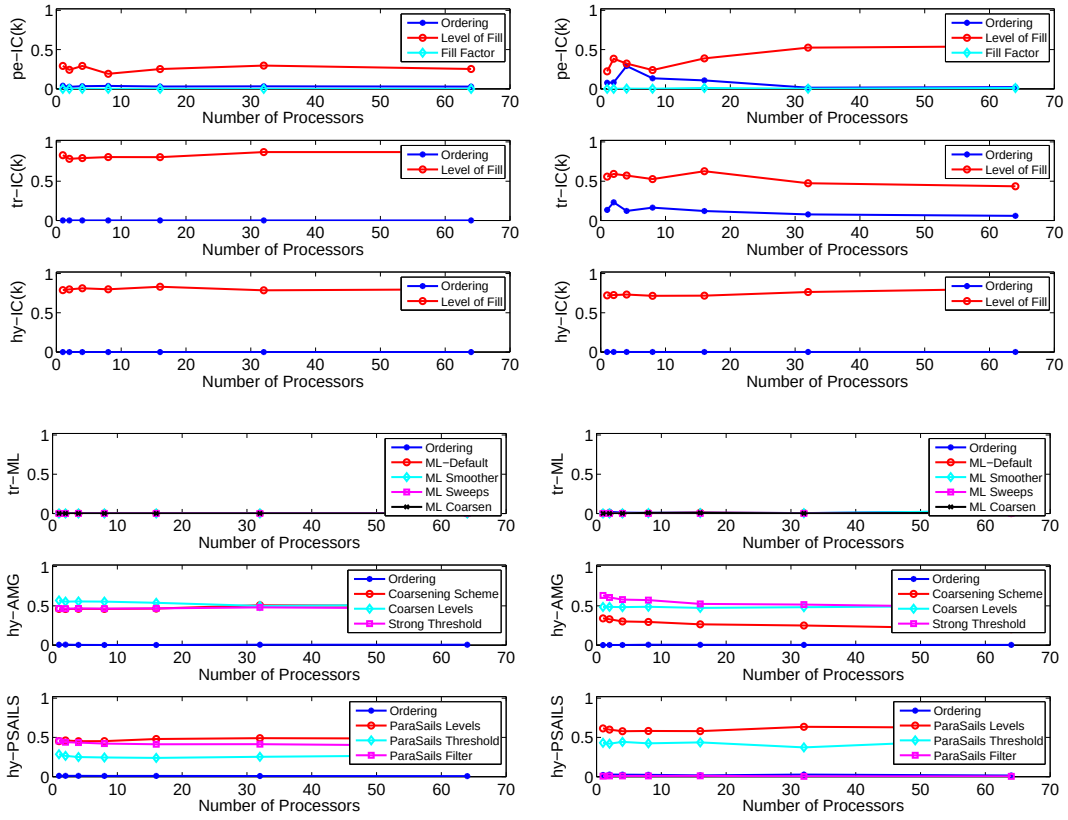


Fig. 28. Average normalized variation with respect to memory and time for each of the fine-tuneable parameters of the various preconditioners in the serial case.



Variation in memory Variation in time

Fig. 29. Average normalized variation with respect to memory and time for the fine-tuneable parameters of the various preconditioners in the parallel case. Each curve in the subplots corresponds to a parameter that is varied in our study.

others fixed. The relative heights of the bars is important in this case since it gives an indication of the variance on performance due to each parameter. This is more evident when you compare the plots for PETSc IC(k) in Figures 26 and 28. For example, in Figure 26, we observe that the ordering and fill factor parameters have comparable conditional variance-based scores in comparison to the level of fill parameter. However, in Figure 28, it is clear that the variation in performance due to level of fill is much higher in comparison to that due to ordering and fill factor. Although the general trends for other preconditioners are similar to those based on the conditional variance-based sensitivity plot, there are some subtle differences. For example, the variation in performance due to ordering is almost negligible for both Hypre BoomerAMG and ParaSails. Similarly, the variation due to ParaSails threshold parameter is negligible with respect to time.

Figure 29 shows the effect of fine-tuning the various parameters on memory and time as the number of processors is increased. The level of fill has the larger influence on memory and time for all the three IC(k) implementations in PETSc, Trilinos, and Hypre. Only in the case of PETSc and Trilinos IC(k), did the ordering scheme cause any significant variation with respect to both memory and time. In the case of PETSc BlockSolve95, the variations with respect to time and memory were minor and are, therefore, omitted. For Hypre BoomerAMG, the ordering scheme has the least effect on both time and memory, while the effect of the other parameters (coarsening scheme/levels and strong threshold) is substantial, but nearly flat as the number of processors varies. In the case of Hypre ParaSails, the number of levels has the maximum impact on both time and memory. Although the filter parameter has a significant impact on memory, with respect to time, the variation is negligible. The trends for both conditional variance based sensitivity and variance based parameter fine-tuning plots show similar behavior except in case of Trilinos ML due to correlation between parameters, which we discuss below.

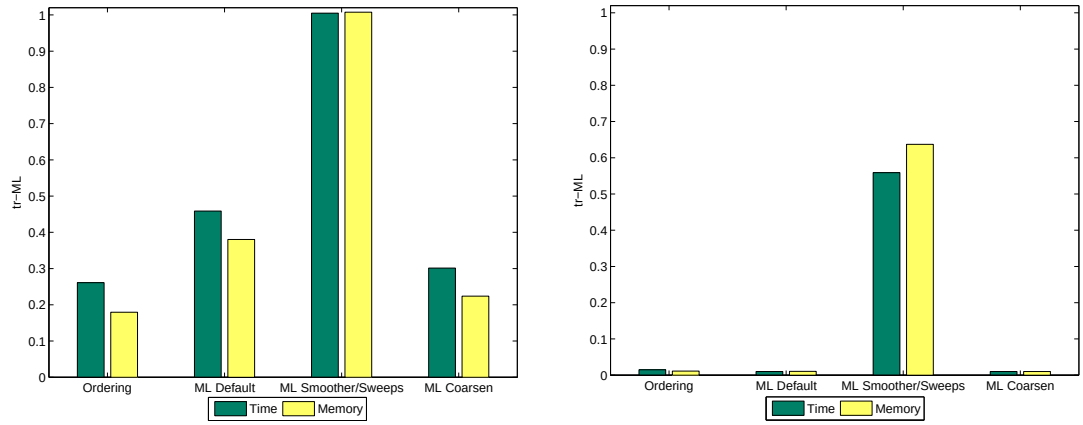
d. Correlated Parameters

The fine-tuning measure discussed above is suitable for cases where each choice of a parameter can occur with all other choices of the rest of parameters to form a valid configuration, i.e., the parameters are statistically independent and correlated. In such a case, using the fine-tuning measure, one can order the parameters and fine-tune them one at a time. However, in certain cases, there tend to exist groups of highly (positively or negatively) correlated parameters, e.g., smoother and number of smoother sweeps (correlation coefficient $<$ -0.7)in case of Trilinos ML. When there is a strong dependence among the parameters, then freezing all the other parameters has an implicit effect of limiting the range of possible values for the parameter in question resulting in a misleading value for the fine-tuning measure. For such scenarios, it is more appropriate to form groups of highly correlated parameters and simultaneously fine-tune them based on their joint fine-tuning score. Figures 30 and 31 show the fine-tuning measures for Trilinos-ML with grouping of the parameters smoother and number of sweeps in the serial and parallel case respectively. We observe that the combined parameter ML Smoother/Sweeps is the most important parameter in Figures 30 and 31 and the scales indicate that the associated variances are significant.

In the current data, we observed that across all the solver configuration groups, the parameters (smoother and number of smoother sweeps) are the only ones with absolute correlation $\geq$ 0.5. Hence, the fine-tuning scores depicted in the Figures 30 and 31 are realistic for the rest of the parameters.

4. Relative Performance of Preconditioner Implementations

We now use the MTP metric to compare the performance of all the preconditioner implementations studied in this paper. We compare the various implementation under two scenarios. We first compare the overall best (PIB) or the experimentally determined de-



(a) Relative importance of the parameters in Trilinos ML

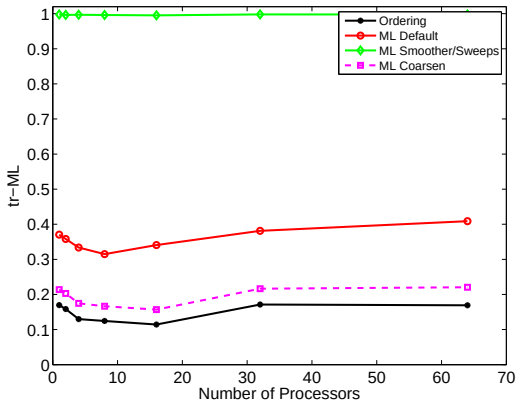
(b) Variance based fine-tuning score of the parameters in Trilinos ML

Fig. 30. Conditional variance based sensitivity scores and variance based fine-tuning scores with respect to memory and time for each of the fine-tuneable parameters of Trilinos ML preconditioner in the serial case.

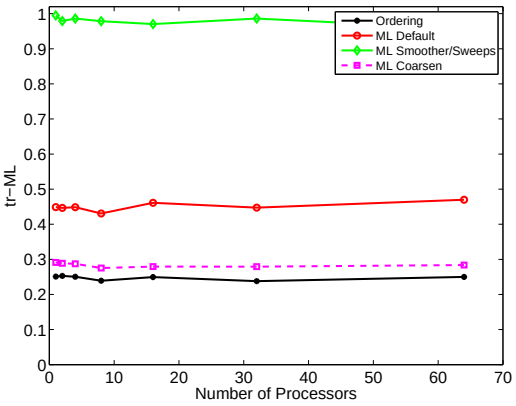
fault configurations of the preconditioners, i.e., we choose the parameter configuration for each preconditioner that has the best overall performance on our test suite. In the second scenario, we compare the preconditioners based on their PSB (problem specific best) configurations, i.e., we pick the best parameter configuration of each preconditioner for each individual matrix. For both the PIB and PSB scenarios, we present the results on a single processor and on 64 processors. We also present the results of simultaneously projecting multiple performance metrics, which helps in analyzing the relative memory, time, and robustness of the preconditioners, both in the serial and parallel case.

a. Problem Independent Best Configurations

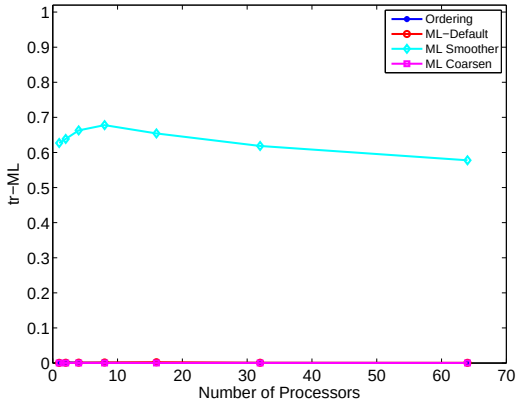
The best MTP parameter combinations for all the preconditioners are shown in Tables X and XI for the 1 and 64 processor cases. These solver configurations are good candidates for default values that have a high probability of yielding a small memory-time product for an arbitrary problem.



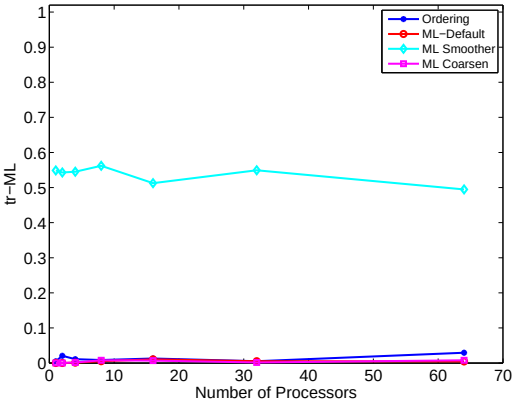
(a) Relative importance of the parameters in Trilinos ML w.r.t. memory



(b) Relative importance of the parameters in Trilinos ML w.r.t. time



(c) Variance based fine-tuning score of the parameters in Trilinos ML w.r.t. memory



(d) Variance based fine-tuning score of the parameters in Trilinos ML w.r.t. time

Fig. 31. Conditional variance based sensitivity scores and variance based fine-tuning scores with respect to memory and time for each of the fine-tuneable parameters of Trilinos ML preconditioner in the parallel case. Each curve in the subplots corresponds to a parameter that is varied in our study.

Table X. Iterative solver configurations that resulted in the best overall performance with respect to memory-time product profile area in the serial case.

Preconditioner	**Solver**	**Ordering**	**Preconditioner Parameters**
PETSc IC(k)	CG	RCM	Fill factor 1, Level of fill 0
PETSc BlockSolve	CG	RCM	-
Trilinos IC(k)	CG	RCM	Level of fill 8
Trilinos ML	CG	RCM	Smoothed aggregation, Symmetric Gauss-Seidel smoother Smoother sweeps 3, ParMETIS Coarsening
Hypre IC(k)	CG	RCM	Level of fill 1
Hypre BoomerAMG	CG	RCM	PMIS Coarsening, Aggressive coarsening levels 10 Strong threshold 0.9
Hypre ParaSails	CG	ND	Number of levels 1, Threshold 0.1, Filter 0.001
Ilupack MLICT	CG	RCM	Drop-tolerance 0.03, Inverse norm estimate 75
WSMP ICT	Auto	RCM	Drop tolerance 0.003 Fill factor 4.9, SHIFT-ON

Table XI. Iterative solver configurations that resulted in the best overall performance with respect to memory-time product profile area in the 64 processor case.

Preconditioner	**Solver**	Ordering	**Preconditioner Parameters**
PETSc IC(k)	CG	RCM	Fill factor 1, Level of fill 0
PETSc BlockSolve	CG	RCM	-
Trilinos IC(k)	CG	RCM	Level of fill (6)
Trilinos ML	CG	NONE	Smoothed aggregation, Symmetric Gauss-Seidel smoother Smoother sweeps 3, Hybrid Uncoupled-MIS Coarsening
Hypre IC(k)	CG	RCM	Level of fill 1
Hypre BoomerAMG	CG	NONE	PMIS Coarsening, Aggresive coarsening levels 10 Strong threshold 0.7
Hypre ParaSails	CG	ND	Number of levels 1, Threshold 0.1 , Filter 0.001

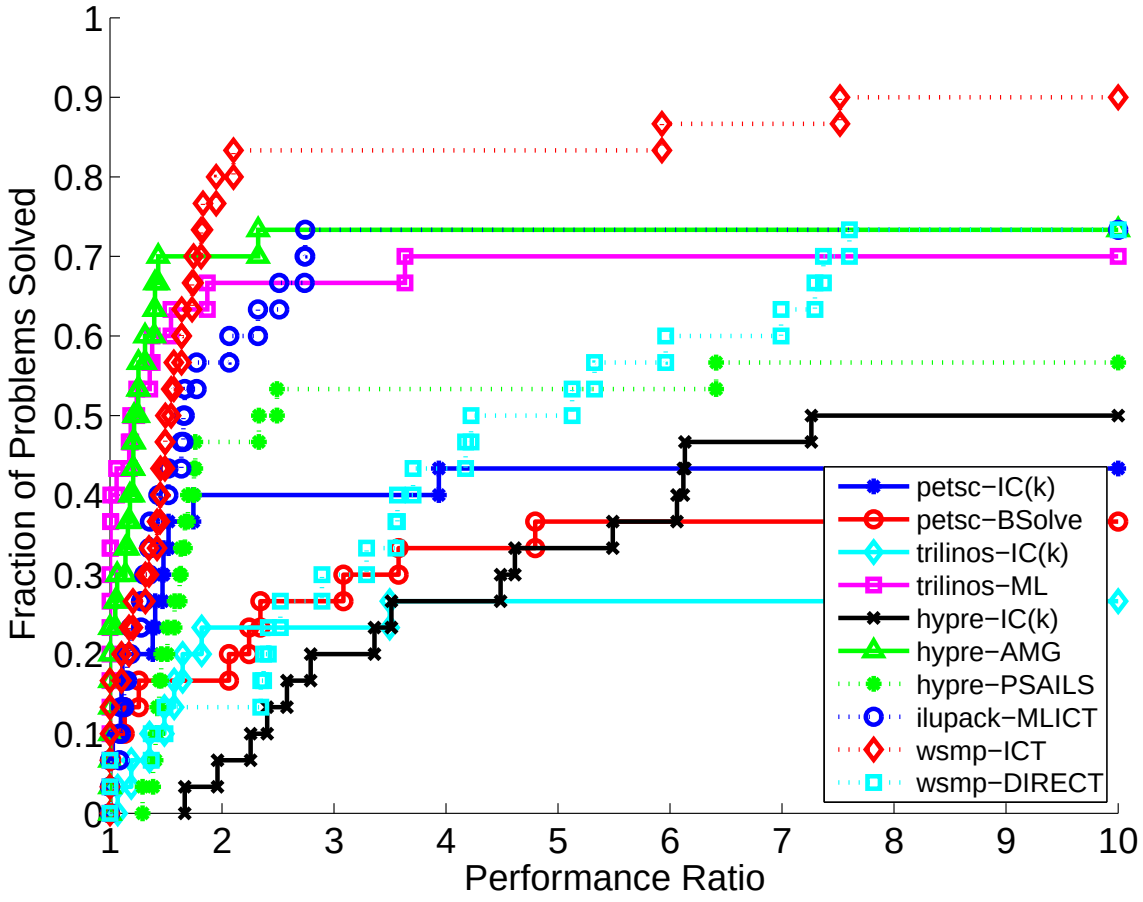


Fig. 32. Memory performance profile curves for the direct solver and the overall best memory-time product configurations of the various IC(k), ICT, AMG, and SAI preconditioner implementations in PETSc, Trilinos, Hypre, ILUPACK, and WSMP for the single processor case.

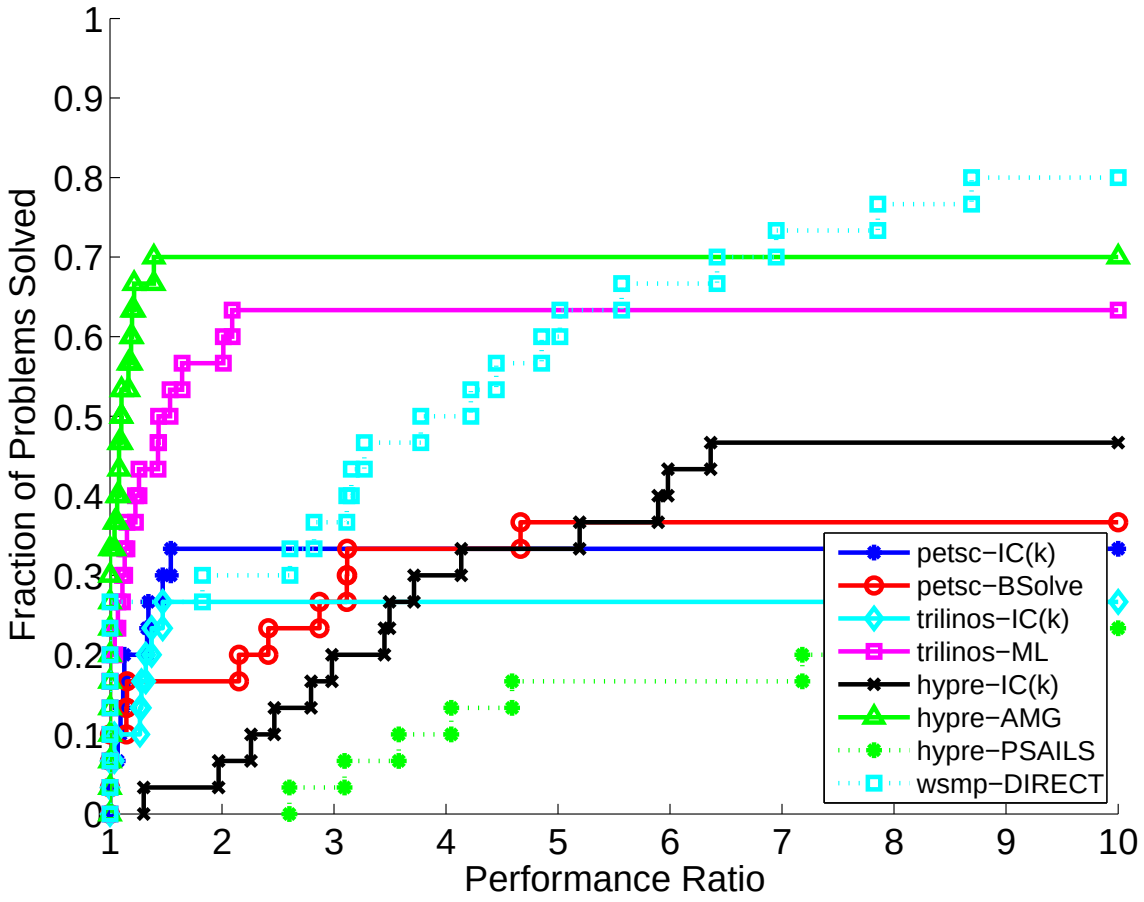


Fig. 33. Memory performance profile curves for the direct solver and the overall best memory-time product configurations of the various IC(k), AMG, and SAI preconditioner implementations in PETSc, Trilinos, and Hypre for the 64 processor case.

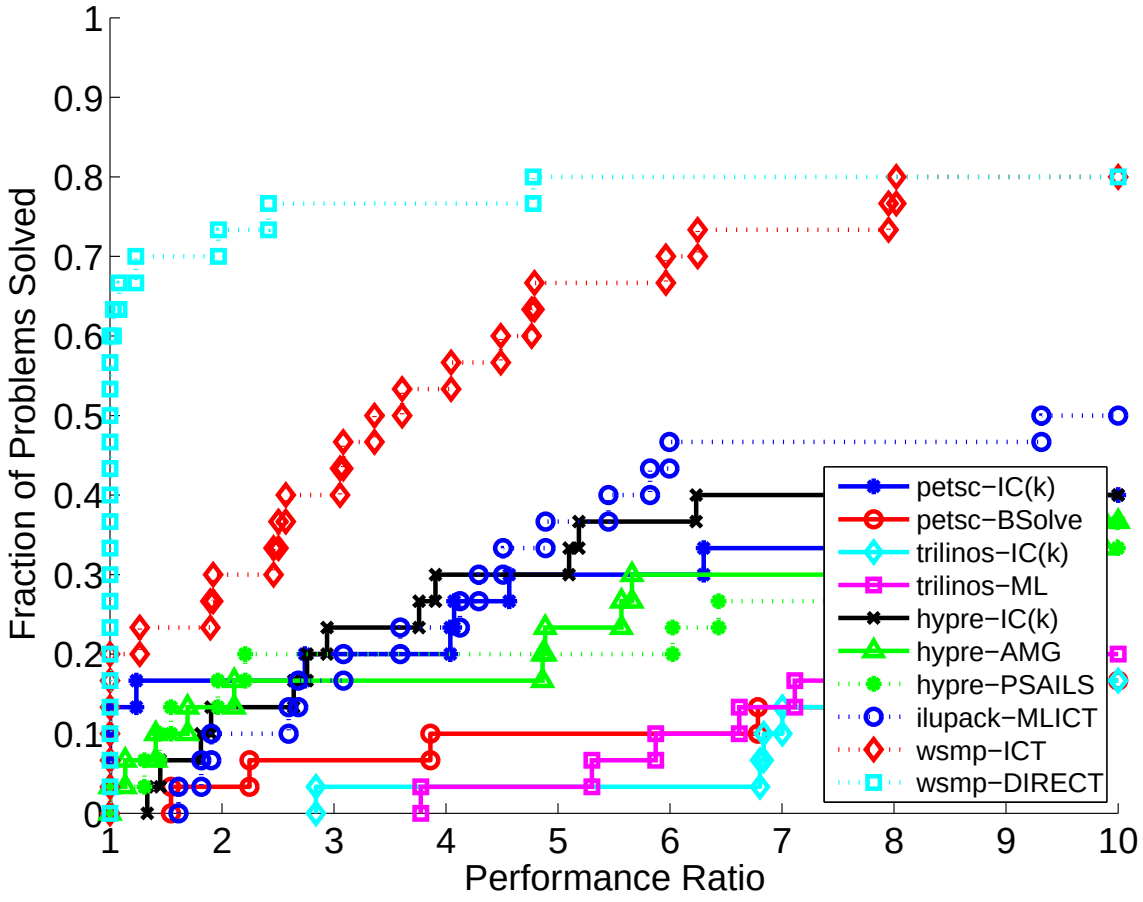


Fig. 34. Time performance profile curves for the direct solver and the overall best memory-time product configurations of the various IC(k), ICT, AMG, and SAI preconditioner implementations in PETSc, Trilinos, Hypre, ILUPACK, and WSMP for the single processor case.

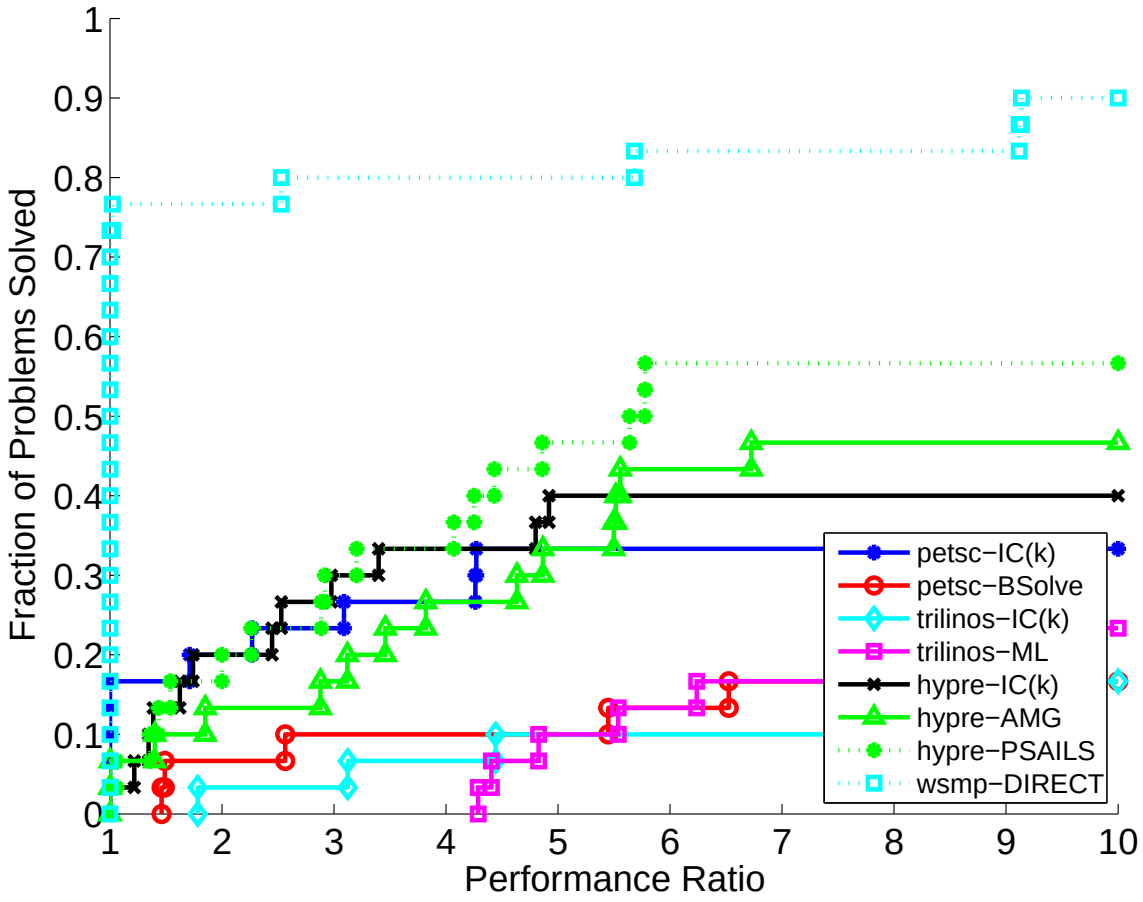


Fig. 35. Time performance profile curves for the direct solver and the overall best memory-time product configuration of the various IC(k), AMG, and SAI preconditioner implementations in PETSc, Trilinos, and Hypre for the 64 processor case.

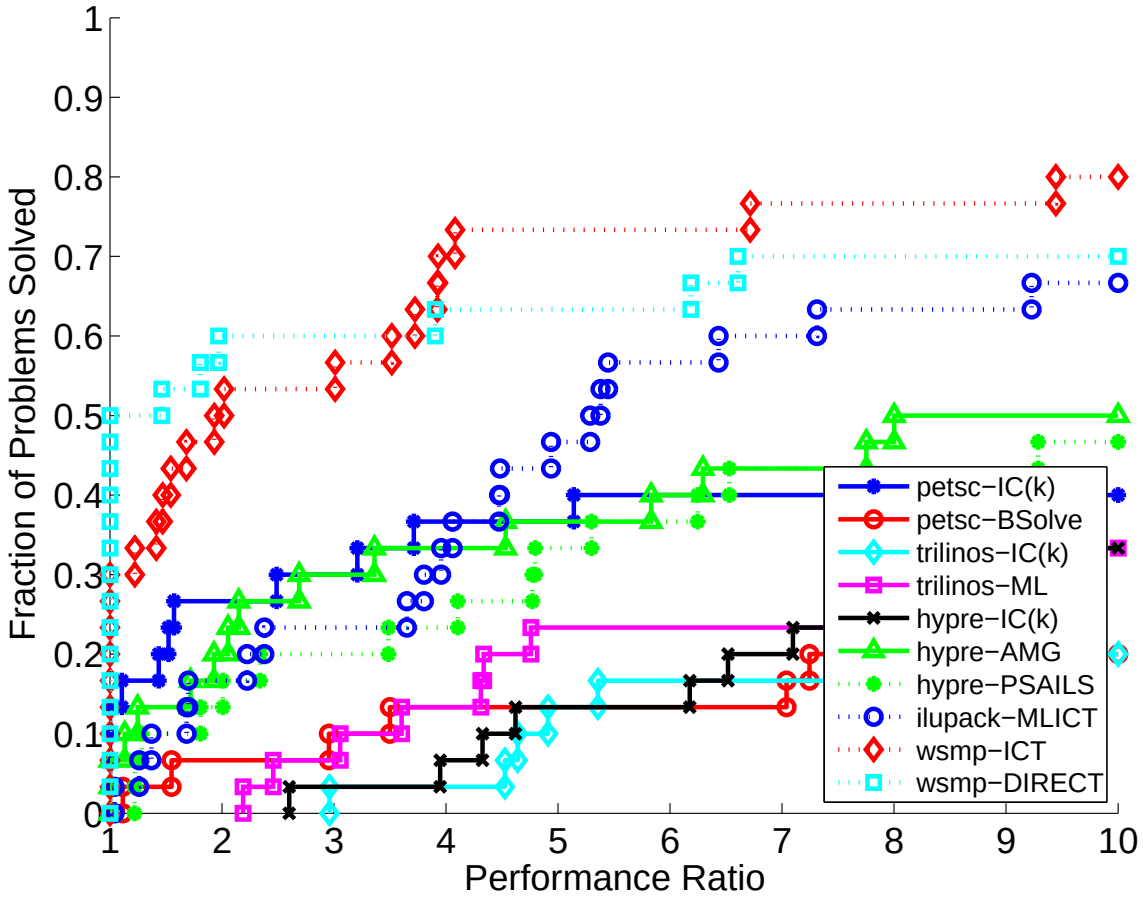


Fig. 36. Memory-time product performance profile curves for the direct solver and the overall best memory-time product configuration of the various IC(k), ICT, AMG, and SAI preconditioner implementations in PETSc, Trilinos, Hypre, ILUPACK, and WSMP for the single processor case.

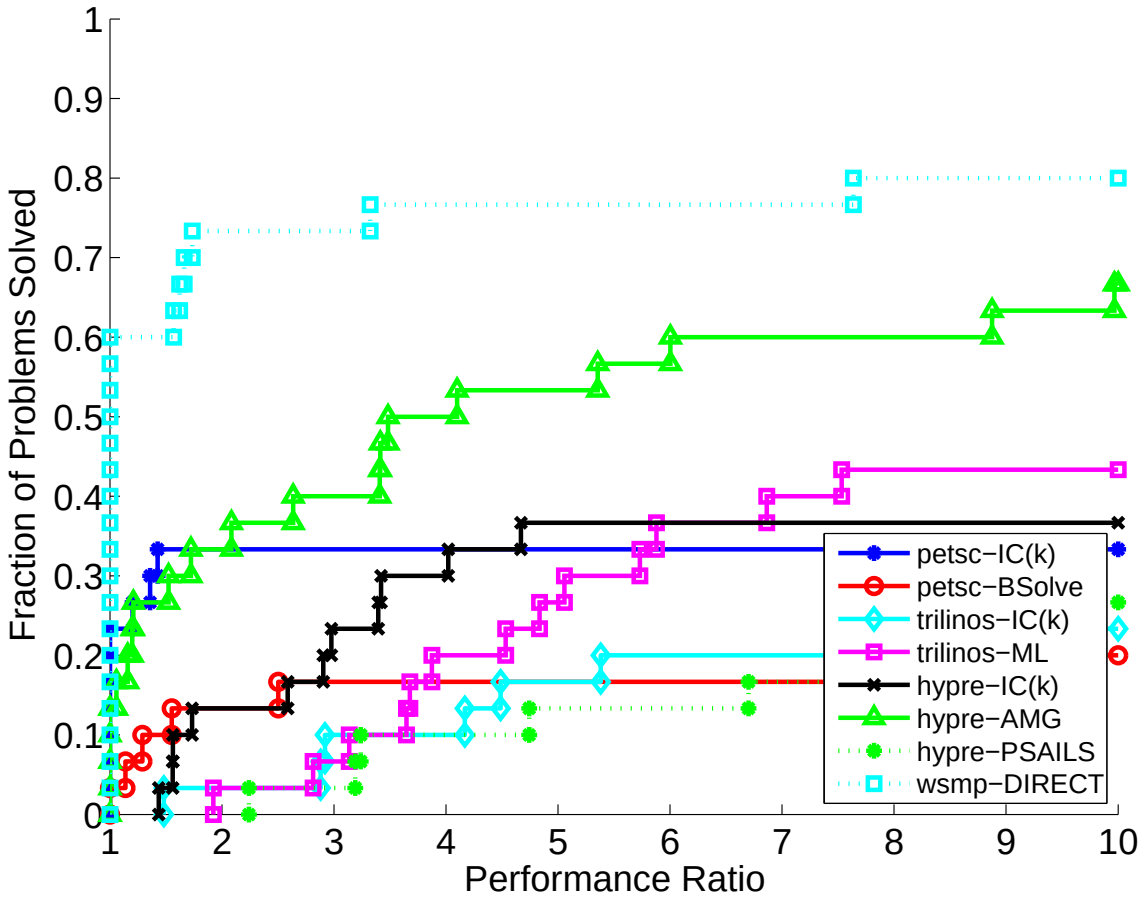


Fig. 37. Memory-time product performance profile curves for the direct solver and the overall best memory-time product configuration of the various IC(k), AMG, and SAI preconditioner implementations in PETSc, Trilinos, and Hypre for the 64 processor case.

Figures 32 and 33 show the memory profiles of the configurations shown in Tables X and XI, respectively. The memory profile of WSMP direct solver is also included in these figures. Recall that the memory plotted here corresponds to the total memory needed for storing the nonzeros in the linear system as well as the memory allocated in the heap during the preconditioner creation.

For the single processor case in Figure 32, Hypre BoomerAMG, Hypre ParaSails, Trilinos ML, WSMP ICT, and ILUPACK MLICT appear to be the most memory efficient and robust. The IC(k) preconditioners are not as robust as the others and their respective curves flatten out fairly early. For the 64 processor case in Figure 33, Hypre BoomerAMG is the most memory efficient followed by Trilinos ML, PETSc IC(k), and Trilinos IC(k). The relative ranking of other preconditioners remains the same except for Hypre ParaSails, which shows a higher memory usage than in the serial case. The high memory usage of Hypre ParaSails is due to the specific implementation choice in which, all the external rows needed by a processor are collected and stored for each processor. While this choice improves the time performance, we observed that its memory consumption increases with the number of processors.

Figures 34 and 35 show the time profiles of the best configurations. In Figure 34, the direct solver turns out to be the fastest solver for about 67% of the problems in the serial case. This is followed by PETSc IC(k), which is the fastest one for about 12% of the problems. However, understandably, PETSc IC(k) does not do as well for more difficult problems and its time profile curve is soon surpassed by that of WSMP ICT. Hypre BoomerAMG and Trilinos ML, which are highly memory efficient, appear to be slower than Hypre ParaSails, ILUPACK, and WSMP ICT. For the 64 processor case in Figure 35, the direct solver is still the fastest for about 70% of the problems followed by Hypre IC(k), PETSc IC(k), Hypre ParaSails, and Hypre BoomerAMG.

Figures 36 and 37 show the memory-time product profiles for the various solver con-

figuration groups along with that of the WSMP direct solver. The relative positions of most memory-time product profiles in the serial case are very similar to that of the corresponding time profiles in Figures 34 and 35. In the 64 processor case, the curve for Hypre Boomer-AMG moves up because of its excellent memory efficiency and that for Hypre ParaSails curve moves down due to its high memory usage in the multi-processor case.

A comparison of the memory and time performance of the iterative solvers relative to WSMP's direct solver confirms the conventional wisdom that direct solvers are generally fast and robust, but require more memory resources. Conventional wisdom also holds that the preconditioned iterative solvers should outperform the direct solver on larger problems. In addition, the performance crossover point between iterative and direct solvers would be observed for relatively larger matrices that result from two dimensional physical problems as compared to three dimensional ones. Our results simply indicate that, although half of the problems in our test suite have more than half a million unknowns, the average problem size is still too small for most iterative solvers to outperform the direct solver in terms of solution time.

b. Problem Specific Parameter Selection

While the overall best or the PIB configuration of a preconditioner offers a good choice of parameter settings for an arbitrary problem, users may be able to improve the performance of their applications by tuning the parameters for the matrices arising in their applications. In this section, we discuss the relative performance of various preconditioner implementations when the best parameter configuration is chosen individually for each problem from a reasonably comprehensive set of configurations. This analysis can give a good indication of the best possible performance that a preconditioner is capable of delivering for each problem. While it is not practical to fine tune the parameters for each individual problem, fine-tuning can be useful when all matrices arising in a particular application have similar

properties.

Figures 38 and 39 show the memory profiles (for 1 and 64 processors, respectively) when the parameter configuration for each problem was chosen individually to minimize its memory-time product. These figures show that problem specific fine-tuning results in remarkable improvements in memory use for most preconditioners, when compared with the best overall parameter configuration. All iterative solver curves move upwards with respect to the direct solver curve in Figures 38 and 39. Besides consuming less memory, most preconditioners are able to solve more problems successfully with problem-specific parameter tuning. The most remarkable improvement with respect to memory occurs for Hypre IC(k).

Figures 40 and 41 show the time profiles of all the preconditioners when the parameter configuration for each problem was chosen individually to minimize its memory-time product. Just like the memory profiles, the time profiles of the preconditioners improve significantly when compared to those for the overall best parameter configuration. The most notable improvements in the serial case are for ILUPACK MLICT, Hypre IC(k) and Hypre ParaSails. While using 64 processors, a comparison of the figures shows that Hypre's BoomerAMG, IC(k), and ParaSails reduce the time performance gap with the WSMP direct solver.

Tables XII and XIII show best iterative solver configuration and its time and memory consumption for each problem for the 1- and 64-processor case. These tables also show the time and memory used by the direct solver in each case. The values corresponding to the best memory-time product are in bold font. In the single processor case, the direct solver has the better memory-time product for 12 out of the 30 matrices. As expected, the iterative solvers do much better for large 3-D problems. Among the iterative solvers, WSMP ICT does best for 13 problems, PETSc IC(k) and Hypre BoomerAMG for 2 problems each, and Hypre ParaSails for one problem. For the 64 processor case shown in Table XIII, the

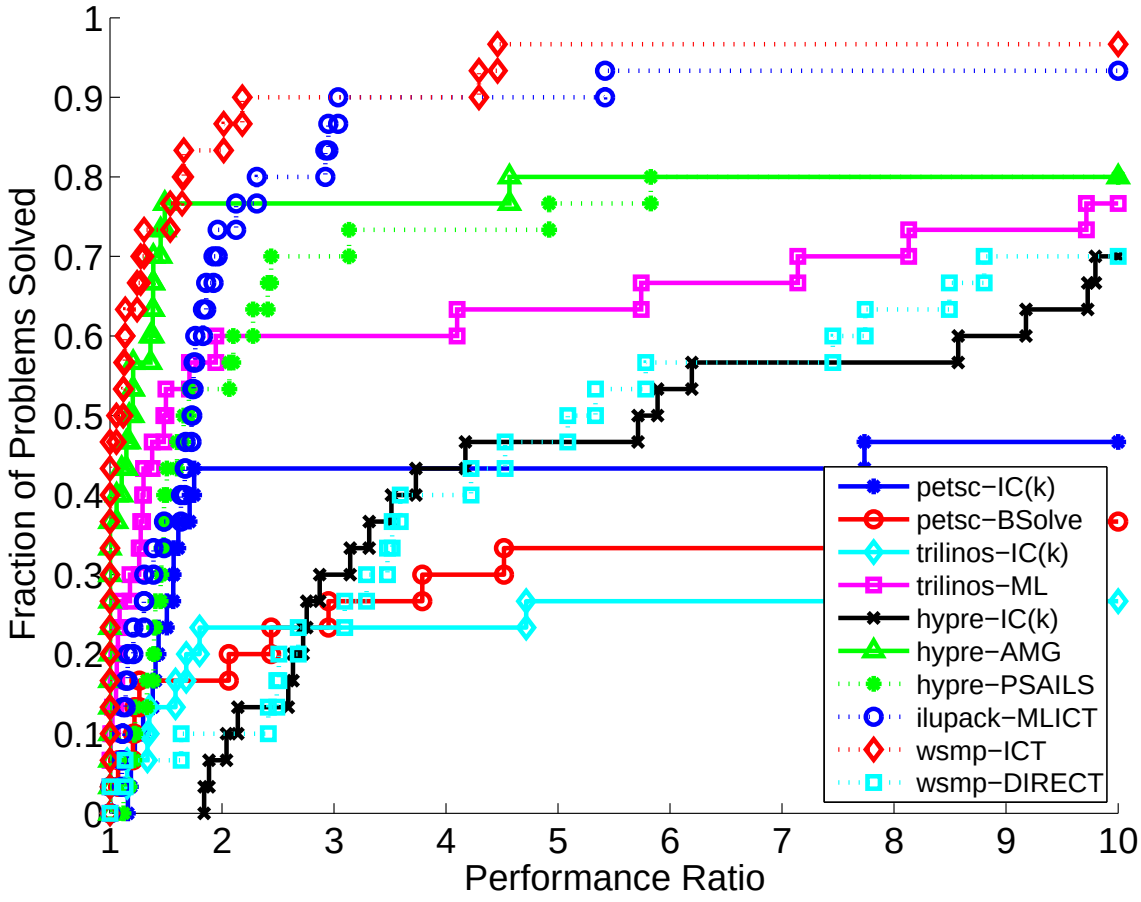


Fig. 38. Memory performance profiles for the direct solver and the memory values corresponding to the best problem specific memory-time product configuration of the various IC(k), ICT, AMG, and SAI preconditioner implementations in PETSc, Trilinos, Hypre, ILUPACK, and WSMP in the single processor case.

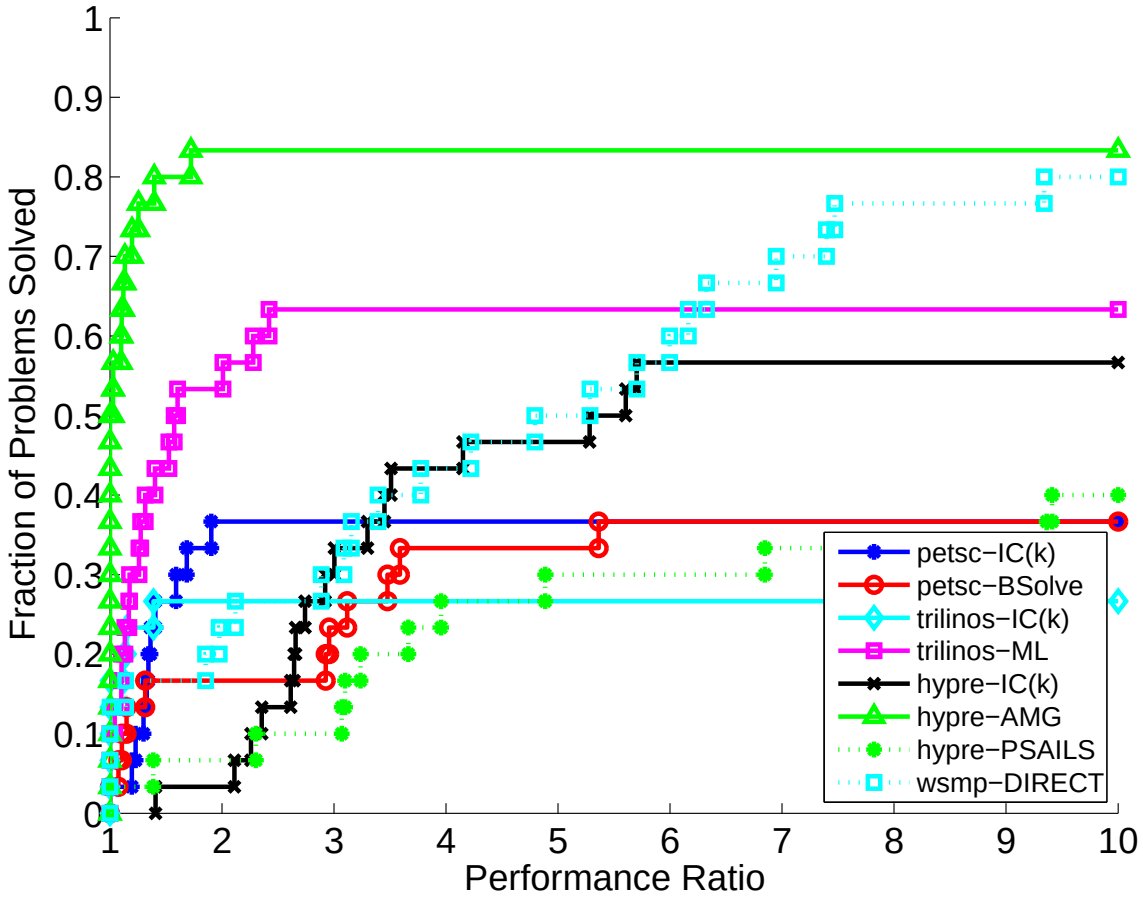


Fig. 39. Memory performance profiles for the direct solver and the memory values corresponding to the best problem specific memory-time product configuration of the various IC(k), AMG, and SAI preconditioner implementations in PETSc, Trilinos, and Hypre in the 64 processor case.

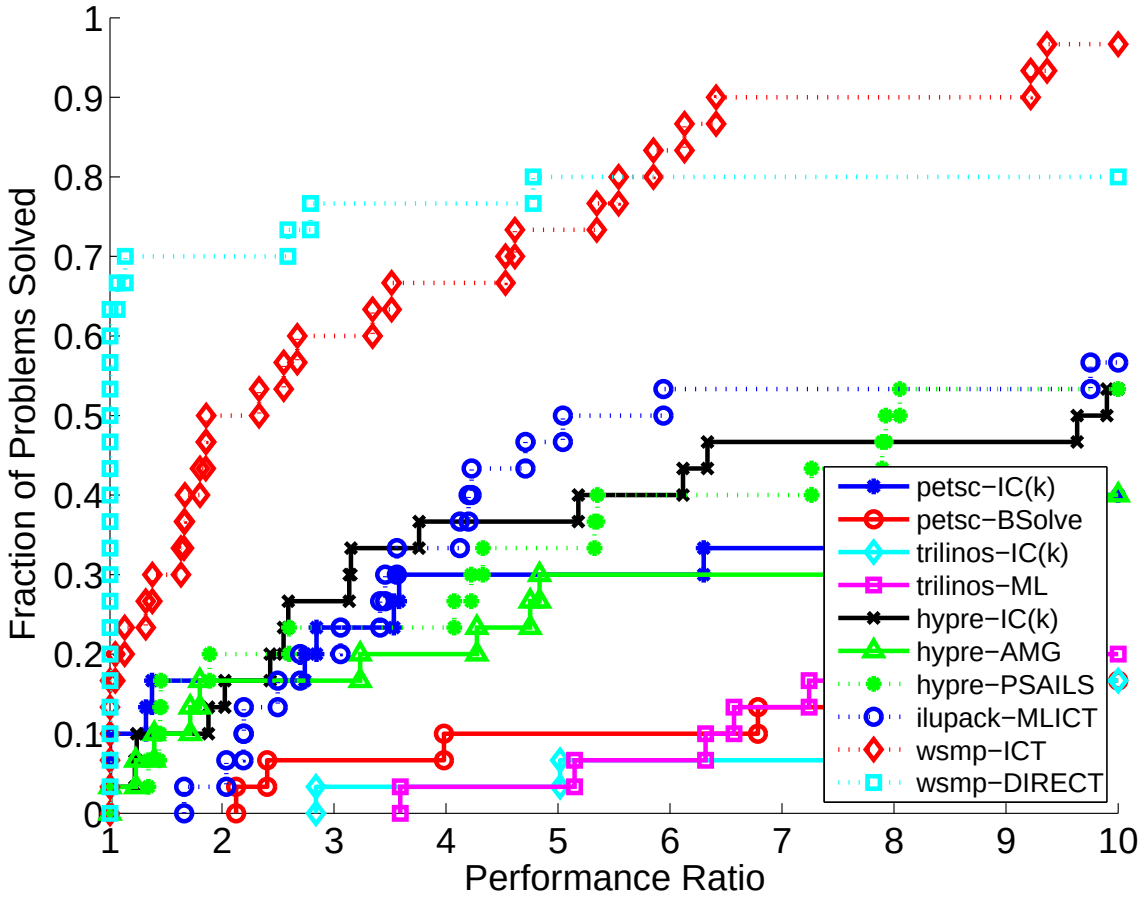


Fig. 40. Time performance profiles for the direct solver and the time values corresponding to the best problem specific memory-time product configuration of the various IC(k), ICT, AMG, and SAI preconditioner implementations in PETSc, Trilinos, Hypre, ILUPACK, and WSMP in the single processor case.

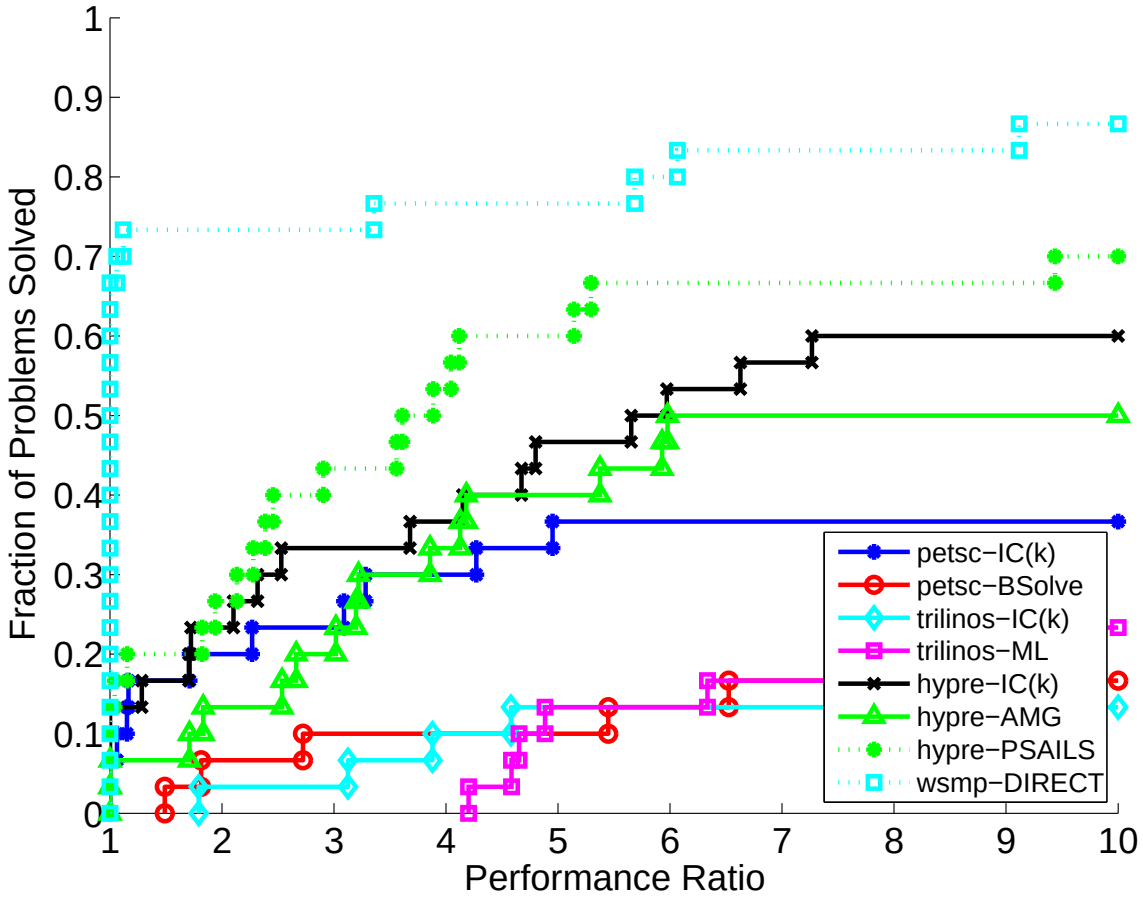


Fig. 41. Time performance profiles for the direct solver and the time values corresponding to the best problem specific memory-time product configuration of the various IC(k), AMG, and SAI preconditioner implementations in PETSc, Trilinos, and Hypre in the 64 processor case.

Table XII. Table showing the time (in seconds) and memory values (in megabytes) corresponding to the best problem specific memory-time product for iterative and direct solvers in the single processor case. The bold values indicate the solver configuration for which the product of memory and time was the lowest.

Matrix	Iterative Parameter	Iter. Mem	Iter. Time	Dir. Mem	Dir. Time
90153	wsmp, ICT, AUTO, RCM, DT1e-2, F2.5, ON	**55.2**	**6.61**	194	2.47
af_shell7	wsmp, ICT, AUTO, RCM, DT1e-2, F2.5, OFF	223	62.80	**830**	**11.75**
algor-big	wsmp, ICT, AUTO, RCM, DT1e-2, F4.9, OFF	**788**	**305.73**	-	-
audikw_1	wsmp, ICT, AUTO, RCM, DT1e-2, F2.5, ON	**959**	**336.16**	9500	870.00
bmwcra_1	wsmp, ICT, AUTO, RCM, DT1e-2, F2.5, ON	**110**	**51.10**	568	11.28
ctu-1	wsmp, ICT, AUTO, ND, DT3e-4, F2.5, OFF	2830	531.93	**3210**	**86.78**
ctu-2	wsmp, ICT, AUTO, ND, DT3e-3, F4.1, ON	936	462.96	**2350**	**100.30**
cfd1	petsc, IC(k), CG, RCM, LF0, F1	**29.1**	**6.59**	157	2.41
cfd2	hypre, PSAILS, CG, NONE, PLev1, Th.1, Flt.05	80.1	26.81	**310**	**6.35**
conti20	wsmp, ICT, AUTO, RCM, DT3e-3, F3.3, ON	35.9	5.63	**64**	**0.96**
garybig	hypre, AMG, CG, RCM, FALG, AGG10, ST.7	**6660**	**11597**	-	-
G3_circuit	wsmp, ICT, AUTO, RCM, DT1e-2, F2.5, OFF	**179**	**37.39**	956	20.11
hood	wsmp, ICT, AUTO, ND, DT3e-3, F4.1, OFF	170	4.10	**257**	**2.51**
inline_1	wsmp, ICT, AUTO, RCM, DT3e-4, F2.5, OFF	1910	91.75	**1500**	**27.43**
kyushu	petsc, IC(k), CG, RCM, LF0, F1	**417**	**32.66**	9220	1336.48
ldoor	wsmp, ICT, AUTO, ND, DT3e-4, F4.9, OFF	965	40.51	**1370**	**22.45**
msdoor	wsmp, ICT, AUTO, ND, DT3e-4, F4.9, OFF	499	28.39	**492**	**5.12**
mstamp-2c	hypre, PSAILS, CG, RCM, PLev0, Th.1, Flt0	**726**	**76.35**	-	-
nastran-b	wsmp, ICT, AUTO, RCM, DT1e-2, F4.1, OFF	**1160**	**474.16**	8620	538.88
nd24k	wsmp, ICT, AUTO, RCM, DT1e-2, F2.5, ON	**88.3**	**23.61**	2750	412.18
oilpan	wsmp, ICT, AUTO, ND, DT3e-4, F2.5, OFF	60.3	2.38	**94.5**	**1.02**
parabolic_fem	wsmp, ICT, AUTO, RCM, DT1e-2, F2.5, ON	**70.8**	**7.39**	233	2.90
pga-rem1	wsmp, ICT, AUTO, RCM, DT1e-2, F4.9, OFF	277	39.03	**742**	**11.10**
pga-rem2	wsmp, ICT, AUTO, RCM, DT1e-2, F2.5, OFF	**437**	**74.17**	1980	44.63
qa8fk	hypre, AMG, CG, RCM, PMIS, AGG10, ST.25	**17.9**	**2.03**	193	4.60
qa8fm	wsmp, ICT, AUTO, RCM, DT1e-2, F2.5, OFF	**3.43**	**0.16**	187	4.21
ship_003	wsmp, ICT, AUTO, RCM, DT1e-2, F2.5, ON	**62.2**	**17.66**	529	16.85
shipsec5	wsmp, ICT, AUTO, RCM, DT1e-2, F3.3, ON	**88**	**12.15**	448	12.93
thermal2	wsmp, ICT, AUTO, RCM, DT1e-2, F2.5, OFF	139	59.36	**484**	**6.44**
torso	wsmp, ICT, AUTO, RCM, DT1e-2, F2.5, ON	**47.6**	**5.79**	677	24.43

direct solver does best for 15 problems. Among the iterative solvers, Hypre BoomerAMG does best for 7 problems, PETSc IC(k) for 6 problems, and Hypre ParaSails and PETSc BlockSolve for one problem each. Note that WSMP ICT and ILUPACK do not yet have parallel implementations suitable for 64 processors.

c. Relative Strengths of Preconditioners and Sensitivity to Parameter Tuning

We have observed that different preconditioners and solvers have different strengths and weaknesses. Some are more memory efficient than others, while some are faster than oth-

Table XIII. Table showing the time (in seconds) and memory values (in megabytes) corresponding to the best problem specific memory-time product for iterative and direct solvers in the 64 processor case. The bold values indicate the solver configuration for which the product of memory and time was the lowest.

Matrix	Iterative Parameter	Iter. Mem	Iter. Time	Dir. Mem	Dir. Time
90153	hypre, IC(k), CG, ND, LF4, NzINF	352	0.84	**195**	**0.13**
af_shell7	hypre, AMG, CG, RCM, PMIS, AGG10, ST.9	240	1.88	**848**	**0.45**
algor-big	hypre, AMG, CG, NONE, PMIS, AGG10, ST.7	**987**	**9.95**	32200	90.41
audikw_1	hypre, AMG, CG, RCM, FALG, AGG0, ST.9	**1540**	**86.63**	9750	26.91
bmwcra_1	hypre, AMG, CG, RCM, PMIS, AGG10, ST.25	100	6.70	**572**	**0.42**
ctu-1	-	-	-	**3460**	**4.64**
ctu-2	-	-	-	**2320**	**2.51**
cfd1	petsc, IC(k), CG, RCM, LF0, F1	**28.8**	**0.28**	164	0.16
cfd2	hypre, AMG, CG, NONE, PMIS, AGG10, ST.7	44.1	5.30	**306**	**0.29**
conti20	-	-	-	**67.2**	**0.08**
garybig	hypre, AMG, CG, ND, FALG, AGG10, ST.7	**6680**	**263.61**	-	-
G3_circuit	petsc, IC(k), CG, RCM, LF0, F1	**187**	**2.12**	939	0.69
hood	petsc, BSolve, CG, RCM, ALL, NONE	116	0.95	**303**	**0.15**
inline_1	-	-	-	**1530**	**1.33**
kyushu	petsc, IC(k), CG, RCM, LF0, F1	**409**	**2.68**	9220	34.49
ldoor	hypre, PSAILS, CG, ND, PLev0, Th0, Flt.05	1760	2.99	**1510**	**0.83**
msdoor	hypre, PSAILS, CG, ND, PLev1, Th0, Flt.05	1520	4.08	**558**	**0.27**
mstamp-2c	petsc, IC(k), CG, ND, LF0, F1	**1000**	**2.01**	15900	62.79
nastran-b	hypre, AMG, CG, RCM, PMIS, AGG10, ST.9	**1350**	**79.26**	9130	19.21
nd24k	hypre, PSAILS, CG, NONE, PLev2, Th.1, Flt.05	**1450**	**1.76**	2680	10.66
oilpan	hypre, AMG, CG, RCM, HMIS, AGG0, ST.9	48.5	6.09	**103**	**0.06**
parabolic_fem	hypre, AMG, CG, NONE, HMIS, AGG10, ST.25	**76.6**	**0.41**	235	0.16
pga-rem1	hypre, IC(k), CG, RCM, LF1, NzINF	525	2.36	**736**	**0.49**
pga-rem2	petsc, IC(k), CG, RCM, LF0, F1	**535**	**5.09**	2020	2.24
qa8fk	hypre, AMG, CG, NONE, HMIS, AGG10, ST.25	**19.9**	**0.16**	186	0.20
qa8fm	hypre, AMG, CG, RCM, PMIS, AGG0, ST.25	**17**	**0.03**	185	0.20
ship_003	hypre, AMG, CG, RCM, HMIS, AGG10, ST.25	90.9	7.91	**560**	**0.76**
shipsec5	petsc, BSolve, CG, RCM, ALL, NONE	**113**	**1.31**	505	0.51
thermal2	hypre, AMG, CG, ND, PMIS, AGG10, ST.25	159	1.37	**492**	**0.36**
torso	petsc, IC(k), CG, RCM, LF0, F1	**54**	**0.16**	685	0.91

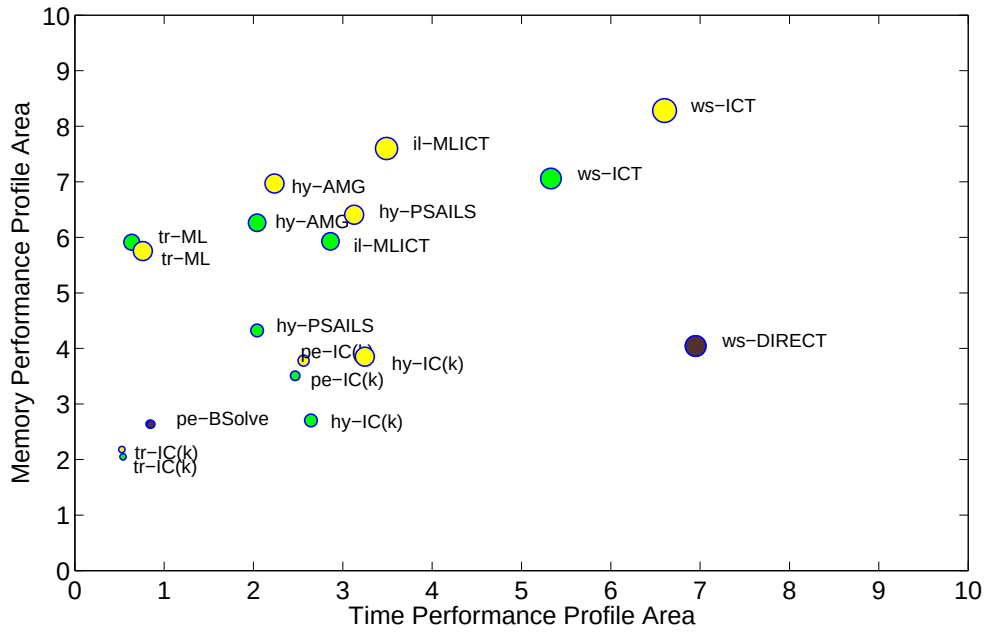


Fig. 42. Plot of the time profile area versus the memory profile area for various preconditioner implementations (single processor case). Each circle represents a preconditioner whose name consists of the first two letters of the name of the package followed by the type of preconditioner. The size of a circle is proportional to the number of problems solved. The green (dark) circles correspond to profile areas for the default parameter configuration and the yellow (light) ones correspond to profile areas for problem-specific best parameters. If the yellow and green circles overlap, it is shown as a brown circle.

ers. They also have different degrees of robustness. In Section 2, we also saw that, as expected, most preconditioners performed significantly better when their parameters were permitted to be tuned to each coefficient matrix. However, different preconditioners displayed different degrees of improvement. Figure 42 displays all this relative information about the performance of various preconditioners by means of a single information-rich graphic. The figure has two sets of circles for each preconditioner. The green (dark) circles correspond to the default parameter configurations. The yellow (light) circles correspond to the corresponding problem-specific best parameters. The x- and y-coordinates of each circle are the areas under the time and memory profile curves of the corresponding preconditioner derived from plotting the default and problem specific performance profiles in a single plot. The size of each circle is proportional to the number of problems solved.

The height of a circle in Figure 42 is indicative of the memory efficiency of the corresponding preconditioner. Similarly, the distance from the y-axis towards the right is indicative of its speed. The figure shows at a glance which solvers and preconditioners are most memory efficient and which ones are most time efficient. For example, in Figure 42 the direct solver is very fast and robust, but is less memory efficient than many of the preconditioners. On the other hand, Hypre BoomerAMG is very robust and memory efficient, but is relatively slow. For the default parameters, Hypre Parasails and ILUPACK are faster, but use slightly more memory than Hypre BoomerAMG. WSMP ICT with default parameters is fairly memory efficient and significantly faster. Some preconditioners benefit a great deal from parameter tuning. This is evident from the fact the yellow (light) circles corresponding to most preconditioners lie above and to the right of their dark counterparts. The most remarkable improvement can be seen in the case of Hypre IC(k). ILUPACK MLICT and Hypre ParaSails also show significant improvement in both time and memory. Another interesting observation from Figure 42 is that the time, memory, and robustness of different implementations of the same underlying preconditioning method can be very

different. Whether with default or with fine-tuned parameters, the best preconditioner implementations lie on the periphery of the plot. When looking for candidates for the best preconditioner for an application at hand, a user is likely to fare best by picking one of Hypre BoomerAMG, ILUPACK, Hypre Parasails, WSMP ICT or WSMP Direct solver, depending on the desired balance between computation time and memory. PETSc IC(k) and Hypre IC(k) also emerge as strong preconditioners in terms of memory and time efficiency, although they are able to solve fewer problems compared to the other leading preconditioners.

In Figure 43, we compare the relative performance of the preconditioners in the 64 processor case. Just like the serial case, the direct solver is very fast but memory intensive in comparison to other preconditioners. Hypre BoomerAMG is relatively more efficient with respect to both time and memory and the benefits of fine-tuning are also somewhat more pronounced than in 42. Hypre IC(k) outperforms PETSc IC(k) and Trilinos IC(k) with respect to both memory and time. The relative time efficiency of Hypre ParaSails increases, but it comes at the expense of increased memory usage. To summarize, Hypre BoomerAMG is highly memory efficient, WSMP direct and Hypre ParaSails are relatively fast whereas Hypre IC(k) seems to balance both time and memory.

5. Parallel Efficiency

An important measure often reported for parallel implementations is the time efficiency across multiple processors. Most of the packages in this study are used for large scale scientific simulations involving thousands of processors and might exhibit excellent weak scaling. Since the matrices in our test are of fixed size, we only perform strong scaling analysis. The efficiency obtained during a weak scaling study will be much better than that observed for our strong scaling study. Since efficiency is measured using the ratio of the serial time to the parallel time, the values depend heavily on the serial implementation of

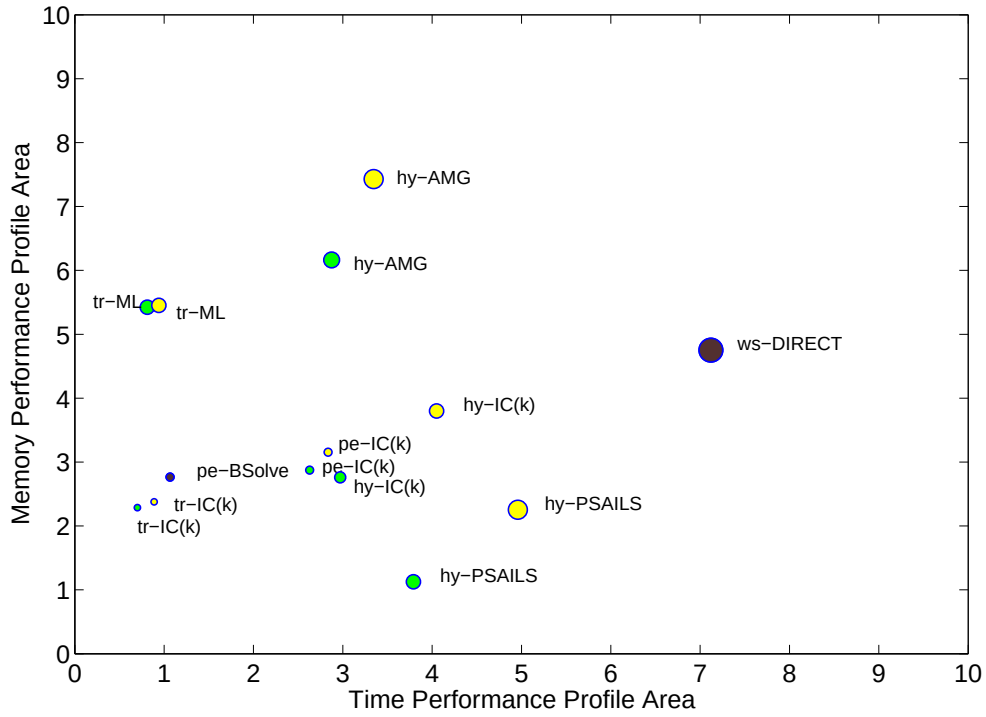


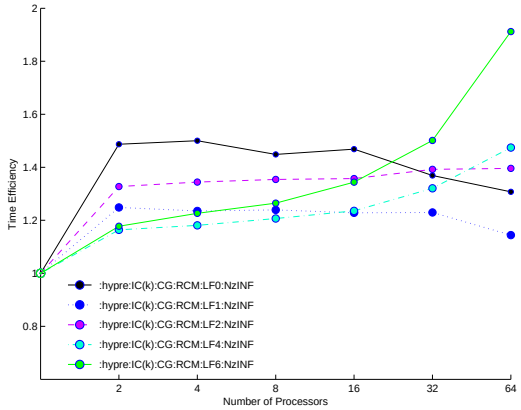
Fig. 43. Plot of the time profile area versus the memory profile area for various preconditioner implementations (64 processor case). Each circle represents a preconditioner whose name consists of the first two letters of the name of the package followed by the type of preconditioner. The size of a circle is proportional to the number of problems solved. The green (dark) circles correspond to profile areas for the default parameter configuration and the yellow (light) ones correspond to profile areas for problem-specific best parameters. If the yellow and green circles overlap, it is shown as a brown circle.

the respective preconditioners. Therefore, for each configuration group, we first determine the solver configurations that resulted in the best serial MTP over all the problems. If a particular problem is solved using the best serial MTP solver configuration on all the different processor configurations, then it is included for calculating the average efficiency.

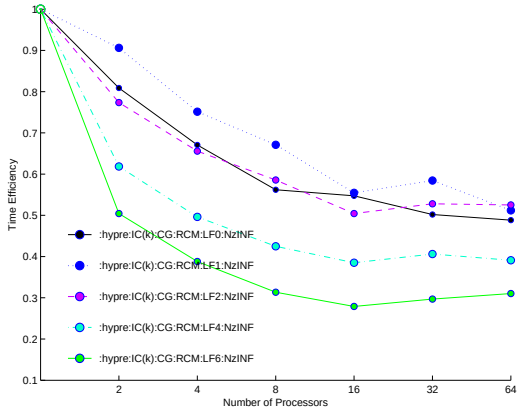
The rest of this section is organized as follows. We first study the variation of time efficiency with respect to preconditioner sparsity for the preconditioner generation phase, iterative solution phase, and the total solution time for IC(k), BoomerAMG, and ParaSails preconditioners in the Hypre package. This is followed by a comparison of the average efficiencies across the various configuration groups composed of various package and preconditioner combinations.

a. Effect of Efficiency on Preconditioner Density

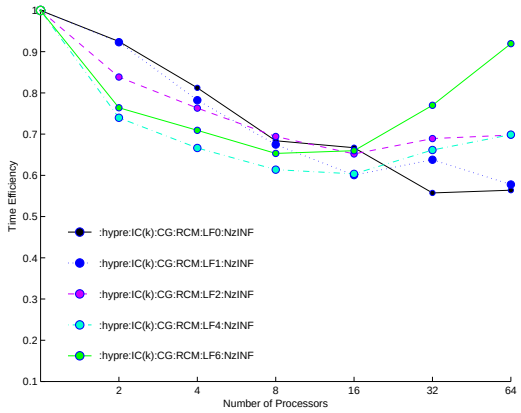
Hypre IC(k): Figure 44(a) shows the average efficiency curves for the preconditioner generation phase for the various level of fill values in Hypre IC(k). One can observe superlinear speedup for most fill factor values. The high fill factor values which typically result in dense preconditioners have a lower average efficiency value than the sparser preconditioners for low number of processors. For higher number of processors, the densest preconditioners shows the maximum super linear speedup. However, the iterative solution phase in Figure 44(b) shows a progressive drop in efficiency with increasing number of processors for all the level of fill values. The drop in efficiency is highest for the densest preconditioner. Figure 44(c) shows the average efficiency for the combined preconditioned generation and iterative solution phases. Since the overall time efficiency and preconditioner generation time efficiency are fairly similar, we can assume that the iterative solution phase is only a small fraction of the overall time. This partly explains the progressive drop in efficiency observed in Figure 44(b) in comparison to the preconditioner generation time and the overall time.



(a) Preconditioner generation phase.

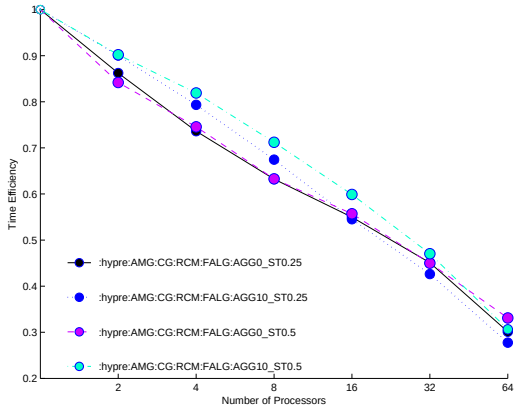


(b) Solution phase.

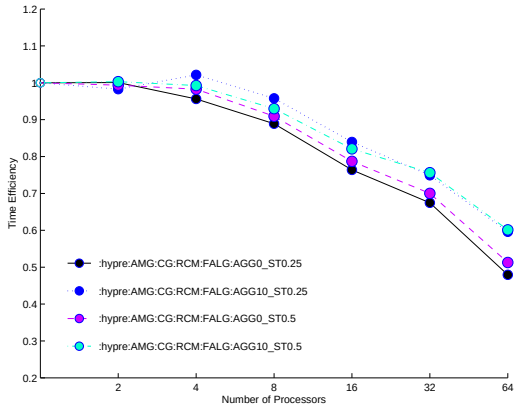


(c) Overall.

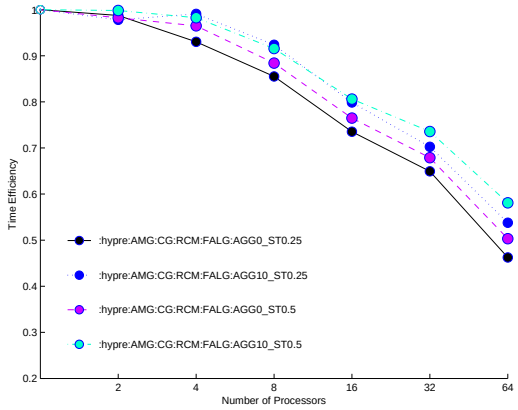
Fig. 44. Average time efficiency for the preconditioner generation phase, the iterative solution phase, and the overall time of Hypre IC(k) for various level of fill values and RCM ordering.



(a) Preconditioner generation phase.

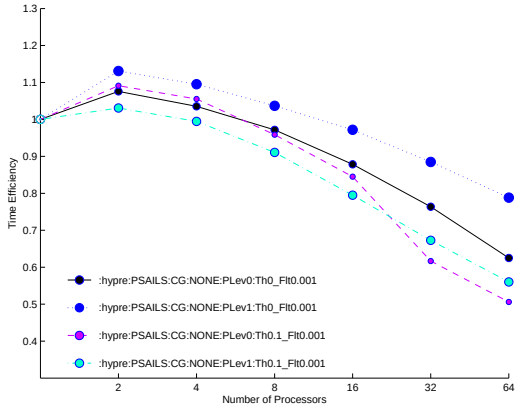


(b) Solution phase.

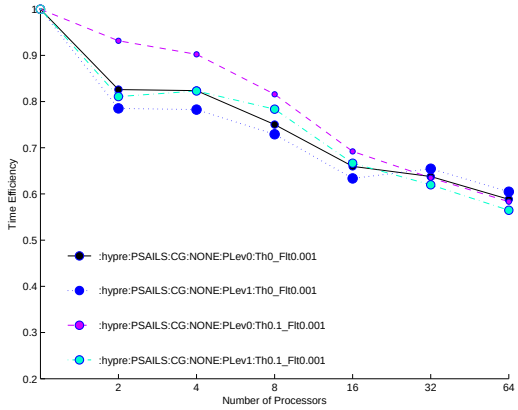


(c) Overall.

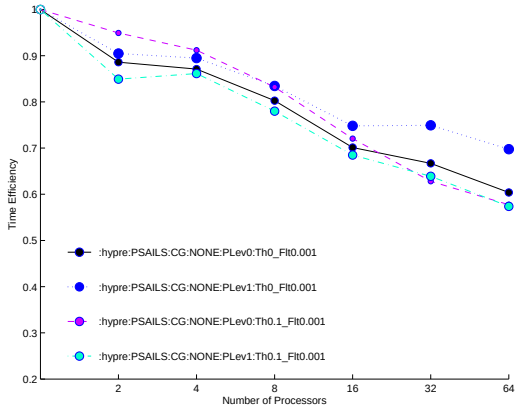
Fig. 45. Average time efficiency for the preconditioner generation phase, the iterative solution phase, and the overall time of Hypre BoomerAMG for strong threshold values (ST0.25, ST0.5) for multiple aggressive coarsening levels (AGG0, AGG10).



(a) Preconditioner generation phase.



(b) Solution phase.



(c) Overall.

Fig. 46. Average time efficiency for the preconditioner generation phase, the iterative solution phase, and the overall time of Hypre ParaSails for various number of levels (Lev0, Lev1) and multiple threshold values (Th0, Th0.1).

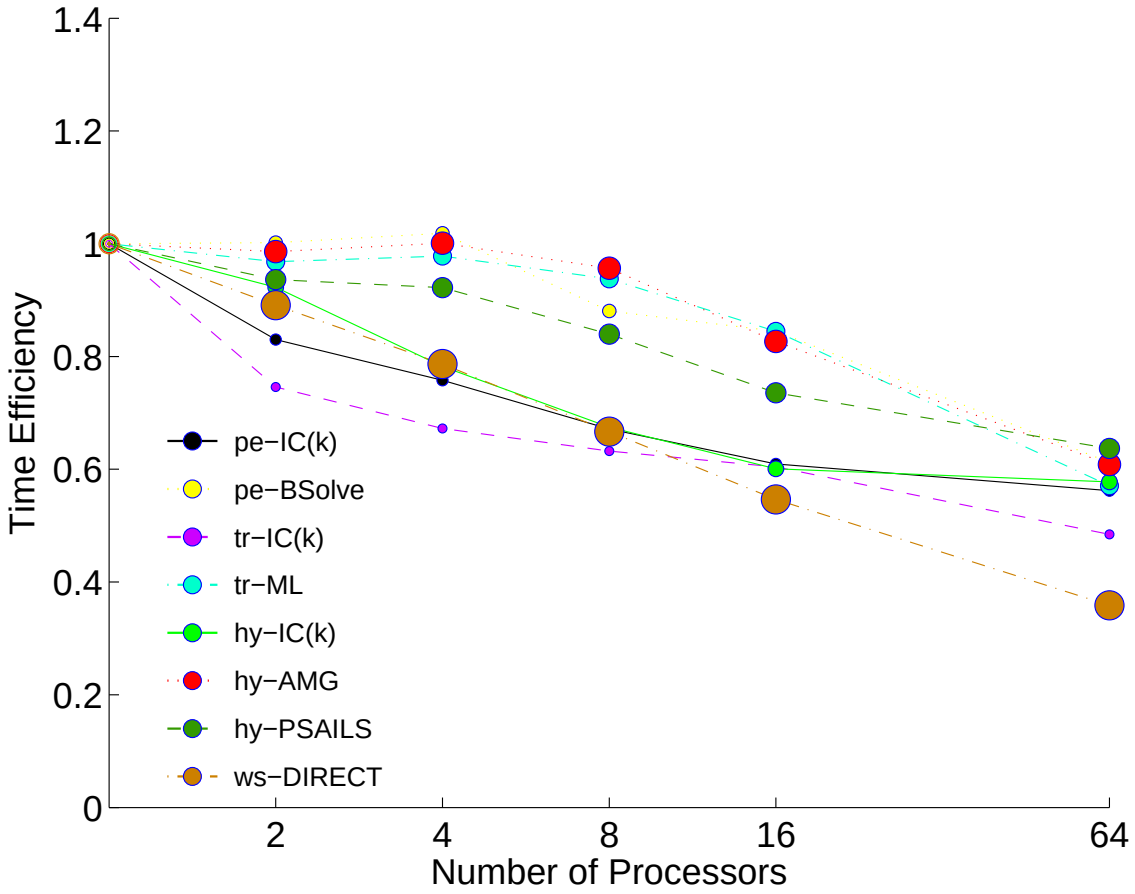


Fig. 47. Average time efficiency corresponding to the PIB parameters of the various preconditioner implementations. The legend names consists of the first two letters of the name of the package followed by the type of preconditioner. The size of a circle is proportional to the number of problems solved.

Hypre BoomerAMG: In Section 1, we observed that the use of high aggressive coarsening levels and high threshold values result in less expensive preconditioners. In order to choose four solver configurations that result in preconditioners of varying sparsity, we varied the aggressive coarsening levels and the strong threshold values while keeping other parameters fixed. The densest preconditioners correspond to those with no aggressive coarsening (AGG0) and low values of threshold (ST0.25). Figure 45(a) shows the average efficiency curves for the preconditioner generation phase. For lower number of processors, the denser

preconditioners exhibit slightly more drop in efficiency, however, as the number of processors increases, all the curves are fairly close. The drop in efficiency for the iterative solution phase shown in Figure 45(b) is not as much as in the preconditioner generation phase. Similar behavior is observed even in the case of average efficiency curves for the overall time, i.e., the sparse preconditioners show lesser drop in efficiency in comparison to dense preconditioners.

Hypre ParaSails: As observed in our earlier analysis, lower number of levels and higher threshold values lead to sparser preconditioners. In order to show the effect of preconditioner density on the parallel time efficiency, we chose ParaSails configurations with varying number of levels and threshold values, while keeping the filter parameter fixed to 0.001. Figure 46 shows the average time efficiency values for the preconditioner generation phase, iterative solution phase, and overall time respectively. The effect of preconditioner density on the efficiency is more clearly seen in the case of nonzero values of the level parameter since impact of higher values of threshold on density is more significant for this case.

In Figure 46(a), we observe that the drop in efficiency for the preconditioner generation phase is more for sparser preconditioners whereas the opposite behavior is observed in Figure 46(b) for the iterative solution phase. However, the overall time efficiency curves in Figure 46(c) seems to be in between that of the preconditioner generation phase and iterative solution phase suggesting that both the time taken for each of the phases are fairly close.

b. Comparison Across Configuration Groups

Figure 47 shows the average time efficiency for all the solved problems while using the various parallel preconditioners. ILUPACK MLICT and WSMP ICT results are not present since they do not have distributed memory implementations yet. The efficiency plots show a completely different scenario from that seen in the problem specific time profile plots

comparing these preconditioners. PETSc BlockSolve, Trilinos IC(k), and ML preconditioners exhibit very little drop in efficiency for the problems that it could solve. Hypre ParaSails and Hypre IC(k) show similar drop in efficiencies. WSMP Direct shows the maximum drop in efficiency. This drastic drop could also be attributed to its superior time performance in the serial case in comparison to other preconditioners.

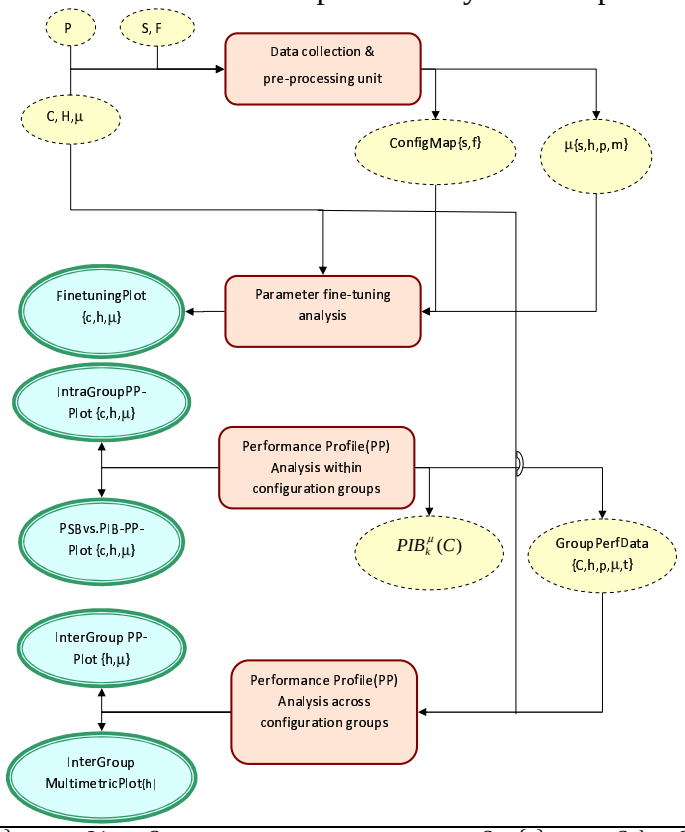
E. Performance Analysis Infrastructure

In this section, we describe the software infrastructure that we used for the semi-automated collection, analysis, and visualization of performance data for the iterative solvers. Most results presented in Section D were generated using this framework. Figure 48 shows the various components of the framework, which is composed of a performance data collection unit and multiple analysis and visualization units. Although the framework is implemented for studying preconditioned iterative solvers, it is readily extensible to other domains where it may be useful to perform a comparative evaluation of several configuration groups with a large number of configurations with respect to their performance on multiple metrics. We now describe each component in more detail.

1. Data Collection and Preprocessing Unit

This consists of serial and parallel driver programs for the solver packages of interest. The input to this component includes the set of linear systems $\mathcal{P}$, a set of hardware configurations (i.e., number of processors) $\mathcal{H}$, the set of supported solver configurations $\mathcal{S}$ with the details of the solvers, preconditioners, and related options and parameters, and a set of performance metrics μ. In addition, the user also specifies a grouping (C) of the solver configurations which can be a combination of any of the solver configuration components such as package, preconditioner, solver, ordering, etc.

Fig. 48. Overview of the performance analysis infrastructure. Boxes represent the processing units, dotted ellipses represent the input and output data while the plots generated for visualization are represented by solid ellipses.



P = {p}	:	Linear Systems	S = {s}	:	Solver Configurations
H = {h}	:	Hardware Configurations	F = {f}	:	Fine-tunable Factors
T = {t}	:	Type (PIB, PSB, PIB vs. PSB)	C	:	Solver Configuration Group
μ	:	Performance Metric (time, memory, MTP)	PP	:	Performance Profile

Other choices of interest include driver specific information such as the right hand side (RHS), initial solution, exact solution and, stopping criterion (e.g., the relative residual norm or maximum number of iterations of the solver), each of which has a default value and can also be modified by the user. The system performs empirical trials for each possible setting, collects the specified metrics, and pre-processes them appropriately to generate the performance data (denoted by μ). It also generates a mapping (*ConfigMap*) from the solver configurations S to various key attributes such as the package-name, preconditioner-name, solver and the associated parameters. This parameter mapping is required to partition the configurations into sub-groups that differ along a single parameter. For example, the configuration map for PETSc-IC(k) would include the ordering scheme, level of fill and fill factors as parameters and a sample *ConfigMap* entry for this case will have an integer value that corresponds to each parameter. In the case of ordering, there are two possible values assigned (RCM(1) and ND(2)). Similarly, we assign integer identifiers to other parameters and a single solver configuration is mapped to an integer vector of parameter identifiers.

2. Parameter Fine-tuning Analysis Unit

This unit computes the variability in the performance due to a single parameter while keeping all others fixed. This analysis is especially relevant for solver configurations within each group (Section D.3). It requires as input the performance data μ as well as the solver configuration to parameter map *ConfigMap* generated by the data collection and pre-processing unit. In order to study the effect of a single parameter, for example ordering scheme, we first find groups of parameter vectors that vary only in the ordering scheme value. For example in the case of ILUPACK MLICT, there are 12 groups corresponding to each combination of drop tolerance and inverse norm estimate value and the solver configurations in each group corresponds to 5 different ordering schemes. For each such group, we calculate the average standard deviation of the percentage change in normalized memory and time

performance for the *solved* problems. Each of these average standard deviations are further averaged across the 12 groups to create the normalized plots.

3. Intra-group Analysis Unit

The intra-group analysis unit computes good default solver configurations (*PIB*$_h^\mu(\mathcal{C})$) that result in the maximum AUC among all the solver configurations within each user specified group $\mathcal{C}$. Such groups that typically represent a package preconditioner combination. It also computes the problem specific best configuration (*PSB*$_h^\mu(\mathcal{C})$) among all configurations in group $\mathcal{C}$ for each test case in the problem set $\mathcal{P}$. Although, we use MTP as the metric (μ) of choice, a user can specify other metrics such as memory, time, or a weighted product of memory and time. For a given hardware configuration and performance metric, a series of performance profile (PP) plots are created for each configuration for analysis at a fine-grained resolution. In addition, the effect of fine-tuning is captured by comparing the performance of the best default configuration and the problem specific best performance. The performance results or the μ values corresponding to both *PIB*$_h^\mu(\mathcal{C})$, *PSB*$_h^\mu(\mathcal{C})$ are compiled for each group $\mathcal{C}$, problem p, and hardware configuration h to generate group performance data (*GroupPerfData*) for further analysis.

4. Inter-group Analysis Unit

The inter-group analysis unit provides a coarse grain comparison of the different solver configuration groups based on group performance data *GroupPerfData* corresponding to the best default configuration (*PIB*$_h^\mu(\mathcal{C})$) as well as the problem specific best (*PSB*$_h^\mu(\mathcal{C})$). Performance profile plots are generated for each hardware configuration for both *PIB*$_h^\mu$ and *PSB*$_h^\mu$ values. In addition, we also generate a multi-metric plot that simultaneously captures the trends for up to three metrics (e.g., memory, time and robustness) in the case of both *PSB* and *PIB* performance. This provides a snapshot of the relative strengths and

weaknesses of the various solver configuration groups with respect to the performance metrics under consideration.

F. Discussion

We performed an extensive empirical evaluation of some commonly used preconditioned iterative methods available in free black box solver packages on a collection of matrices drawn from a wide range of scientific applications. For each package and preconditioner combination, we identify the best parameter choices using a novel performance profile based criterion that takes into consideration the number of problems solved along with the time and memory usage across all the problems in the collection. Our experiments reveal parameter configurations that are good candidates for default configurations. For each preconditioner, we quantify the benefits of parameter fine-tuning by comparing the best performance for each problem with the performance of our experimentally determined default parameters. Different preconditioners show varying levels of tunability and optimizing individual parameters impacts the performance to different degrees. We provide a comparison of the performance of various iterative solver configurations relative to the direct solver, which illustrates the successes and challenges in developing preconditioners for iterative solvers. The results also provide insight into the relative strengths and weaknesses of the various black box preconditioned iterative solver packages. We observed that different implementations of the same preconditioning method can vary widely in performance.

This study, admittedly, has its limitations. The results used for analysis are derived from a test suite of only 30 problems. Although our test suite includes problems from multiple applications, we kept its size modest due to the sheer number of trials (2156) for each matrix. Therefore, the results may not be generalizable to specific application domains. However, the performance collection and reporting infrastructure we have developed is in-

dependent of the test suite and can be used on any set of test matrices from any domain. As part of continuing work on this topic, we plan to set up an anonymous ftp site so that application scientists can provide us with specific matrices in their domain and matrices and obtain a report on the relative performance of each solver configuration group for those matrices. The methodology described in this paper can be helpful to researchers in evaluating different aspects of new solver techniques in a systematic fashion.

CHAPTER IV

SOLVER RECOMMENDATION SYSTEM

In this chapter, we present a novel multi-stage learning-based methodology for determining the "best" solver configuration(s) given the user constraints and the desired performance behavior for any given linear system. Unlike Chapter III which deals with providing coarse guidance to practitioners in terms of the best default configuration and influential parameters for a solver configuration group independent of the linear system, the current chapter specifically focuses on using properties of the linear systems to determine the suitable solver configuration(s). Our solver recommendation methodology relies on a modular formulation consisting of three key sub-problems: (a) solvability modeling, (b) performance modeling, and (c) performance optimization. This decomposition allow us to readily address practical issues arising from solver failure and multi-objective optimization in an efficient and effective manner. Specifically, the solvability model is used to filter out failure-prone configurations before modeling the performance statistics. Further, to accommodate optimization of multiple criteria, we separately learn models for each of the core performance statistics (e.g, time/memory/error). The optimization step involves combining the learned performance models to identify the top solver choices for the specified performance criteria.

We begin by motivating the need for a problem-specific solver recommendation system in Section A and discuss the desiderata for such a system in Section B. In Section C, we describe how the performance data is represented and provide a formal definition of the problem. Section D presents details of our multi-stage learning approach followed by a description of a prototype system in Section E. In Section F, we discuss the strengths and limitations of the proposed methodology relative to existing techniques for solver selection.

A. Motivation

As we discussed earlier in Chapter III, Section D.2, empirical evidence indicates that one can often obtain a significant improvement in performance relative to even a carefully chosen default solver configuration by performing problem-specific fine-tuning of preconditioner and solver parameters. This performance improvement is especially critical for large matrices since it corresponds to a substantial reduction of computational effort.

Knowledge of the influential preconditioner parameters (Chapter III, Section D.3) partially alleviates the solver selection problem by reducing the search space. Even so, choosing the best preprocessing options, preconditioner and fine-tuning the preconditioner's parameters for a particular linear system is a challenging task, even for experts in computational linear algebra due to several reasons. As mentioned earlier, the diversity of the preconditioners and the variability in the different implementations for the same preconditioner heavily limits the utility of any theoretical analysis. The search space determined by the influential preconditioner parameters is often fairly large since most of these parameters tend to be continuous-valued and there also exist strong mutual dependencies requiring exploration of the joint space. The enormous computational resources required for solving large linear systems make it extremely expensive for practitioners to adopt a simple trial and error strategy over the numerous choices of solver configuration components. To make matters worse, many applications require the solution of a series of systems with the coefficient matrices changing gradually and the set of parameters that are best for the first system may not be suitable for the later ones.

Therefore, it is desirable to have a more intelligent strategy for exploring the solver configuration space by making efficient use of readily observable characteristics of the linear systems such as the number of nonzeros, Frobenius norm, etc. A naive approach would be to identify key groups of linear systems based on their numerical characteris-

tics and obtain the best solver configuration for each group using the methodology described in Chapter III. Unfortunately, the linear system properties tend to span a large multi-dimensional continuous space and performance data is available only for a small number of matrices, thus severely limiting the generalization power of discrete recommendations over each problem group. Hence, we consider an alternate approach for learning a statistical model that can predict the best choice(s) for a linear system from a set of solver configurations. Note that the best choice is not intrinsic to the linear system and the solver configuration, but dependent on the entire available set of choices, thus requiring a more complex filtering mechanism as we discuss in the following section.

B. Desiderata for a Solver Recommendation System

From a practitioner's perspective, the process of choosing an iterative solver is usually not a straightforward statistical modeling and optimization problem, but rather an interactive decision making task where the recommendations are supported by evidence. In this section, we now describe the key desiderata for a solver recommendation system, some of which clearly distinguish it from a typical product recommendation system.

1. Prediction of Solver Failure

Empirical studies [18, 32, 34] indicate that iterative solvers have a high rate of failure. The performance metrics obtained for failed trials can often be misleading, for example, a solver could result in extremely low memory usage for a particular linear system, but not converge to the desired accurate solution. Therefore, it is essential to predict and filter out the infeasible trials to ensure reasonable recommendations.

2. Interpretability via Performance Estimates

Often, it is desirable to not only have high quality solver recommendations, but also provide relevant evidence in the form of performance estimates of a solver-configuration for a particular linear system. Such estimates are also essential in order to handle performance-based constraints, e.g., upper limits on memory usage or time taken. The performance predictions can even be directly used to optimize work flow of scientific computation programs in parallel environments.

3. Robustness to Variability in Performance Metrics

Due to variations in the structure and the size of the matrices as well as the nature of the solver configurations, the performance metrics often have a highly skewed distribution with an extremely large range spanning multiple orders of magnitude (e.g., run time often varies from a fraction of milliseconds to several hours). In contrast, in most product recommendation systems, the user preference ratings lie in a small fixed range, e.g., 1–10, and follow a fairly well-behaved distribution. An accurate estimation of performance metrics, therefore, requires a sophisticated modeling approach with suitable distributional transformations and normalization.

4. Optimization of Multiple Hybrid Performance Criteria

In general, for a given linear system, there is no single solver configuration that performs best with respect to time, memory and accuracy. Often, solver configurations that are really fast consume significantly more memory while those with lower memory usage take considerably longer . A practitioner in such a case might prefer a solver that performs reasonably with respect to the memory-time product or some other hybrid additive and/or multiplicative combinations. A methodology that can allow a wide range of such hybrid

criteria without having to model each from scratch would be quite beneficial.

5. Fast and Memory Efficient Online Recommendations

Unlike typical product recommendation systems where the accuracy of the top user preference ratings is paramount, the key purpose of a solver recommendation system is to reduce the computational effort involved in solving large linear systems. Hence, it is highly imperative that the solver selection process itself is highly efficient and the overall objective is to optimize the combined computational effort in identifying the appropriate solver configuration and deploying it for a given linear system.

6. Cold Start Solution for Unseen Matrices

Another important distinguishing characteristic of solver recommendation systems relative to typical recommendation systems is that once a linear system is solved correctly using a particular solver configuration, there often is no need for an additional recommendation. Hence, the main goal of the solver selection problem is to provide recommendations for new unseen matrices, often referred to as the “cold start” scenario, which is not addressed by most collaborative filtering based recommendation techniques, making it essential to rely on the properties of the linear systems and the solver configurations.

In addition to the above requirements on the operation and the output of the recommendation system, one also needs to consider that the main benefit of a solver recommendation system is for solving large linear systems where one cannot afford to waste computational resources with a substantially, suboptimal solver or multiple attempts, and for which unfortunately we are likely to have highly sparse training data. Hence, it is also extremely essential to have an intelligent mechanism for collecting performance data as well as a learning methodology that can generalize from sparse data.

C. Problem Formulation

The desiderata described above clearly point to the need to be able to (i) predict the failure of a solver with respect to a linear system, (ii) estimate performance metrics for any possible trial, as well as (iii) obtain top-k choices for hybrid combinations of the performance metrics. To address these needs, we consider a modular problem formulation that effectively targets each of these three tasks. Before presenting a concrete formulation of the problem, we first describe the representation of the performance data.

1. Data Representation

a. Linear System Features

The performance of a solver configuration with respect to a linear system is highly dependent on the choice of the various solver parameters and their interaction with the numerical and structural properties of the coefficient matrix A. In order to model these interactions, we represent each linear system as a vector of certain key features or attributes $\mathbf{x}(A)$ derived from the matrix A. A key characteristic of all these features is that they are inexpensive to compute. Since the very purpose of employing iterative solvers is to reduce the time and memory consumption, we would like to avoid computing expensive features such as condition number, eigenvalue spectrum, etc. Therefore, choosing simple features is essential for providing real time recommendations in an online scenario. Details of the linear system features used for our experimental studies are provided in Chapter V.

b. Solver Configurations

An iterative solver configuration comprises of many elements such as the choice of solver, matrix preprocessing steps, preconditioner and various numerical/categorical parameters specific to the preconditioner and solver choice. We represent each solver configuration

M as a vector of attributes $\mathbf{y}(M)$ corresponding to the various solver components. To accommodate parameters that are meaningful only for some preconditioners, we allow the parameter attributes to also take a value "not-applicable". Details of the solver features used in various experiments are provided in Chapter V.

c. Empirical Trials and Performance Metrics

To enable problem-specific solver selection, we encode the performance results at the granularity of an empirical trial, i.e., a combination of solver configuration and linear system features. Specifically, the performance results of a trial (A, M) are represented as a vector of performance attributes $\mathbf{z}(A, M)$, which include criteria that are of importance to a user, e.g., computation time, memory usage and accuracy. In general, these observed performance metrics not only depend on the linear system and the solver configuration, but also on the hardware configuration that was used for the empirical trial. To keep the exposition simple, our current work assumes that the performance results are based on a single specific hardware configuration

2. Formal Problem Definition

Let $S_A = \{A_i\}_{i=1}^m$ denote the set of linear systems, $S_M = \{M_j\}_{j=1}^n$ denote the set of solver configurations, and $\mathcal{T} \subset S_A \times S_M$ denote the set of empirical trials for which performance data is available. Let $\mathbf{x}_i = \mathbf{x}(A_i)$ and $\mathbf{y}_j = \mathbf{y}(M_j)$ denote the attribute vectors associated with the i^{th} linear system and j^{th} solver configuration. Let $\mathbf{z}_{ij} = \mathbf{z}(A_i, M_j)$ denote the performance vector associated with the trial (A_i, M_j) so that the empirical performance data, can be represented as a set of 3-tuples $\{(\mathbf{x}_i, \mathbf{y}_j, \mathbf{z}_{ij}) \mid (A_i, M_j) \in \mathcal{T}\}$. We now formally define the key sub-problems in our formulation.

a. Solvability Prediction

Since iterative solvers are known to have a high rate of failure [18], it is essential to identify and eliminate the infeasible trials. In order to formalize the notion of solver failure, we define solvability as a pre-specified boolean function over the observed performance metrics (e.g., convergence is achieved within 10 hours with relative error norm less than 0.01). The solvability prediction problem is, therefore, to estimate this boolean property without actually performing the trial. Let $s_{ij} = s(A_i, M_j) = s(\mathbf{z}_{ij})$ denote the solvability of linear system A_i with respect to configuration M_j. Given empirical observations $(s_{ij}, A_i, \mathbf{x}_i, M_j, \mathbf{y}_j), \forall (i,j) \in \mathcal{T}$, the first task is to predict the solvability $s(A, M)$ for any potential trial involving a linear system A and a solver configuration M. Thus, solvability modeling essentially requires learning a binary dyadic response where the dyads correspond to pairs of linear systems and solver configurations.

b. Performance Estimation

As discussed earlier in Section B, a desirable feature of a solver recommendation system is to provide performance estimates of a solver configuration for a particular linear system, which can also be used for approximate estimation and optimization of other hybrid additive and/or multiplicative combinations such as memory-time product. Therefore, the second sub-problem involves predicting the various performance metrics of interest for the trials that are deemed successful. The modeling problem in this case is similar to that of solvability with the only substantial difference being that we need to deal with multiple real-valued performance metrics instead of binary values. Given empirical observations $(\mathbf{z}_{ij}, A_i, \mathbf{x}_i, M_j, \mathbf{y}_j), \forall (i,j) \in \mathcal{T}$, the performance modeling can be formally stated as predicting the performance metrics $\mathbf{z}(A, M)$ for any potential trial involving a linear system A and a solver configuration M.

c. Top-k Solver Configurations

The final task is to identify the top solver choices for a given linear system that optimize certain performance based *quality* criterion while satisfying the solvability criteria. To formalize the notion of the quality of a solver configuration with respect to the linear system, we define it as a function that maps the performance metrics of the corresponding trial to a real-valued score with lower score being preferable.

Let $g(\mathbf{z})$ denote the quality criteria. A special class of criteria of interest are those based on multiplicative combinations of the core performance metrics, i.e., $g(\mathbf{z}) = \prod_r (z^{(r)})^{\alpha_r}$ where $z^{(r)}$ denotes the r^{th} performance metric, α_r indicates the relative importance of $z^{(r)}$. An example of $g(\mathbf{z})$ is memory-time product, where $\alpha_{memory} = 1$, $\alpha_{time} = 1$ and the rest zero.

Given a linear system A_i, the ranking problem reduces to identifying the top-k solver configurations, or in other words, a mapping $h : \{1, \cdots, k\} \mapsto S_M$ such that:

1. Top-k configurations are solvable, i.e., $s_{ij} = true,\ \forall j \in range(h)$

2. Top-k configurations are ordered by their quality and better than the rest, i.e., $g(\mathbf{z}_{ih(l_1)}) \leq g(\mathbf{z}_{ih(l_2)}) \leq g(\mathbf{z}_{ij})$, where $1 \leq l_1 < l_2 \leq k$ and $j \notin range(h)$.

The values estimated from the performance model are used to determine the quality of a solver configuration with respect to a linear system.

D. Multi-stage Learning Approach

In this section, we describe the key algorithmic components of our approach for addressing the three sub-problems described in Section C.2.

1. Solvability Prediction

Since solvability is a boolean-valued function of the empirical trials, it can be readily modeled in terms of binary classification over the trials. A natural choice for trial features includes the attributes of the linear system and solver configuration along with the product interactions [60]. Given these features and the observed solvability values, one can use any standard classification algorithm such as decision trees or support vector machines along with feature selection [40] to learn a solvability model. An alternate collaborative filtering-like approach is to view the solvability prediction problem as a matrix imputation problem where one seeks to predict missing values in the solvability matrix with linear systems as the rows and solver configurations as the columns. This perspective ignores the trial features and focuses exclusively on leveraging the correlations in the solvability matrix via low rank matrix approximation and bi-clustering techniques [5, 46]. Recently, Agarwal et al. [3] proposed an approach based on predictive discrete latent factor models (PDLF) to simultaneously make use of the available features as well as the local structure in the dyadic response using bi-clustering. However, this approach is limited to generalized linear models and does not readily accommodate feature selection, which is critical for our application since the raw trial features (including interaction features) number in thousands.

We adopt a strategy that mimics the key idea in [3] while explicitly taking care of the feature selection requirements. First, we learn a classifier on the training data while performing feature selection over the raw features. The misclassification error resulting from this classifier is clustered using a bi-clustering algorithm appropriate for ternary (false positive, false negatives and true predictions) response values [5] to identify bi-clusters of linear systems and solver configurations. The bi-cluster memberships are then used to augment the earlier selected features and a new classifier is learned. Figure 49 shows the detailed steps.

Input: Solvability values $s_{ij} = s(A_i, M_j) = s(\mathbf{z}_{ij}),\ \forall (A_i, M_j) \in \mathcal{T}$, linear system attributes $\mathbf{x}_i = \mathbf{x}(A_i), \forall A_i \in S_A$, solver configuration attributes $\mathbf{y}_j = \mathbf{y}(M_j), \forall M_j \in S_M$, number of clusters *k,l*

Output: Solvability model $\hat{s}(A, M)$

Method:

Compute raw and interaction trial features

$u_{ij}^{raw} = [\mathbf{x}_i, \mathbf{y}_j]$

$u_{ij}^{inter} = [x_{i1}, \cdots; y_{j1}, \cdots; x_{i1}x_{i2}, \cdots; x_{i1}y_{j1}, \cdots; y_{j1}y_{j2}, \cdots]$.

Perform feature selection

$u_{ij}^{reduced} = FeatureSelection(\{\mathbf{u}_{ij}^{inter}, s_{ij}\})$

Learn initial classifier

$\hat{s}^{initial} \leftarrow ClassificationAlgorithm(\{s_{ij}, \mathbf{u}_{ij}^{reduced}\})$

Compute misclassification error

$e_{ij} \leftarrow \hat{s}_{ij}^{initial} - s_{ij},\ \forall (A_i, M_j) \in \mathcal{T}$

Perform co-clustering

$(\rho, \gamma) \leftarrow BiclusteringAlgorithm(\{e_{ij}\})$, where $\rho : S_A \mapsto \{1, \cdots, k\}$ and $\gamma : S_M \mapsto \{1, \cdots, l\}$ map the linear systems and solver configurations to their respective clusters.

Augment features

$\mathbf{u}_{ij}^{final} = [\mathbf{u}_{ij}^{reduced}, \mathbb{1}(\rho(A_i) = 1 \wedge \gamma(M_j) = 1),$

$\cdots, \mathbb{1}(\rho(A_i) = k \wedge \gamma(M_j) = l)]$,

where $\mathbb{1}(\rho(A_i) = g \wedge \gamma(M_j) = h)$ denotes membership in gh^{th} bi-cluster

Learn final classifier

$\hat{s} \leftarrow ClassificationAlgorithm(\{s_{ij}, \mathbf{u}_{ij}^{final}\})$

return $\hat{s}$

Fig. 49. Solvability Modeling

2. Performance Prediction

While estimating the performance metrics such as time taken and memory used, we need to deal with real-valued variables that have a large variability for different linear systems (for example, A_1 might take 1-5 hrs to be solved while A_2 only needs 1-100 ms). This variability in the response values leads to a large uncertainty in the modeling process. One way to handle this problem is by normalizing the actual metrics by the performance of a specific default solver. However, when the default solver configuration does not solve all the problems, it might result in ill-defined values. In addition, to obtain better sensitivity for lower performance values, it is desirable to log-transform the performance ratios. This

transformation has the additional benefits of making the response more Gaussian-like and simplifying estimation of multiplicative combinations of the core performance metrics, e.g., memory-time product.

To model the performance metrics, we use traditional regression approaches such as multivariate linear regression [59] and support vector regression [77], augmented with bi-clustering based features as in the case of solvability modeling. The hybrid regression approach based on bi-clusters can be viewed as a variant of Gaussian-PDLF algorithm [3] and follows the hybrid solvability modeling approach outlined in Figure 49. Specifically, for each metric, we first learn a linear regression model over the raw trial features. The prediction error for each trial is computed from this model and is subjected to bi-clustering based on Gaussian distribution to yield clusters of linear systems and solver configurations, which are then used to learn a new regression model. In Chapter V Section A, we compare results using a single regression technique with different sets of features (raw, interactions, and interactions along with bi-clusters) and choose the best option among these for further modeling of configuration group specific models (Chapter V, Section B).

3. Top-k Performance Ranking

Given a linear system and specific performance-based quality and solvability criteria, a naive approach for identifying the top-k solver configurations would be to estimate the quality and solvability of each configuration (assuming the possible set of configurations is finite) and sort the solvable ones in terms of the quality criterion. A faster alternative would be to exploit the fact that the estimates for all the configurations are generated from the solvability and performance models. To illustrate the main idea, consider a hypothetical case where there is a performance metric of interest that depends on just two interaction features (modeled as $\exp(\beta_1 x_1 y_1 + \beta_2 x_2 y_2)$). For a given linear system and the performance model, the features x_1, x_2 and parameters β_1, β_2 are fixed and the quality of the solver

configurations depends only on y_1, y_2. When $\beta_1 x_1$ and $\beta_2 x_2$ are both positive, pre-sorting the solver configurations by y_1 and y_2 into two lists would allow fast identification of the top k configurations for the performance metric of interest.

In general, the performance models tend to contain multiple features. However, the same principle holds, i.e., pre-sorting the features can speed up ranking based on a monotone aggregate function of the features. Our choice of linear regression for modeling the performance is specifically suited for such rank aggregation because the response is modeled as monotonic transformation of linear combination of the feature values. Specifically, for each of the core performance metrics $z^{(r)}$, the estimated value is given by $\hat{z}^{(r)} = \exp({\boldsymbol{\beta}_r}^T \mathbf{u})$, where $\mathbf{u}$ denotes trial features and $\boldsymbol{\beta}_r$ denotes the coefficient vector for the r^{th} performance metric. The exponentiation is required because of the log transformation. The quality criteria, which can be expressed as multiplicative combinations of the core performance metrics, also happen to be simple aggregations over the features, i.e., $g(\mathbf{z}_{ij}) = \prod_r (z_{ij}^{(r)})^{\alpha_r} \Rightarrow g(\hat{\mathbf{z}}_{ij}) = \exp\left((\sum_r \alpha_r \boldsymbol{\beta}_r)^T \mathbf{u}_{ij}\right)$. When the trial features $\mathbf{u}_{ij}$ consist only of raw or simple product interactions of attributes of the linear system and solver configuration, for a fixed linear system A_i, we can directly express the quality of a solver in terms of the solver attributes alone, $g(\hat{\mathbf{z}}_{ij}) = \exp(\boldsymbol{\delta}_i^T \mathbf{y}_j)$, where $\mathbf{y}_j$ denotes features that depend on solver configuration and $\boldsymbol{\delta}_i$ depends on attributes of A_i as well as the coefficient vectors $\{\alpha_r, \boldsymbol{\beta}_r\}_r$. By absorbing the sign of the coefficients $\boldsymbol{\delta}_i$ into the features $\mathbf{y}_j$ themselves, the quality criterion can be reduced to a monotone aggregate of the solver features. There are a number of rank aggregation techniques to obtain the top k choices for a monotone aggregate of the features. In our current work, we employ Fagin's threshold algorithm [27], which has been shown to be optimal in the number of accesses and requires a small constant buffer. The main idea is to efficiently explore potential top choices and stop when one is confident that the unexplored items are not going to make it to the top-k. Figure 50 shows the detailed steps.

Input: Linear system A_i, number of recommendations k, performance model coefficients $\{\boldsymbol{\beta}_r\}_r$, quality criterion $g(\mathbf{z}) = \prod_r (z^{(r)})^{\alpha_r}$, solvability model $\hat{s}(A, M)$, solver configuration set S_M, trial attributes $\{\mathbf{u}_{ij} | (A_i, M_j) \in \mathcal{T}\}$.
Output: Top k recommendations $h : \{1, \cdots, k\} \mapsto S_M$ as defined in Section C.2
Method:
Initialize and sort solver configuration features
$\mathbf{y}_j \leftarrow \mathbf{y}(M_j) = [y_1(M_j), \cdots, y_P(M_j)]$, features of solver configuration $M_j \in S_M$, (P denotes # solver dependent features)
$L_p \leftarrow S_M$ sorted by the p^{th} solver feature $(1 \le p \le P)$
Compute feature coefficients and sign for specified linear system
Choose $\boldsymbol{\delta}_i$ s.t. $\boldsymbol{\delta}_i^T \mathbf{y}_j = (\sum_r \alpha_r \boldsymbol{\beta}_r)^T \mathbf{u}_{ij}$
$\mathbf{w}_{ip} \leftarrow sign(\delta_{ip}),\ \delta_{ip} \leftarrow |\delta_{ip}|,\ \ (1 \le p \le P)$
Initialize candidate solver configuration set
$C_M \leftarrow \emptyset$
repeat
Access top element in the sorted feature lists in the appropriate direction
$M^{(p)} \leftarrow pop(L_p, w_{ip})\ \ (1 \le p \le P)$
Check for solvability
$C_M \leftarrow C_M \bigcup \{M^{(p)}\}$ if $(\hat{s}(A_i, M^{(p)})) = true$
Compute threshold
$\tau \leftarrow \sum_{p=1} w_{ip} \boldsymbol{\delta}_{ip} y_p(M^{(p)})$
until C_M has k objects with $g(A_i, :) \le \tau$
return top k list of C_M in terms of $g(A_i, :)$

Fig. 50. Top-k Performance Ranking

E. Prototype Recommendation System

In this section, we describe a prototype solver recommender system based on the multi-step approach described in Section D. We have implemented this system in C using a combination of external software packages.

Figure 51 shows the various components of the proposed system and their interactions. The functional units are represented as boxes while data is represented as ellipses. At a coarse level, the proposed system has two main components — (a) an offline unit dedicated to empirical data collection and learning solvability/performance models, and (b) an online interactive unit that generates solver recommendations and answers user queries. Each

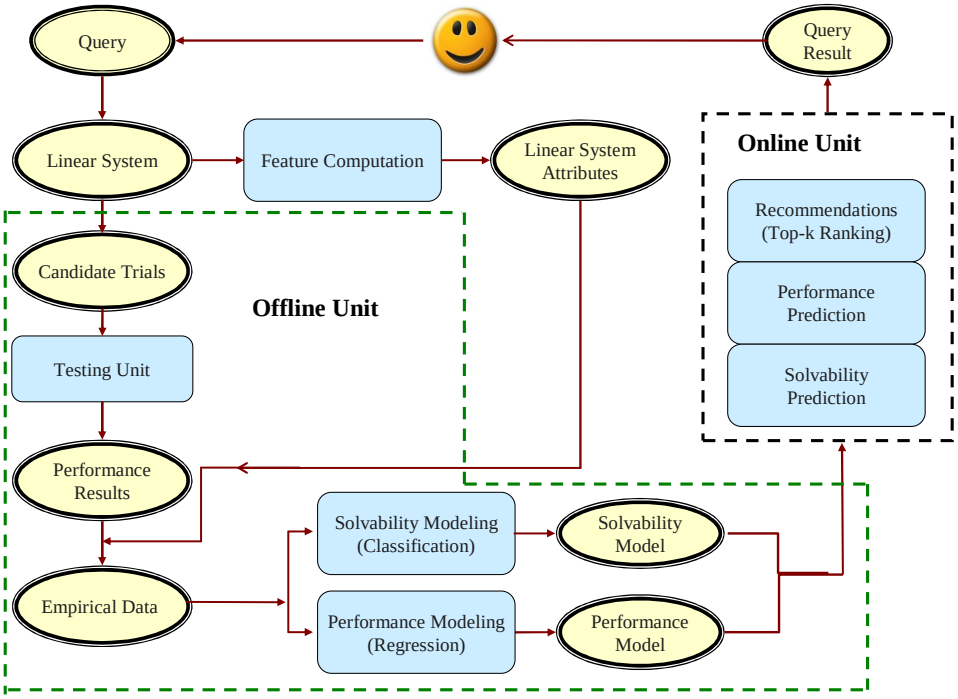


Fig. 51. Prototype Solver Recommender System

of these have multiple sub-components for performing certain focused tasks, which are described below.

- *Empirical Testing Unit* executes the chosen trials under controlled settings and records the performance results in a database. Currently, this functionality is implemented via a driver script on a an IBM HPC cluster 1600 based on 1.9 GHz Power5+ processors.

- *Feature Computation Unit* computes a specified set of attributes for a given linear system. The mapping between the linear system and the derived features is then recorded in the empirical results database.

- *Solvability Modeling Unit* selects informative features and learns binary classifier(s) to predict the solvability of a linear system with respect to a solver configuration. This is accomplished in the current system by extending the SPIDER toolkit [78] and

LIBSVM software, which provide support for multiple classification algorithms such as decision trees and support vector machines as well as multiple feature selection techniques based on information gain, L1 norm, etc.

- *Performance Modeling Unit* learns predictive models for the performance metrics of interest in the solvable region, using the available empirical performance data. This typically involves a combination of multi-variate regression and feature selection techniques and is currently accomplished using routines in the SPIDER kit and LIBSVM.

- *Solvability Prediction Unit* predicts whether a given linear system can be solved using a particular solver configuration using the learned solvability model.

- *Performance Prediction Unit* provides predictions on the expected performance for user specified combinations of linear systems and solver configurations using the learned performance model.

- *Recommendation Unit* provides a top-k ranked list of the solver configurations for a user specified linear system and quality criterion taking into account the specified constraints using a judicious application of the Fagin's threshold algorithm.

F. Discussion

In this section, we contrast our approach with the existing state-of-art machine learning based approaches for selection of sparse iterative solvers.

As discussed in Chapter II, in recent years, there has been a lot of interest in applying machine learning techniques for choosing scientific software. The problem of solver selection for sparse linear systems, in particular, has been the focus of three recent papers, which are all based on classification algorithms. Of these works, the first one by Dongarra

et al. [13] poses the problem of finding the "best" (fastest as well as correct) solver for a given linear system among a small set of possible choices as a multi-class classification problem. The linear systems are represented as multi-dimensional vectors and classified using a Bayesian classifier where the individual class-conditional densities are modeled as multi-variate Gaussian distributions. An important limitation of this method is that the notion of "best" requires a comparison among the various solvers, which necessitates a multi-class formulation that is impractical for a large number of solvers, unlike a simple binary-formulation based on intrinsic solvability criteria (e.g., time $\leq$ 4 hours and relative error $\leq 10^{-4}$). Hence, this approach can only give coarse (e.g., solver configuration group) recommendations. The assumption of Gaussian class-conditional distributions also often does not hold as discussed in [10].

The second work by Xu et al. [85, 86] considers only four different solver configurations (ILU0 and ILU(k) with $k = 1, 2, 3$) and focuses on predicting the intrinsic solvability status (e.g, solved, no convergence, out of memory) for a given matrix and *a single solver configuration* (one of the chosen three) using a support vector machine classifier. Since a classifier needs to be learned for each solver configuration, this approach again does not scale with the number of solvers, but the extended notion of solvability status instead of binary (solved/not solved) has considerable diagnostic value. This approach, however, does not attempt to provide a recommendation among possible solver choices and merely predicts the solvability status. The third work by Bhowmick et al. [10] poses the solver selection problem in terms of identifying all the solver configurations that improve the computational time relative to a arbitrarily chosen "default" configuration by a factor γ. This choice of a target response does not require an explicit comparison among all the solver configurations and is not entirely intrinsic to the current solver configuration due to the dependence on the "default" choice. This idea can be viewed as a compromise between the approaches in [13] and [86] that avoids the scalability issues in [13] while providing

good recommendations when the default is chosen carefully. In this approach, the instances space consists of solver trials, i.e., pairs of linear systems and solver configurations (instead of linear systems as in the previous two approaches) and the problem of determining if the solver configuration in the trial is a "good recommendation" for the matrix reduces to a binary classification, which is solved using alternating decision trees. In the presence of sufficient training data (which was possible since the linear systems used in experiments were small and from a similar domain), this approach was shown to provide fairly high classification accuracy, but the simplistic representation of solver configurations as discrete entities makes it impractical to learn in case of sparse training data.

Our current work shares commonalities with the above three approaches in the use of classification algorithms for solvability prediction and the representation of linear systems as vectors of linear system features. However, there are a few key differences that make our approach more scalable and flexible. First, our representation of solver configurations in terms of features based on various components instead of unrelated discrete entities allows us to readily scale to a large number of configurations and perform fine-grained tuning even over continuous parameters, which is not possible using any the approaches in [10, 13, 85]. The use of solver configuration and linear system interaction properties (e.g., num non-zeros $> 10^5$ and drop tolerance $< 10^{-3}$) in the representation of solver trials is another unique feature of our approach that leads to better predictions. Second, unlike the previous approaches that provide coarse binary (or categorical in case of [13]) recommendations based on a single criterion, we also attempt to estimate the actual performance values and provide a ranking of the solver configurations based on a desired criterion. This modular approach allows us to separately target the basic solvability requirements and the performance optimization in addition to providing fairly interpretable results. The separate modeling of the different performance metrics provides the flexibility to optimize multiple hybrid criteria as well as incorporating resource constraints (e.g., memory usage $<$ 2GB).

It also enables us to estimate the suitability of any subset of solver configurations without having to construct a new model from scratch. Lastly, our choice of data representation and multi-step methodology provides a higher capacity to generalize from sparse training data (which is common for large sized matrices and a large number of solver configurations) compared to the existing approaches. In fact, the large number of solver configurations in our performance dataset (see Chapter V) make it computationally impractical to perform a quantitative comparison of even the data representational aspects of the existing approaches [1].

[1] We did not perform a simplified quantitative comparison of the different solver selection approaches over coarse solver configuration groups since that does not address the key problem of interest, namely, linear system-specific fine-tuning of solver configuration parameters.

CHAPTER V

EMPIRICAL EVALUATION OF RECOMMENDATION SYSTEM

In this chapter, we present empirical evaluation of the various aspects of our solver recommendation approach. In particular we focus on two studies using the multi-step approach outlined in Chapter IV. The first study (Section A) focuses on a single package, Hypre, and explores the use of multiple learning algorithms and different feature sets. The heterogeneity of the feature space as well as varied performance behavior across the various solver configuration groups motivated our second study, which involves learning a separate model for each solver configuration group. In Section B, we describe the results from solver configuration specific modeling for Trilinos ML, Hypre ParaSails, and WSMP ICT.

A. Package Specific Modeling

We describe the results of a prototype recommendation system based on package specific models. We have chosen the Hypre package for this purpose since it had a wide range of preconditioner encompassing level based and threshold based incomplete factorization, approximate inverse, and multigrid based preconditioners. First, we present the details on the performance dataset including the linear system and solver configuration features. We then show the results on solvability and performance modeling on new trials involving the matrices in the training set. Finally, we present the precision and quality of the top-k recommendations.

1. Performance Dataset

For this study, we used the 30 SPD test matrices in our empirical study in Chapter III and 317 solver configurations from the Hypre package. Table XIV lists the set of features of the

Table XIV. Linear system features along with the p-values for the Pearson correlation coefficient with respect to memory, time and solvability values on randomly selected 20% training data.

Linear System Features	Memory	Time	Solvability
Geometric Dimension (GD)	2.6-06	3.8-07	1.5-09
Number of rows/columns	1.3e-05	9.8-05	2.8e-11
Number of Non-zeros	8.9e-08	1.2e-07	5.1e-13
Avg. non-zeros per col. (AvgNzPerCol)	0.8	0.5	7.6e-03
Std. dev. of AvgNzPerCol(stdAvgNzPerCol)	0.75	0.6	1.7e-08
Weight of longest column	0.02	0.13	2.7e-06
Weight of shortest column	2.1-04	0.2	0.16
%Weakly diagonally dominant columns	1e-10	8.9e-13	6.4e-28
Maximum bandwidth	1.3e-05	1e-04	1.3e-10
Average diagonal dominance (avgDiagDom)	1.2e-09	5.0e-08	4.1e-12
Frobenius Norm	6e-12	7.8e-08	0.93
Max. over min. of row sum (mm-RS)	0.52	4.1e-03	7.4e-08
Std. dev. of row sum (stdRS)	0.76	4e-05	0.29

matrices that we consider, along with the p-values of their Pearson correlation coefficients to three key performance metrics. A key characteristic of all these features is that they are inexpensive to compute. Since the very purpose of using iterative solvers is to use moderate time and memory resources, we would like to avoid computing expensive features such as condition number, eigenvalue spectrum, etc. Therefore, choosing simple features is essential for providing real time recommendations in an online scenario. The low p-values of the Pearson correlation coefficients of most features in Table XIV suggest that these features have a correlation with the performance metrics that is significantly different from zero. Table XV contains a list of the solver features that we used in our experiments. Based on the description of the solver options in Table XV, we identified 15 attributes such as solver, restart, preconditioner, level of fill and drop tolerance. Using the above solver configurations, we generated performance data on the set of matrices in our collection. For this set of experiments, the right hand size was chosen corresponding to an exact solution of all ones. In addition, we set a limit of 1000 for the maximum number of iterations and the relative residual norm stopping criterion as 10^{-8}. For each trial, we obtained the memory usage, time taken, relative error norm, and also recorded solver failure where applicable.

Table XV. Description of the components of solver configuration.

Package	Solver	Preconditioner	Orderings	Preconditioner Parameters
HYPRE	CG	IC(K)	RCM, ND	*Level of fill:* 0, 1, 2 *Fill factor:* 3, 5, 8, 10 or *Max NNZ/row:* 5, ∞
	GMRES Restart(30,65,100)	ILUT	RCM, ND	*Drop tolerance (DT):* 1e-2, 3e-2, 1e-3, 5e-4 *Fill factor:* 3, 5, 8, 10 or *Max NNZ/row:* 5
	CG	ParaSails	RCM, ND, NONE	*Number of levels (Lev):* 0, 1, 2 *Threshold (Thresh):* 0, 0.01, 0.1, -0.75, -0.9 *Filter:* 0, 0.001, 0.05, -0.9
	CG	BoomerAMG	RCM, ND, NONE	*Maximum number of levels:* 25 *Number of aggressive coarsening levels:* 0, 10 *Coarsening schemes:* Falgout, HMIS, PMIS *Strong threshold (ST):* 0.25, 0.5, 0.8. 0.9

2. Solvability Modeling

In our evaluation, a trial was considered to be successful, i.e., the linear system was deemed solvable by a particular configuration, if the final relative error norm was less than 10^{-2} or if the relative residual norm was less than 10^{-8} and the relative norm of the error was in the range [0.01, 0.1]. Furthermore, we enforce a wall time limit of 3 hours and memory limit of 16 GB.

To test the effectiveness of learning based approach for predicting solvability, we split the performance datasets into multiple train splits (of varying size — 20% to 80%) and a test split containing 20% of the trials. For each such split, we considered four different sets of features — (a) raw features formed by concatenating those of the linear system and solver configuration (Raw), (b) raw features along with linear interactions (Interaction), (c) only bi-cluster membership features (BiClust), (d) concatenation of interaction features with complementary bi-cluster membership features (Inter-BiClust). Each of the above feature sets was further refined using mutual information gain based feature selection and in each case, we learned a solvability model using three different classification algorithms: (1) support vector machines (SVM) [81], (2) decision tree (J48) [48], and (3) K-nearest neighbor (KNN) [60]. For each run, we computed (a) classification error, (FN+FP)/(TN+TP+FN+FP), (b) specificity, TN/(TN+FP), and (c) sensitivity, TP/(TP+FN).

Here FN, FP, TN, and TP denote the numbers of false negatives, false positives, true negatives, and true positives, respectively.

Figure 52 shows the classification error, sensitivity, and specificity using different feature sets for the various classifiers on a 20% training data averaged over 5 runs. We find that the SVM and KNN classifiers significantly outperform the decision tree classifier. The raw features seem to be quite predictive of solvability and result in a substantial improvement over the baseline classification error (45.6% using the majority classification). Including the interaction features leads to even better classification accuracy. Figure 53 depicts a 3×3 bi-clustering of the trials. On examining the clusters, it was observed that the third linear system cluster (bottom) consisted mainly of matrices that could be solved by most of the methods. The second cluster consists of linear systems that could not be solved using the IC(k) and ILUT preconditioners as well as many ParaSails and BoomerAMG based solver configurations while the first one contains matrices that could not be solved by the IC(k) and ILUT preconditioners, but were solved by most configurations of ParaSails and BoomerAMG preconditioners. Though the latent bi-clusters discovered in isolation are valuable in the absence of observed trial characteristics, we find that there was no additional benefit in using interaction features along with bi-cluster membership, possibly because our interaction feature set was rich enough to subsume information from the bi-clusters. The first column in Table XVI lists the top 5 interaction features that were selected for classification.

3. Performance Modeling

We used the subset of trials deemed to be solvable to learn regression models for the time taken and memory used. These values were normalized by the corresponding values for specific default configuration(s) that correspond to the "best" choice independent of the linear system. To identify the overall "best" solver configuration, we considered performance

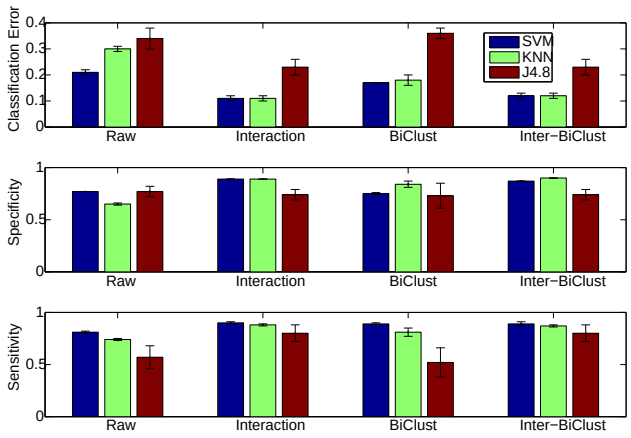


Fig. 52. Classification error, sensitivity and specificity on test set for solvability prediction for SVM, KNN and J48 classifiers on a trial with 20% training split averaged over 5 runs.

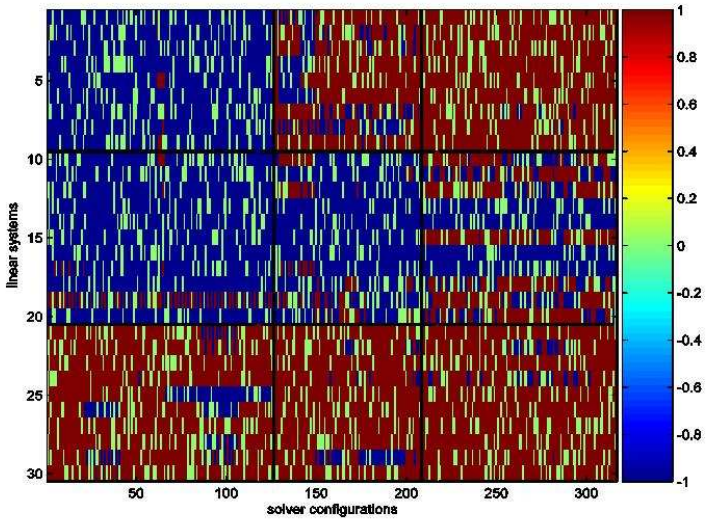


Fig. 53. Linear system-solver configuration bi-cluster for solvability. Blue indicates solver failure, red indicates solver successes and green indicates missing values.

Table XVI. Top 5 interaction features selected for classification, memory and time prediction on a 20% training data. The features with an "is_" prefix are solver features.

Solvability	**Memory**	**Time**
GD × avgNnzPerCol	GD×avgNnzPerCol	GD×is_ST-0.7
GD × is_CG	GD×is_CG	GD×avgNnzPerCol
GD × is_Restart-100	GD×is_Restart100	GD×is_CG
mmRS × is_SAI-Lev0	GD×is_GMRES	GD×is_Restart-100
mmRS × is_ILUT-DT1e-3	stdRS×isSAI-Lev2	GD×is_SAI-Thresh0

profile curves [24] of the different solver configurations, i.e., plots of the cumulative distribution of the performance ratios with respect to a performance metric. The default solver configurations were then chosen so as to optimize both the performance and the number of linear systems solved using the area under the performance profile curves (Chapter III) .

As in the case of solvability modeling, we created train splits of varying sizes (20% to 80%) and a test split containing 20% of the solvable trials. For each such split, we again considered four different sets of features (Raw, Interaction, BiClust, Inter-BiClust) and in each case applied multi-variate linear regression along with feature selection. To study the effects of variability, we modeled the performance values after log transformation. For each run, we computed the R^2 statistic defined as $1 - \frac{\sum_i (\hat{z}_i - z_i)^2}{\sum_i (\hat{z}_i - \bar{z})^2}$, where $\hat{z}$ is the predicted value, z is the actual value and $\bar{z}$ is the mean of the actual values.

Figure 54 shows the R^2 statistic for the predicted memory and time values using different feature sets for different sizes of training data. From the figure, the observed features as well as the bi-clustering memberships are clearly very predictive and provide a significant reduction (61% for memory, 41% for time) in the quadratic loss in the best case. As in the case of solvability, the interaction features proved critical for improving the performance estimates. As expected, increasing the size of the training set results in a steady increase in the prediction accuracy. The second and third columns in Table XVI shows the top 5 predictive interaction features for memory and time, respectively.

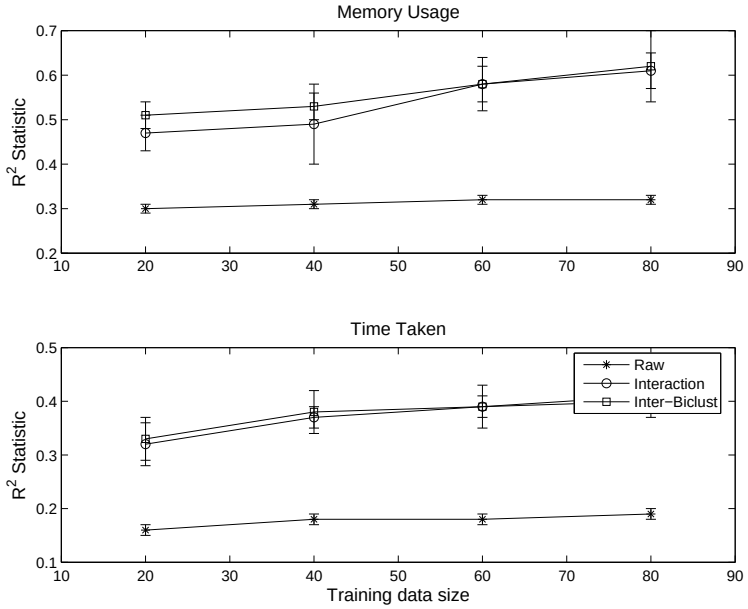


Fig. 54. R^2 statistic for memory and time prediction with varying training data size averaged over 5 runs for multiple feature sets.

4. Top-k Recommendations

We now present results on the top-k recommendations for each of the linear systems given. To highlight the flexibility of our approach, we consider three different criteria for determining the best solvers. The first two involve optimizing core performance values, i.e., memory usage and computational time while the third one focuses on optimizing the memory-time product.

For each linear system, we used the solvability and performance models (with log-transformed response) trained only on 20% of the trials using the best feature set (Inter-BiClust) to identify the top-k (k =25) solver configurations for each criterion. Using the full performance data, the actual top-k solutions were also identified. We measured the quality of the recommendations in terms of two performance metrics: (a) top-k precision, i.e., fraction of the predicted top-k solutions that are in the actual top-k list, (b) improvement over the problem independent best choice (PIB) in terms of average quality value of the

top-k recommendations.

Figure 55 shows the top-k precision of the solver recommendations for optimizing memory, time, and memory-time product. Our approach identifies a large fraction of the top solutions (approximately 52% for memory and 43% for time) in a purely automated manner and requires evaluating the performance models only for a small subset of possible choices. Figure 56 (a-c) shows the performance improvement that can be obtained using the generated recommendations over the PIB choice. In this case, the PIB solver configuration (dotted lines) were chosen based on the overall best performance on the entire test suite using performance profile areas. The problem specific best (PSB) fine-tuning curves shows the average performance value of the actual top-k solutions, which is the best achievable improvement. We observe that the recommender fine-tuning curve is always lower than the PIB choice for all the three criteria and fairly close to the PSB curve. The recommendations for memory-time product indicate that our approach can be quite effective even for optimizing a hybrid performance criterion.

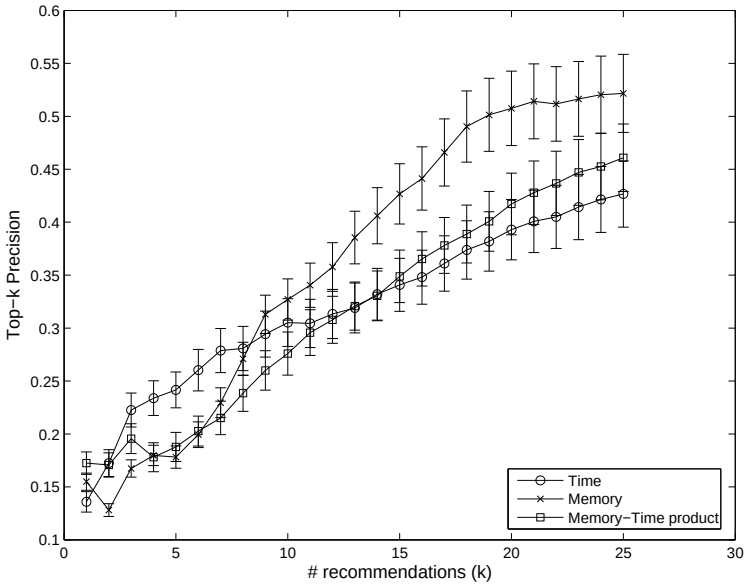


Fig. 55. Fraction of the true best choices for memory, time and memory-time product that is present in top-k recommendations.

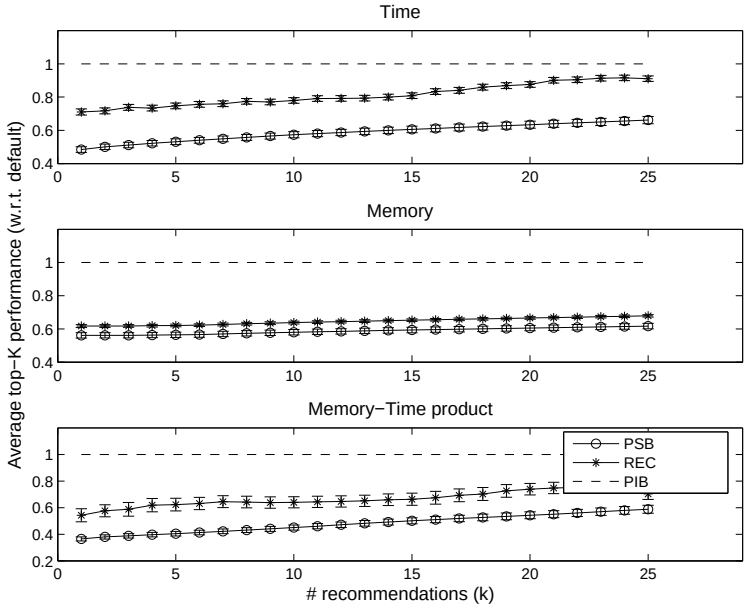


Fig. 56. Average improvement in the memory, time and memory-time product due to fine–tuning over that of the PIB choice for multiple values of k.

5. Discussion

The recommendations using package specific models look promising, however, there are a number of issues that need to be addressed. The performance data used for learning and testing the model involved only those matrices which were solved by the PIB configuration. This restriction removed a number of complex challenging matrices and the resulting performance data. Another issue is that all our testing is performed on trials associated with matrices represented in the training data. In the next section, we will remove the restriction that matrices need to be solved by the PIB choice and examine the quality of the model by testing on an entirely new set of matrices that are not in the training set.

B. Solver Configuration Group Specific Modeling

In this section, we present the results on solver configuration specific models that were created primarily to address the heterogeneity in the solver feature space and inadequacy of the PIB choice in solving certain problems. First, we present details on the performance dataset used for this study. We then present solvability and performance results on a 20% test data of unseen trials on matrices used for learning the model as well as a set of new matrices that did not have any presence in the training set.

1. Performance Dataset

Tables XVII and XVIII shows the SPD matrices that are used for training and testing the models. These train and test matrix sets were chosen in order to have adequate representation of the application domains. Table XIX shows the linear system features that were extracted from these matrices, which similar to those used in Section A with a few minor changes. A detailed analysis of the correlation of the top interaction features will be provided later in Sections 2 and 3. Adequate training data is paramount to learning good models for solvability and performance prediction. Some of the solver configuration groups in this study had very few parameters and due to solver failures, there was very little training data for learning performance models. For those solver configuration groups, the respective PIB configuration is quite competitive and the use of a model is probably not necessary. In our empirical setup there are only 5 solver configuration groups which had at least 50 or more solver configurations. Of these, we chose a representative solver configuration group corresponding to preconditioners based on multigrid, sparse approximate inverse, and theshold-based incomplete Cholesky factorization. These solver configuration groups are shown in Table XX along with the number of solver configurations in each and the number of trials used for learning the solvability and performance models. The

Table XVII. SPD test matrices with their order (N), number of non-zeros (NNZ) and the application area of origin

Matrix	N	NNZ	Application
46053	46053	2863917	Sheet metal forming
af_shell7	504855	17588875	Sheet metal forming
Autodesk-big	1073724	84317460	Static stress analysis
apache2	715176	4817870	3D finite difference structural analysis
BenElechi1	245874	13150496	Structural analysis
bmwcra_1	148770	10644002	Automotive crankshaft modeling
bone010	986703	71666325	Micro finite element analysis of bone
ctu-1	1017397	74144859	Structural analysis
ctu-2	384012	28069776	Structural analysis
cfd1	70656	1828364	C.F.D. pressure matrix
conti20	20341	1854361	Structural analysis
crankseg_1	52804	10614210	Linear static analysis of crankshaft
F1	343791	26837113	Structural analysis
G3_circuit	1585478	7660826	Circuit simulation
hb_drawlins	282576	21718350	Structural mechanics
hood	220542	10768436	Automotive
inline_1	503712	36816342	Structural engineering
kyushu	990692	26268136	Structural engineering
msdoor	415863	20240935	Structural analysis
mstamp-1c	354816	26143920	Metal stamping
nastran-b	1508088	111614436	Structural analysis
nd24k	72000	28715634	3D mesh problems (ND problem set)
oilpan	73752	3597188	Structural analysis
parabolic_fem	525825	3674625	C.F.D. convection-diffusion
pga-rem-1	5978665	29640547	Power network analysis
pga-rem-2	1480825	7223497	Power network analysis
pwtk	217918	11634424	Stiffness matrix - Pressurized wind tunnel
qa8fk	66127	1660579	F.E.M. stiffness matrix for 3D acoustic problem
ship_003	121728	8086034	Structural analysis - ship structure
shipsec5	179860	10113096	Structural analysis - ship section
thermal2	1228045	8580313	Steady state thermal problem
tmt_sym	726713	5080961	Electromagenetic simulation
torso	201142	3161120	Human torso modeling

Table XVIII. SPD test matrices with their order (N), number of non-zeros (NNZ) and the application area of origin

Matrix	N	NNZ	Application
90153	90153	5629095	Sheet metal forming
audikw_1	943695	77651847	Automotive crankshaft modeling
boneS10	914898	55468422	Micro finite element analysis of bone
garybig	42459173	238142243	Circuit simulation
ldoor	952203	46522475	Structural analysis
mstamp-2c	902289	70925391	Metal stamping
pga-rem3	2928711	15510973	Circuit Simulation

Table XIX. Linear system features extracted from the matrices for solvability and performance modeling.

Linear System Features	**Abbreviation**
Number of rows/columns	NUMCOLS
Number of non-zeros	NUMNONZEROS
Aveage non-zeros per column	AVGNZPERCOL
Standard deviation of AVGNZPERCOL	STDAVGNZPERCOL
Maximum bandwidth	MAXBANDWIDTH
Average bandwidth	AVGBANDWIDTH
%Weakly diagonally dominant columns	PERCENTAGEWEAKDIAGDOMROWS
Average diagonal dominance	AVGDIAGDOM
Maximum over minimum of row sum	MAXOVERMINROWSUM
Standard deviation of row sum	STDROWSUM
Geometric Dimension based on maximum bandwidth	GEOMDIMMAXBANDWIDTH
Geometric Dimension based on average bandwidth	GEOMDIMAVGBANDWIDTH
Standard deviation of diagonal	STDDIAG
Size of supernode	SUPNODESIZE

table also lists the maximum number of interaction features that were used for each solver configuration group.

In order to generate the performance data, for each trial composed of a matrix and solver configuration, we set the right hand side to a vector of all ones and solve the resulting linear system. The iterations are stopped if the number of iterations exceed 1000 or if the relative residual norm drops below 10^{-5}. In each trial, we record the memory usage, time taken, final relative residual, and the relative error norm of the final solution. The relative error norm is computed by using the exact solution obtained using the direct solver.

Table XX. Solver configuration groups used for learning individual models along with the number of solver configurations, number of interaction features, total number of trials used for learning solvability and performance models.

Configuration Group	#Configurations	#Interaction features	#Total Trials Solvability	#Feasible Trials Performance
Trilinos ML	66	372	1743	1020
Hypre ParaSails	99	349	2614	798
WSMP ICT	64	288	1690	1512

2. Solvability Modeling

As discussed earlier in Chapter IV, we modeled solvability as a boolean function of trials. A trial is considered to be successful, only if the final relative error norm is less than 2×10^{-2} and the relative residual norm is less than 10^{-5}. In addition, we also enforce a wall time limit of 4 hours and memory limit of 16 GB. In order to avoid over-fitting the models, we experimented with 10 different feature set sizes on five different train-test crossfold splits. Before creating the feature sets, the various features are first ordered based on mutual information criteria [60] and then normalized such that each feature vector has zero mean and unit variance. We then select 10 feature sets of increasing sizes starting from 10% all the way to 100% from the set of meaningful interaction features. We used the SVM classifier with the RBF kernel for learning the solvability model. An important choice for achieving good results using SVM is the value of the kernel parameter γ and the penalty parameter C. As proposed by Lin et al. [16], we perform a parallel grid search over values of $C = 2^{-5}, 2^{-3}, 2^{-1}, 2^{1}, \cdots, 2^{13}, 2^{15}$ and $\gamma = 2^{-15}, 2^{-13}, 2^{-11}, \cdots, 2^{-1}, 2^{1}, 2^{3}$. The results shown in this section correspond to the model learned using the parameter that resulted in the best accuracy over the 110 combinations of C and γ. This parallel grid search is performed for each feature set and crossfold split and the best C and γ are chosen. The results shown in the following sections are the averaged values over the various splits

for the best feature set. We now discuss the results on new trials over the train matrices and also unseen test matrices. We also present a brief analysis of the predictive features of the best solvability models for the different solver configuration groups.

a. Results on New Trials Over Train Matrices

Figure 57 shows the (a) classification accuracy (TP+TN)/(TP+FP+TN+FN), (b) precision TP/(TP+FP), and (c) recall TP/(TP+FN) for a 20% test split of the trials averaged over 5 runs. In this case, we used the remaining 80 % of the data for training the model. Experiments with a smaller training data (50%) also yield comparable accuracy, precision, and recall values. We observe that for all the solver configuration groups the results are significantly superior to the baseline, which corresponds to a constant prediction of the majority class. For precision and recall, we do not show the baseline since the baseline precision is identical to the classification accuracy value and the baseline recall is 1 when the majority class corresponds to the solvable cases as in our data. Here, we do not compare with other classification algorithms and feature selection choices as in Section A since the main goal is to evaluate the overall quality of recommendations.

b. Results on Unseen Test Matrices

The solvability predictions in the previous section are quite good since the test set contains new trials only involving matrices from the training data. The litmus test for these solvability models, however is their ability to generalize on a set of completely new set of matrices, i.e., the models have seen no performance data on any trials involving a particular matrix. Figure 58 shows the classification accuracy, precision, and recall for trials involving all the various solver configurations in a group and the unseen test matrices. Even though the solvability prediction accuracy is better than the baseline predictions, we observe that the prediction quality is not comparable to that for new trials on train matrices in Figure 57.

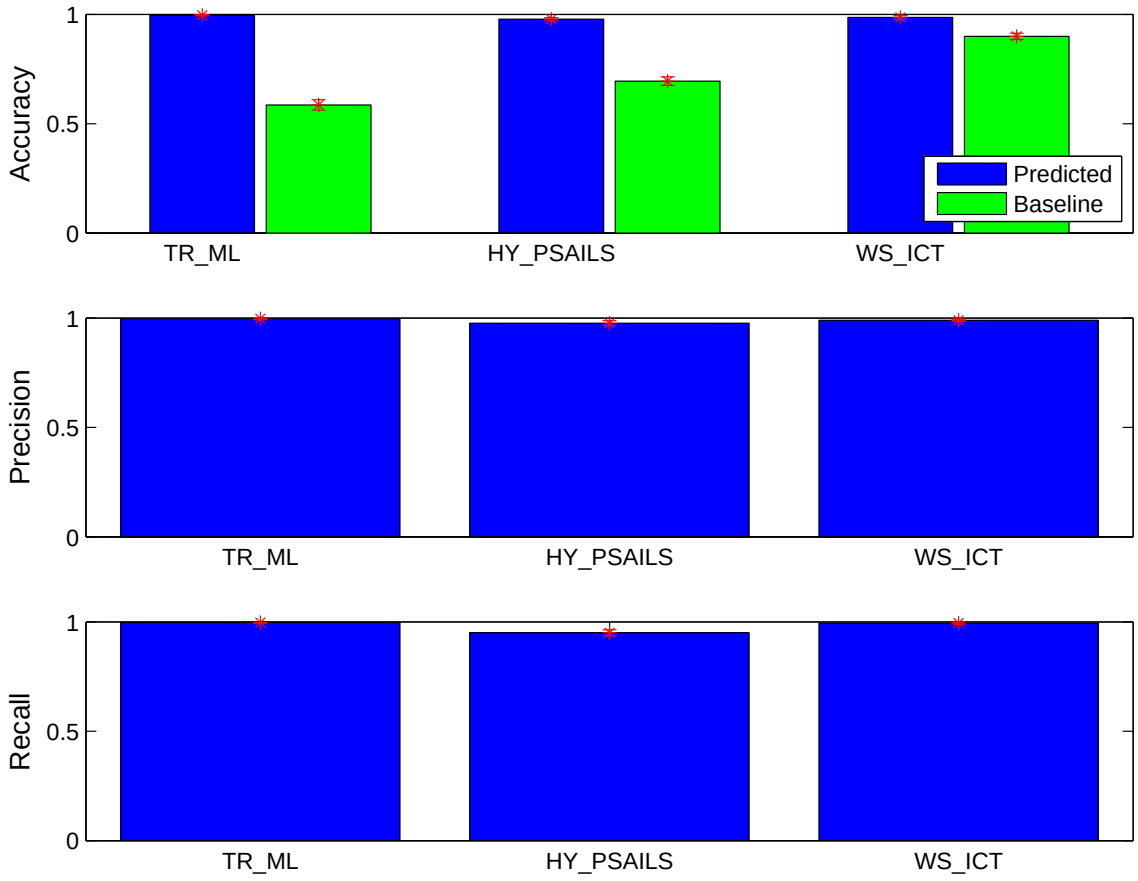


Fig. 57. Classification error, precision, and recall for solvability prediction using SVM classifier on a 20% hold out set of new trials on seen matrices for Trilinos ML, Hypre ParaSails, and WSMP ICT. The values shown are averaged over 5 runs.

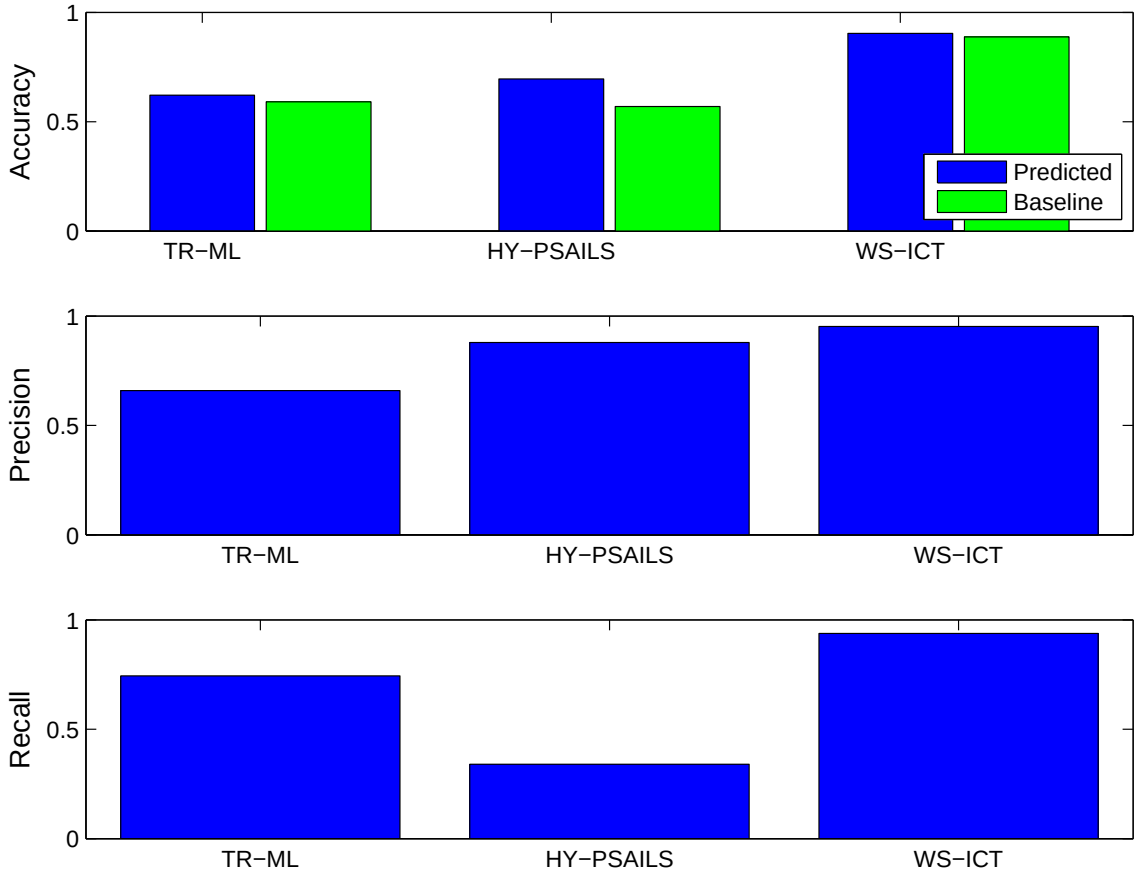


Fig. 58. Classification accuracy, precision, and recall for solvability prediction using the best solvability model on trials comprising of unseen matrices and solver configurations for Trilinos ML, Hypre ParaSails, and WSMP ICT.

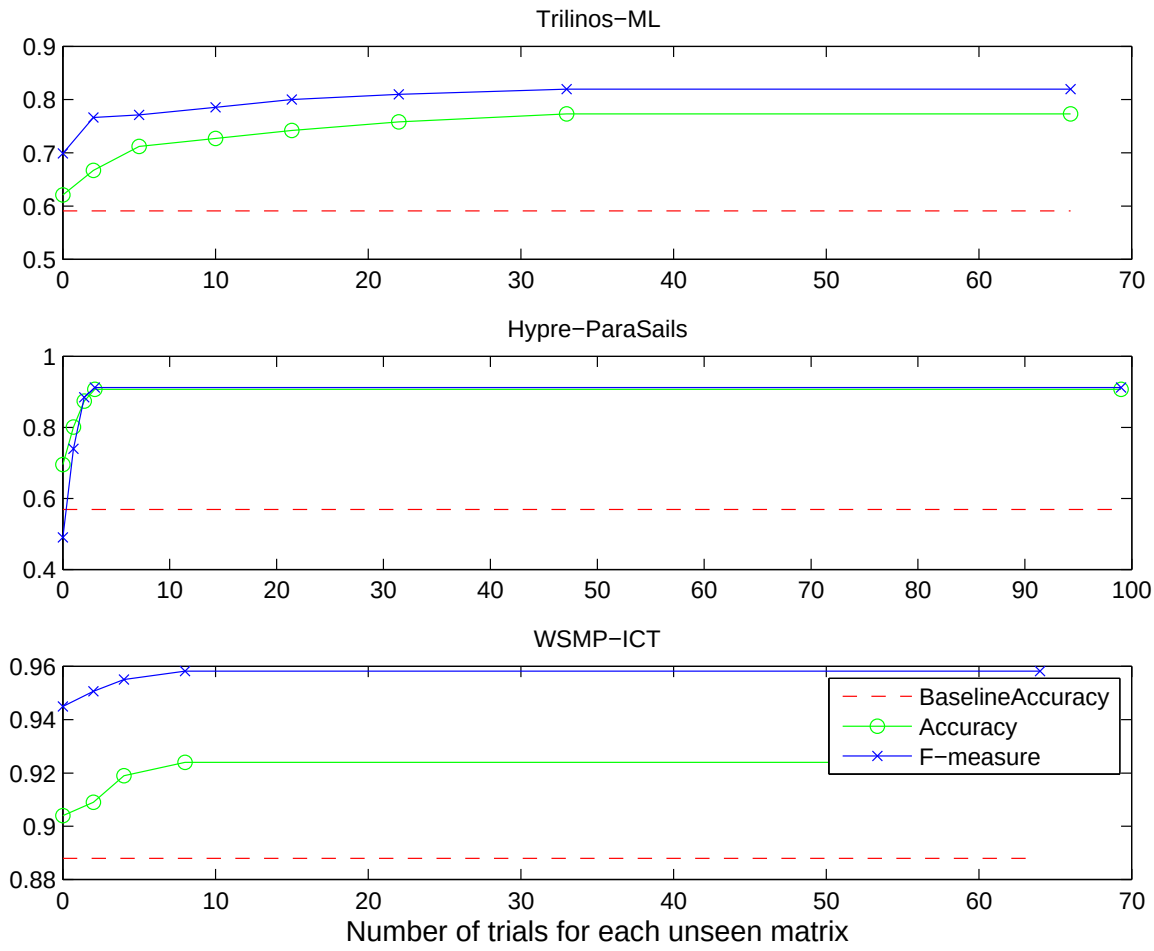


Fig. 59. Improvement in classification accuracy and f-measure for solvability prediction for increasing number of trials on unseen matrices for Trilinos ML, Hypre ParaSails, and WSMP ICT.

This indicates that there is a significant drift in the train and test data distributions and the i.i.d assumptions inherent in our inductive classification approach are not completely true.

To analyze this further, we retrained the classification models using the training data on the seen train matrices and increasing number of trials on unseen test matrices and evaluated them on the rest of the trials on unseen test matrices. Figure 59 shows the learning curves for each of the three solver configuration groups with increasing number of trials on the new matrices on the x-axis and the solvability prediction accuracy and f-measure on the y-axis. For all the three solver configuration groups, there is substantial improvement in the prediction quality with a very small amount of supervision on trials associated with the unseen matrices and after a rapid increase, it seems to flatten out. This behavior can be explained by considering the different classification models. In particular, the classifier trained only on trials associated with train matrices has coefficients for features derived from the linear system characteristics (including interaction features) optimized only for the train matrices. When there is a significant disparity in the distribution of linear system characteristics of the train and test matrices due to the relative sparsity with respect to the linear system feature space (as seems to be the case in our experiments), these models are not likely to do well on the test matrices. However, even a small number of trials on the unseen test matrices can rectify this problem by providing sufficient representation in the relevant linear system feature space leading to a better fine-tuning of the model coefficients for the linear system features and consequently, a big jump in the prediction accuracy, especially when the linear system features are highly informative. The flattening out of prediction quality, on the other hand, can be explained by the fact that there exist only a few solver configuration feature-value combinations that are strongly predictive of solvability, e.g., IS_DT1E-3=1, and their effects can be captured by supervision on a small number of representative solver configurations.

Table XXI. Top 5 interaction features selected for solvability prediction in the case of Trilinos ML. The components of the interaction features are separated with a ":" and the solver features are prefixed with a "IS_".

Feature Name	Correlation	Sign	p-Value
MAXBANDWIDTH:IS_ML-IFPACK	1.461021e-01	1	3.028547e-02
MAXBANDWIDTH:IS_ML-SS1	1.461021e-01	1	3.028547e-02
ISORDERING-ND:IS_ML-IFPACK	1.317742e-01	1	5.094860e-02
ISORDERING-ND:IS_ML-SS1	1.317742e-01	1	5.094860e-02
STDDIAG:IS_ML-IFPACK	1.300553e-01	1	5.407555e-02

Table XXII. Top 5 interaction features selected for solvability prediction in the case of WSMP ICT. The components of the interaction features are separated with a ":" and the solver features are prefixed with a "IS_".

Feature Name	Correlation	Sign	p-Value
STDDIAG:IS_DT1E-2	2.393974e-01	1	3.611661e-03
STDROWSUM:STDDIAG	1.714817e-01	-1	3.849293e-02
ISORDERING-RCM:IS_DT1E-3	1.610345e-01	1	5.216677e-02
AVGNNZPERROW:GEOMDIMAVGBANDWIDTH	1.508481e-01	-1	6.914791e-02
AVGNNZPERROW:GEOMDIMMAXBANDWIDTH	1.502457e-01	-1	7.027865e-02

c. Model Analysis

A salient property of SVM is that the predictions depend only on the support vectors. Therefore, in order to analyze the features, we select the top 15 features based on correlation of the support vectors with the target response. Tables XXI– XXIII show the top interaction features for Trilinos ML, Hypre ParaSails, and WSMP ICT respectively for the solvability models learned on trials from 33 train matrices. The tables also indicate the actual value of the correlation, whether they are positively or negatively correlated, and also the p-values of the correlation coefficients. In the case of Trilinos ML and WSMP ICT, the model seems to have not so high correlation with the top features (as well as high p-values) indicating that the learned model might not be significantly more predictive than the baseline prediction (majority class) on unseen data. However, in the case of Hypre ParaSails in Table XXIII, there are number of features that are highly correlated with the solvability values. For example, it was observed in Chapter III that the use of negative values for threshold and filter parameters result in less robust solver configurations. This is

Table XXIII. Top 20 interaction features selected for solvability prediction in the case of Hypre ParaSails. The components of the interaction features are separated with a ":" and the solver features are prefixed with a "IS_".

Feature Name	Correlation	Sign	p-Value
MAXBANDWIDTH:IS_SAI-FLT-0.9	2.849049e-01	-1	6.238230e-15
AVGBANDWIDTH:IS_SAI-FLT-0.9	2.752059e-01	-1	5.370187e-14
NUMNONZEROS:IS_SAI-FLT-0.9	2.709347e-01	-1	1.349256e-13
NUMCOLS:IS_SAI-FLT-0.9	2.691398e-01	-1	1.977574e-13
AVGNNZPERROW:IS_SAI-FLT-0.9	2.633090e-01	-1	6.714327e-13
IS_SAI-FLT-0.9	2.622759e-01	-1	8.312177e-13
GEOMDIMMAXBANDWIDTH:IS_SAI-FLT-0.9	2.423301e-01	-1	4.276813e-11
SUPNODESIZE:IS_SAI-FLT-0.9	2.403359e-01	-1	6.224908e-11
GEOMDIMAVGBANDWIDTH:IS_SAI-FLT-0.9	2.395155e-01	-1	7.257244e-11
AVGBANDWIDTH:MAXOVERMINROWSUM	2.272560e-01	-1	6.721732e-10
MAXBANDWIDTH:MAXOVERMINROWSUM	2.240591e-01	-1	1.176746e-09
AVGNNZPERROW:MAXOVERMINROWSUM	2.217973e-01	-1	1.740017e-09
NUMNONZEROS:MAXOVERMINROWSUM	2.201226e-01	-1	2.318223e-09
STDAVGNNZPERROW:IS_SAI-FLT-0.9	2.147277e-01	-1	5.752145e-09
NUMCOLS:MAXOVERMINROWSUM	2.130134e-01	-1	7.640011e-09
MAXOVERMINROWSUM	1.996882e-01	-1	6.403093e-08
STDAVGNNZPERROW:MAXOVERMINROWSUM	1.981435e-01	-1	8.118152e-08
MAXBANDWIDTH:IS_SAI-TH-0.9	1.948508e-01	-1	1.337924e-07
STDAVGNNZPERROW:STDROWSUM	1.929104e-01	-1	1.788817e-07
AVGNNZPERROW:IS_SAI-TH-0.9	1.909092e-01	-1	2.406061e-07

evident from the high occurrence and the corresponding negative correlation of the features IS_FLT-0.9 and IS_TH-0.75.

3. Performance Modeling

In Section A, we used multi-variate linear regression to learn a performance model for the Hypre dataset. However, we observed that the performance metrics often exhibit non-linear dependence on the linear system and solver configuration characteristics. We use SVM regression (with a RBF kernel) available in the LibSVM package [16] since it allows us to handle the non-linear dependencies and provide better results in comparison to multi-variate linear regression. There is a lot of variability in the performance data since the matrices are from multiple domains and have widely varying properties. Therefore, we perform a log transformation on the matrix features and the target values as well as reorder the features based on a mutual information criteria before learning the performance models. In addition, we also normalize the resulting interaction feature vectors to have zero mean

and unit variance. For learning the best performance model, we follow an approach similar to solvability modeling, i.e., we perform a parallel grid search over the parameter space of C and γ and multiple feature sets of increasing sizes. We now discuss results on new trials on the train matrices and unseen test matrices. As in the case of solvability, we also present a brief analysis of the predictive features of the best time and memory models for the different solver configuration groups.

a. Results on New Trials Over Train Matrices

Figures 60 and 61 show the median relative error and the R2 statistic for 20% of the trials involving train matrices for the time taken and memory usage for all the three solver configuration groups. We consider median relative error instead of the mean in order to reduce the effect of the outliers. Although the model is learned on the log transformed target values, the results shown in Figures 60 and 61 are computed after applying the inverse of the log transformation. We observe that the median relative error for memory is very low (11 -16 %) and the high R^2 statistic (Section A) values indicate that the model achieves a good reduction of the squared error with respect to a constant model. In case of the time metric, the median relative error is slightly higher (17-18 %) and the R^2 statistic indicates that the predictions are reasonable with the exception of Hypre ParaSails where there is a significant variation across the cross folds.

b. Results on Unseen Test Matrices

Figure 62 and 63 shows the median relative error for trials associated with the unseen test matrices. The high median relative error for the predictions on unseen matrices relative to that of the new trials on train matrices indicates that the performance models might not have been able to accurately capture some of the matrix specific effects. This could be due to the fact that data corresponding to the solved trials used for training the performance

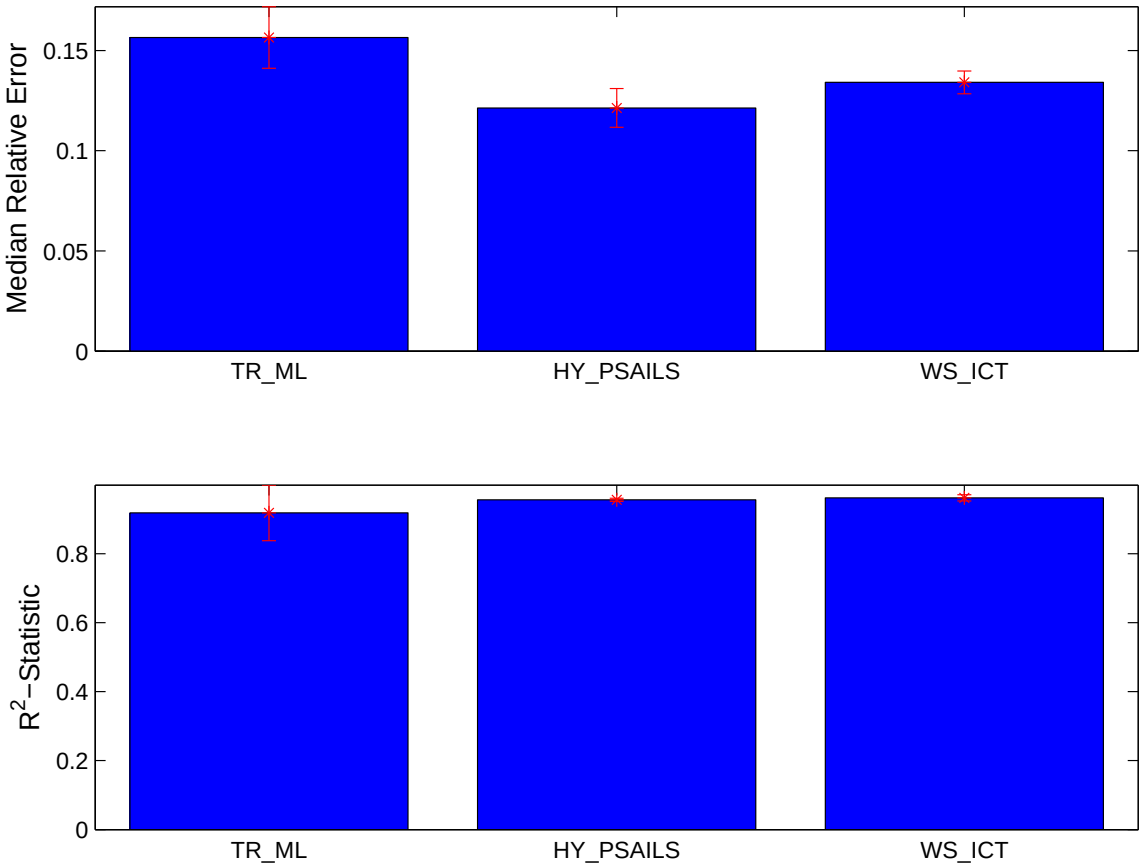


Fig. 60. Median relative error and R^2 statistic for memory prediction for 20 % hold out set of new trials comprising of matrices in the training set for Trilinos ML, Hypre ParaSails, and WSMP ICT. The performance values are averaged over 5 runs.

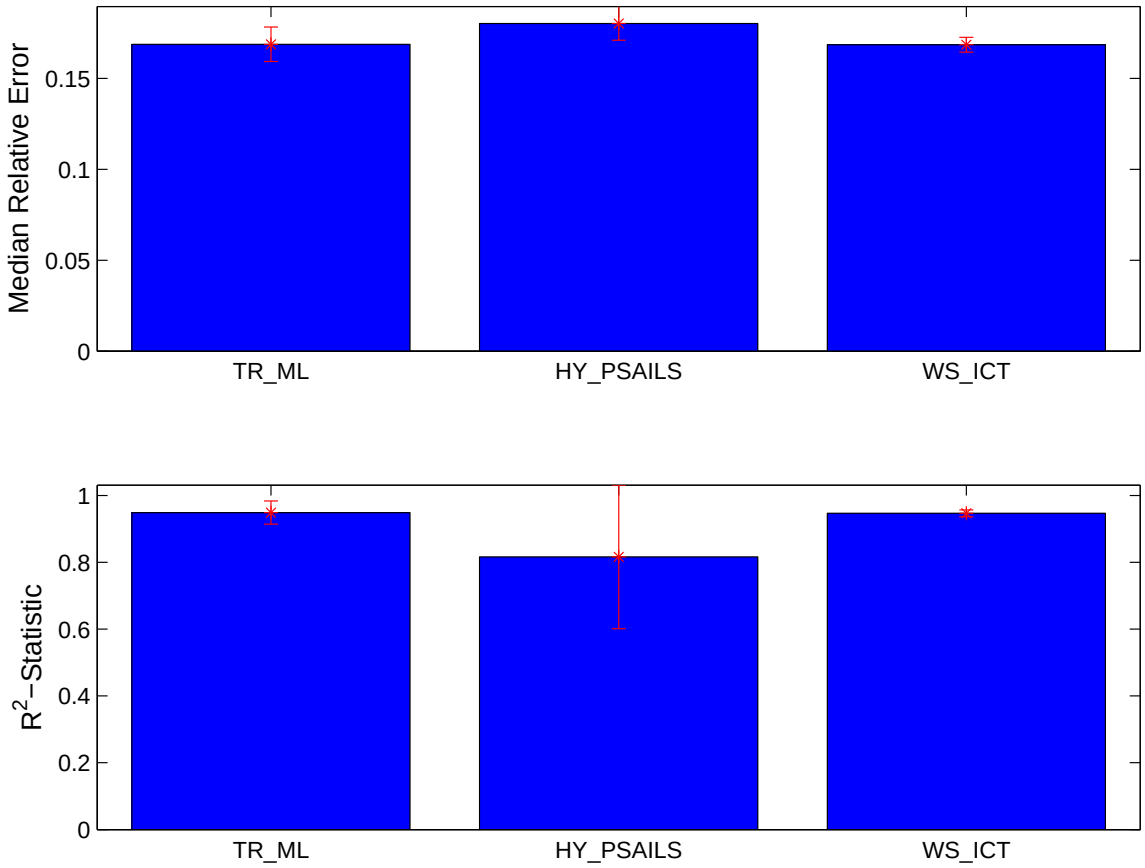


Fig. 61. Mean Relative error and R^2 statistic for time prediction for 20 % hold out set of trials comprising of matrices in the training set for Trilinos ML, Hypre ParaSails, and WSMP ICT. The performance values are averaged over 5 runs.

models is fairly small due to the large number of solver failures, and also because we only employ fairly simple linear system characteristics.

Since our main objective is to obtain recommendations for each matrix, it is sufficient to obtain prediction values correct up to a monotonic transformation. To obtain a better indication of the effectiveness of our predictions, we computed an alternative quality measure that adjusts for matrix-specific effects using an affine transformation. Specifically, we estimate for each matrix, the best multiplicative and additive factors that minimize the cumulative relative error of the predictions. The actual predictions are then transformed using the matrix-specific factors to obtain new predictions, which is then used to estimate the relative error. The bottom plots in Figures 62 and 63 show the median values for this adjusted relative error.

c. Model Analysis

As in the case of solvability models in Section B.2.c , we compute the top features based on the correlation of the support vectors associated with each model to the corresponding target response. We list the top 20 features that are highly correlated with respect to memory and time for Trilinos ML, Hypre ParaSails, and WSMP ICT in the form of tables. The third column shows if the feature is positively or negatively correlated and the fourth column indicates the p-value of the correlation coefficients. A very low value for p-values indicate that there is a non-zero correlation between the feature and the observed target variable. Note that the tables shown below are only for the top 20 features, therefore, the absence of certain solver features in the list does not imply the lack of interactions involving those features. There are also other important interaction features that are negatively correlated, which do not make it to the top 20.

Memory Usage. In the case of Trilinos ML, the use of IFPACK smoother (IS_ML-IFPACK) is highly correlated with the feature smoother sweep value of 1 (IS_ML-SS1)

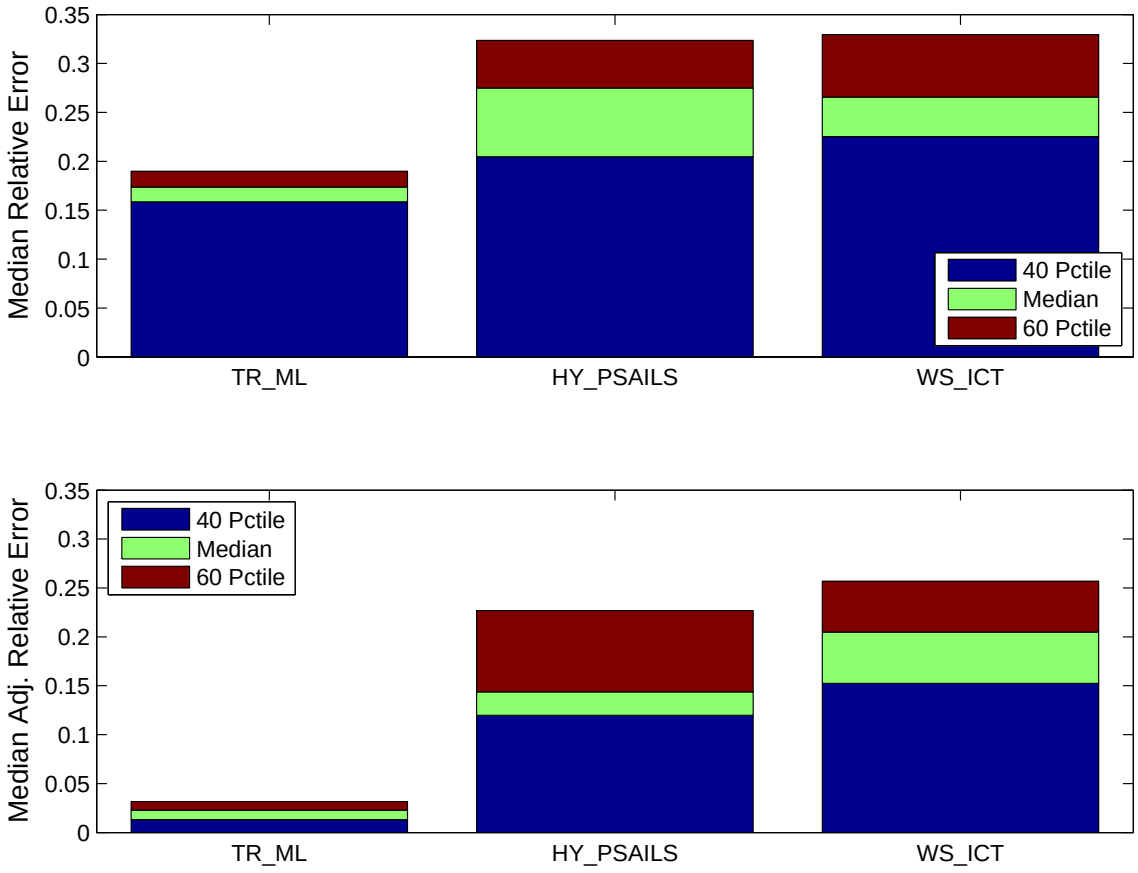


Fig. 62. Median Relative error and R^2 statistic for memory prediction using the best memory model on trials comprising of unseen matrices and solver configurations for Trilinos ML, Hypre ParaSails, and WSMP ICT.

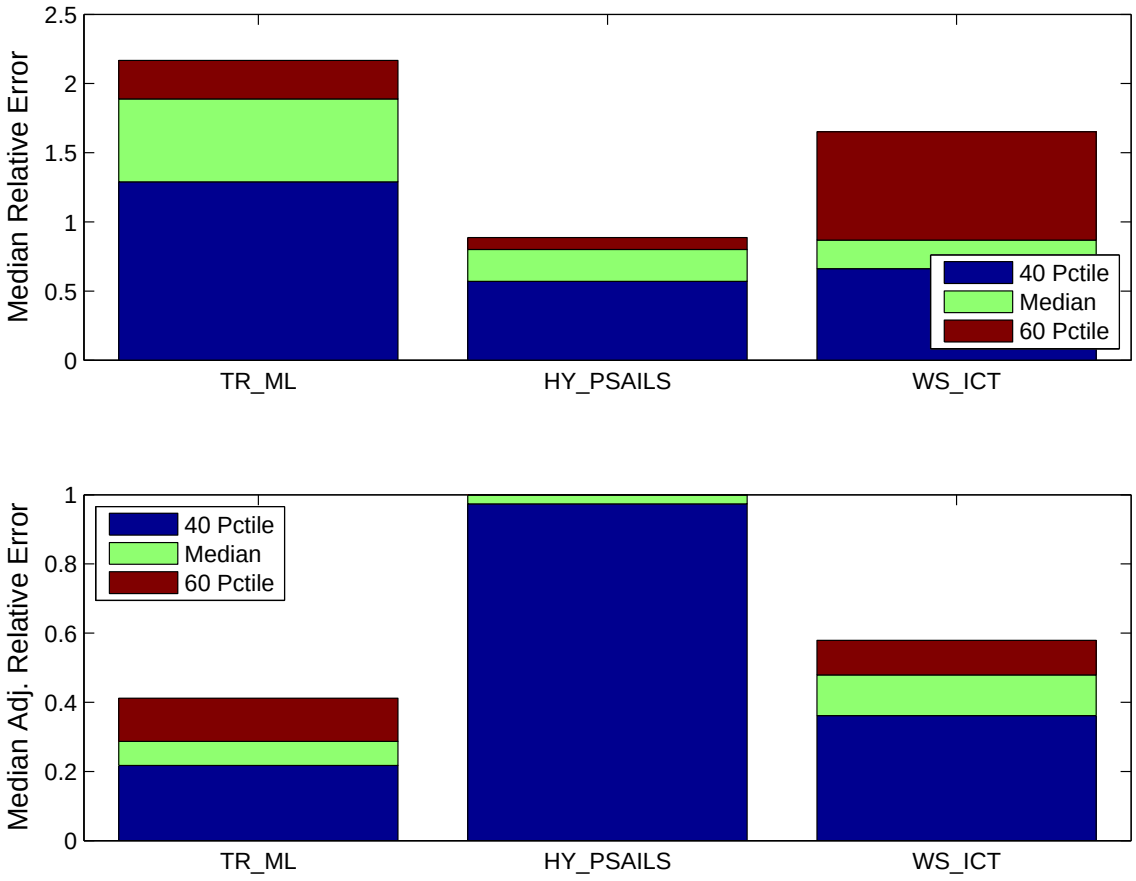


Fig. 63. Mean Relative error and R^2 statistic for time prediction using the best memory model on trials comprising of unseen matrices and solver configurations for Trilinos ML, Hypre ParaSails, and WSMP ICT.

Table XXIV. Top 20 interaction features selected for memory prediction in the case of Trilinos ML. The components of the interaction features are separated with a ":" and the solver features are prefixed with a "IS_".

Feature Name	Correlation	Sign	p-Value
AVGBANDWIDTH:IS_ML-IFPACK	6.255588e-01	1	1.199299e-10
AVGBANDWIDTH:IS_ML-SS1	6.255588e-01	1	1.199299e-10
MAXBANDWIDTH:IS_ML-IFPACK	6.192284e-01	1	2.078400e-10
MAXBANDWIDTH:IS_ML-SS1	6.192284e-01	1	2.078400e-10
NUMCOLS:IS_ML-IFPACK	5.954608e-01	1	1.473868e-09
NUMCOLS:IS_ML-SS1	5.954608e-01	1	1.473868e-09
NUMNONZEROS:IS_ML-IFPACK	5.919616e-01	1	1.940491e-09
NUMNONZEROS:IS_ML-SS1	5.919616e-01	1	1.940491e-09
IS_ML-SS1	5.617411e-01	1	1.831855e-08
IS_ML-IFPACK:IS_ML-SS1	5.617411e-01	1	1.831855e-08
IS_ML-IFPACK	5.617411e-01	1	1.831855e-08
GEOMDIMAVGBANDWIDTH:IS_ML-SS1	5.442973e-01	1	6.063428e-08
GEOMDIMAVGBANDWIDTH:IS_ML-IFPACK	5.442973e-01	1	6.063428e-08
GEOMDIMMAXBANDWIDTH:IS_ML-SS1	5.104237e-01	1	5.150522e-07
GEOMDIMMAXBANDWIDTH:IS_ML-IFPACK	5.104237e-01	1	5.150522e-07
AVGNNZPERROW:IS_ML-IFPACK	4.949126e-01	1	1.273039e-06
AVGNNZPERROW:IS_ML-SS1	4.949126e-01	1	1.273039e-06
ISORDERING-RCM:IS_ML-SS1	4.648037e-01	1	6.528074e-06
IS_ML-SA:IS_ML-SGS	3.906632e-01	-1	2.001932e-04
IS_ML-SGS	3.906632e-01	-1	2.001932e-04

since only the IFPACK smoother uses a smoother sweep of 1 in our experiments. In Chapter III, we observe that the use of IFPACK smoother results in very high memory usage especially in the case of ML-DD and ML-DD-ML default set of parameters. These observations are evident in Table XXIV in the form of high occurrence of features IS_ML-IFPACK and IS_ML-SS1. Also another important observation is that the use of smoothed aggregation with symmetric Gauss-Seidel smoother (IS_ML-SA:IS_ML-SGS) results in best memory usage. This is also captured by our model in the form of a high negative correlation for those features. Other linear system features that seem to affect the memory performance include the average and maximum bandwidth in addition to some other obvious features such as number of columns and number of non-zeros. In Table XXV, the linear system features and their interactions with each other are the dominant features for Hypre ParaSails. An interesting observation in the case of Hypre ParaSails is that matrices with high values for the product of standard deviation of diagonal elements and standard deviation of row sums are likely to have less memory usage. Even in the case of WSMP ICT in Ta-

ble XXVI, we observe that the linear system features dominate the top interaction features with respect to memory. The only solver specific feature that is prominent is the use of the lowest drop tolerance value of 3×10^{-4} (IS_DT3eE-4), which is as expected and confirmed by the observations for WSMP ICT in Chapter III.

Table XXV. Top 20 interaction features selected for memory prediction in the case of Hypre ParaSails. The components of the interaction features are separated with a ":" and the solver features are prefixed with a "IS_".

Feature Name	Correlation	Sign	p-Value
NUMCOLS:NUMNONZEROS	7.481570e-01	1	2.365968e-17
NUMNONZEROS	7.276030e-01	1	4.558446e-16
NUMCOLS	6.301348e-01	1	2.842705e-11
NUMNONZEROS:AVGBANDWIDTH	6.273709e-01	1	3.676034e-11
NUMCOLS:GEOMDIMAVGBANDWIDTH	6.006792e-01	1	3.883344e-10
AVGBANDWIDTH:GEOMDIMAVGBANDWIDTH	5.724516e-01	1	3.750409e-09
NUMCOLS:AVGBANDWIDTH	5.621981e-01	1	8.115016e-09
NUMNONZEROS:MAXBANDWIDTH	5.422520e-01	1	3.386493e-08
NUMCOLS:GEOMDIMMAXBANDWIDTH	5.384312e-01	1	4.406081e-08
AVGBANDWIDTH:GEOMDIMMAXBANDWIDTH	5.268219e-01	1	9.611605e-08
NUMNONZEROS:GEOMDIMAVGBANDWIDTH	5.090499e-01	1	3.002244e-07
PERCENTAGEWEAKDIAGDOMROWS:STDROWSUM	4.941968e-01	1	7.410533e-07
NUMCOLS:MAXBANDWIDTH	4.910021e-01	1	8.950696e-07
MAXBANDWIDTH:GEOMDIMAVGBANDWIDTH	4.844044e-01	1	1.314067e-06
MAXBANDWIDTH:GEOMDIMMAXBANDWIDTH	4.284834e-01	1	2.517528e-05
PERCENTAGEWEAKDIAGDOMROWS:AVGDIAGDOM	4.187313e-01	1	4.002935e-05
AVGNNZPERROW:AVGBANDWIDTH	4.029771e-01	1	8.220672e-05
NUMNONZEROS:GEOMDIMMAXBANDWIDTH	3.966251e-01	1	1.087846e-04
STDROWSUM:STDDIAG	3.943092e-01	-1	1.203141e-04
AVGBANDWIDTH:IS_SAI-FLT0	3.461256e-01	1	8.327912e-04

Time Taken. Tables XXVII– XXIX show the top 20 features for time prediction for Trilinos ML, Hypre ParaSails, and WSMP ICT respectively. Unlike the top memory features for Trilinos ML, the top 20 features for time predictions includes primarily the linear system interaction features as shown in Table XXVII.

However, in the case of Hypre ParaSails in Table XXVIII, we do see a number of solver parameters in the top 20 interaction features. Most notable of these are the threshold parameter (IS_SAI-TH-0.75) and the filter parameter (IS_Flt-0.9). The interesting observation here is that the values chosen for these parameters are negative (Chapter II). The use of negative values results in much sparser less effective preconditioners which probably took a lot more iterations and resulted in longer solve times. For WSMP ICT, in Table XXIX,

Table XXVI. Top 20 interaction features selected for memory prediction in the case of WSMP ICT. The components of the interaction features are separated with a ":" and the solver features are prefixed with a "IS_".

Feature Name	Correlation	Sign	p-Value
NUMNONZEROS	5.614960e-01	1	1.125756e-20
NUMCOLS:NUMNONZEROS	5.285367e-01	1	4.276189e-18
PERCENTAGEWEAKDIAGDOMROWS:STDROWSUM	4.965937e-01	1	7.527504e-16
NUMCOLS:GEOMDIMAVGBANDWIDTH	4.683495e-01	1	4.742490e-14
AVGDIAGDOM:STDROWSUM	4.444692e-01	1	1.184003e-12
NUMNONZEROS:GEOMDIMAVGBANDWIDTH	4.290071e-01	1	8.349970e-12
NUMCOLS	4.007930e-01	1	2.305390e-10
NUMCOLS:GEOMDIMMAXBANDWIDTH	3.792790e-01	1	2.367121e-09
NUMCOLS:AVGNNZPERROW	3.320913e-01	1	2.234824e-07
AVGBANDWIDTH:GEOMDIMAVGBANDWIDTH	3.220396e-01	1	5.362933e-07
NUMNONZEROS:AVGBANDWIDTH	3.103979e-01	1	1.421728e-06
NUMNONZEROS:GEOMDIMMAXBANDWIDTH	3.019918e-01	1	2.802362e-06
NUMCOLS:AVGBANDWIDTH	2.691424e-01	1	3.263911e-05
GEOMDIMAVGBANDWIDTH:IS_DT3E-4	2.672155e-01	1	3.734027e-05
AVGBANDWIDTH:GEOMDIMMAXBANDWIDTH	2.632559e-01	1	4.907649e-05
AVGNNZPERROW:AVGBANDWIDTH	2.609636e-01	1	5.737680e-05
NUMNONZEROS:AVGNNZPERROW	2.561553e-01	1	7.926385e-05
PERCENTAGEWEAKDIAGDOMROWS:AVGDIAGDOM	2.559759e-01	1	8.021554e-05
NUMNONZEROS:IS_DT3E-4	2.477447e-01	1	1.374222e-04
STDROWSUM:SUPNODESIZE	2.475719e-01	-1	1.389571e-04

we observe that the time model is predominantly affected by linear system features more than the solver-specific features.

Table XXVII. Top 20 interaction features selected for time prediction in the case of Trilinos ML. The components of the interaction features are separated with a ":" and the solver features are prefixed with a "IS_".

Feature Name	Correlation	Sign	p-Value
NUMNONZEROS	5.786686e-01	1	5.148768e-24
NUMCOLS:NUMNONZEROS	4.728667e-01	1	1.685356e-15
NUMNONZEROS:GEOMDIMAVGBANDWIDTH	4.514101e-01	1	4.177258e-14
NUMCOLS:GEOMDIMAVGBANDWIDTH	4.431370e-01	1	1.357957e-13
AVGBANDWIDTH:GEOMDIMAVGBANDWIDTH	4.027647e-01	1	2.760662e-11
NUMCOLS:GEOMDIMMAXBANDWIDTH	3.926788e-01	1	9.353786e-11
NUMNONZEROS:GEOMDIMMAXBANDWIDTH	3.765591e-01	1	6.047343e-10
AVGBANDWIDTH:GEOMDIMMAXBANDWIDTH	3.661556e-01	1	1.912658e-09
PERCENTAGEWEAKDIAGDOMROWS:STDROWSUM	3.561514e-01	1	5.571392e-09
MAXBANDWIDTH:GEOMDIMAVGBANDWIDTH	3.537509e-01	1	7.161582e-09
STDROWSUM:STDDIAG	3.467770e-01	-1	1.467839e-08
AVGNNZPERROW:GEOMDIMAVGBANDWIDTH	3.332355e-01	1	5.629584e-08
NUMNONZEROS:AVGBANDWIDTH	3.330439e-01	1	5.735056e-08
MAXBANDWIDTH:GEOMDIMMAXBANDWIDTH	3.161632e-01	1	2.801437e-07
NUMNONZEROS:MAXBANDWIDTH	3.023165e-01	1	9.585510e-07
AVGNNZPERROW:GEOMDIMMAXBANDWIDTH	2.802685e-01	1	5.984878e-06
NUMCOLS:AVGBANDWIDTH	2.508332e-01	1	5.467695e-05
GEOMDIMAVGBANDWIDTH:GEOMDIMMAXBANDWIDTH	2.416269e-01	1	1.036201e-04
STDROWSUM:SUPNODESIZE	2.229059e-01	-1	3.528651e-04
STDAVGNNZPERROW:STDROWSUM	2.226271e-01	-1	3.590962e-04

Table XXVIII. Top 20 interaction features selected for time prediction in the case of Hypre ParaSails. The components of the interaction features are separated with a ":" and the solver features are prefixed with a "IS_".

Feature Name	Correlation	Sign	p-Value
STDAVGNNZPERROW:IS_SAI-TH-0.75	5.009327e-01	1	9.883243e-18
AVGNNZPERROW:IS_SAI-TH-0.75	4.273959e-01	1	7.780590e-13
NUMNONZEROS	3.961742e-01	1	4.342587e-11
GEOMDIMMAXBANDWIDTH:IS_SAI-TH-0.75	3.758721e-01	1	4.781476e-10
STDAVGNNZPERROW:IS_SAI-FLT-0.9	3.751373e-01	1	5.199250e-10
AVGNNZPERROW:AVGBANDWIDTH	3.593030e-01	1	3.006691e-09
GEOMDIMAVGBANDWIDTH:IS_SAI-TH-0.75	3.561506e-01	1	4.216207e-09
ISSAI-LEV2:IS_SAI-TH-0.75	3.468293e-01	1	1.121486e-08
AVGNNZPERROW:MAXBANDWIDTH	3.413456e-01	1	1.964918e-08
NUMNONZEROS:GEOMDIMAVGBANDWIDTH	3.412530e-01	1	1.983411e-08
NUMCOLS:AVGNNZPERROW	3.223788e-01	1	1.259138e-07
NUMNONZEROS:IS_SAI-TH-0.75	3.222282e-01	1	1.277209e-07
AVGBANDWIDTH:IS_SAI-TH-0.75	3.222271e-01	1	1.277340e-07
MAXBANDWIDTH:IS_SAI-TH-0.75	3.212582e-01	1	1.399716e-07
NUMNONZEROS:GEOMDIMMAXBANDWIDTH	3.211771e-01	1	1.410447e-07
NUMCOLS:GEOMDIMMAXBANDWIDTH	3.136813e-01	1	2.831180e-07
AVGNNZPERROW:IS_SAI-FLT-0.9	3.114605e-01	1	3.467714e-07
STDAVGNNZPERROW:IS_SAI-LEV2	3.049816e-01	1	6.207253e-07
NUMNONZEROS:AVGNNZPERROW	3.003735e-01	1	9.312764e-07
NUMCOLS:GEOMDIMAVGBANDWIDTH	2.988053e-01	1	1.067449e-06

Table XXIX. Top 20 interaction features selected for time prediction in the case of WSMP ICT. The components of the interaction features are separated with a ":" and the solver features are prefixed with a "IS_".

Feature Name	Correlation	Sign	p-Value
NUMNONZEROS	5.999327e-01	1	9.233014e-29
NUMCOLS:NUMNONZEROS	5.694695e-01	1	1.807497e-25
NUMCOLS:GEOMDIMAVGBANDWIDTH	5.593037e-01	1	1.911092e-24
NUMNONZEROS:GEOMDIMAVGBANDWIDTH	5.061484e-01	1	1.271693e-19
AVGDIAGDOM:STDROWSUM	4.902143e-01	1	2.470716e-18
NUMCOLS:GEOMDIMMAXBANDWIDTH	4.760276e-01	1	3.053193e-17
NUMCOLS	4.194657e-01	1	2.329602e-13
AVGBANDWIDTH:GEOMDIMAVGBANDWIDTH	4.034504e-01	1	2.194610e-12
NUMNONZEROS:GEOMDIMMAXBANDWIDTH	3.906249e-01	1	1.215357e-11
NUMCOLS:AVGNNZPERROW	3.668004e-01	1	2.414296e-10
GEOMDIMAVGBANDWIDTH	3.536717e-01	1	1.132463e-09
AVGBANDWIDTH:GEOMDIMMAXBANDWIDTH	3.452565e-01	1	2.940328e-09
MAXBANDWIDTH:GEOMDIMAVGBANDWIDTH	3.401505e-01	1	5.174868e-09
NUMNONZEROS:AVGBANDWIDTH	3.370599e-01	1	7.250253e-09
GEOMDIMAVGBANDWIDTH:GEOMDIMMAXBANDWIDTH	2.950967e-01	1	4.949988e-07
NUMCOLS:AVGBANDWIDTH	2.940299e-01	1	5.465242e-07
AVGNNZPERROW:AVGBANDWIDTH	2.925470e-01	1	6.267630e-07
NUMCOLS:STDAVGNNZPERROW	2.872416e-01	1	1.016727e-06
AVGDIAGDOM:MAXOVERMINROWSUM	2.840208e-01	1	1.357308e-06
NUMNONZEROS:AVGNNZPERROW	2.825888e-01	1	1.541570e-06

4. Top-k Recommendations

We now present results on the top-k recommendations for the matrices in our collection. Similar to Section A, we rank the solver configurations based on memory usage, computational time, and memory-time product and measure the quality of the recommendations in terms of top-k precision and average quality value of the top-k recommendations. For measuring the average quality of the recommendations, we normalize the performance values by the problem-independent best (PIB) choice values, as defined in Chapter III. Note that for the performance data involving train matrices, the values are averaged over the contributing matrices for each value of k over 5 crossfold splits whereas the standard deviations are for the matrix averaged values over the crossfold splits alone. However, for individual plots involving unseen matrices, we report the actual values for each matrix. We compare the quality of the recommendations by comparing it with the PIB choice as well as the actual top-k performance values, or in other words, the problem-specific best choice (PSB). An important point to remember is that the PIB choice is a solver configuration optimized over the entire training performance data. Therefore, expecting the recommendations to always improve on the performance of the PIB configuration is not reasonable, especially in case of unseen matrices where the problem specific best performance is very close to the PIB performance.

a. Results on New Trials Over Train Matrices

We now present results on the top-k precision and the average quality of recommendations for solver group specific models for Trilinos ML, Hypre ParaSails, and WSMP ICT.

Trilinos ML. Figure 64 shows the top-10 precision of the solver recommendations for memory, time, and memory-time product for Trilinos ML solver configurations. Our approach identifies around 50% of the actual best configurations for memory and around

45% of the best configurations with respect to time. With respect to memory time product, however, the precision falls to 25%. Figure 65 shows the performance improvement that can be obtained using the generated recommendations over the PIB choice (dotted line). In case of memory, the PIB and PSB curves are fairly close and the recommendations almost follow the PIB curve. However, in the case of time, we observe that the recommended fine-tuning curve is always lower than the PIB choice and quite close to the PSB curve. The recommendations for memory-time product are not as good as the time ones since it seems to be dominated by the memory model.

Hypre ParaSails. Figure 66 shows the top-10 precision of Hypre ParaSails solver recommendations for memory, time, and memory-time product. We are able to identify 70% of the actual best configurations for memory-time product, around 65% of the best configurations for time, and around 55 % for memory. Figure 67 shows the average quality of the top-k recommendations over the PIB choice (dotted line). In case of all the three metrics, we observe that the recommender fine-tuning curve is always lower than the PIB choice and comparable to the PSB curve indicating that the model can provide high quality recommendations for the train matrices.

WSMP ICT. The top-10 precision of Hypre ParaSails solver recommendations for memory, time, and memory-time product is shown in Figure 68. With respect to memory and memory-time product, we are able to identify almost 75% of the actual best configurations. Figure 69 shows the comparison of the average quality of the top-k recommendations with that of the PIB and PSB choices. In case of all the three metrics, we observe that the recommendations perform better than the PIB choice and are again very close to the ideal PSB values.

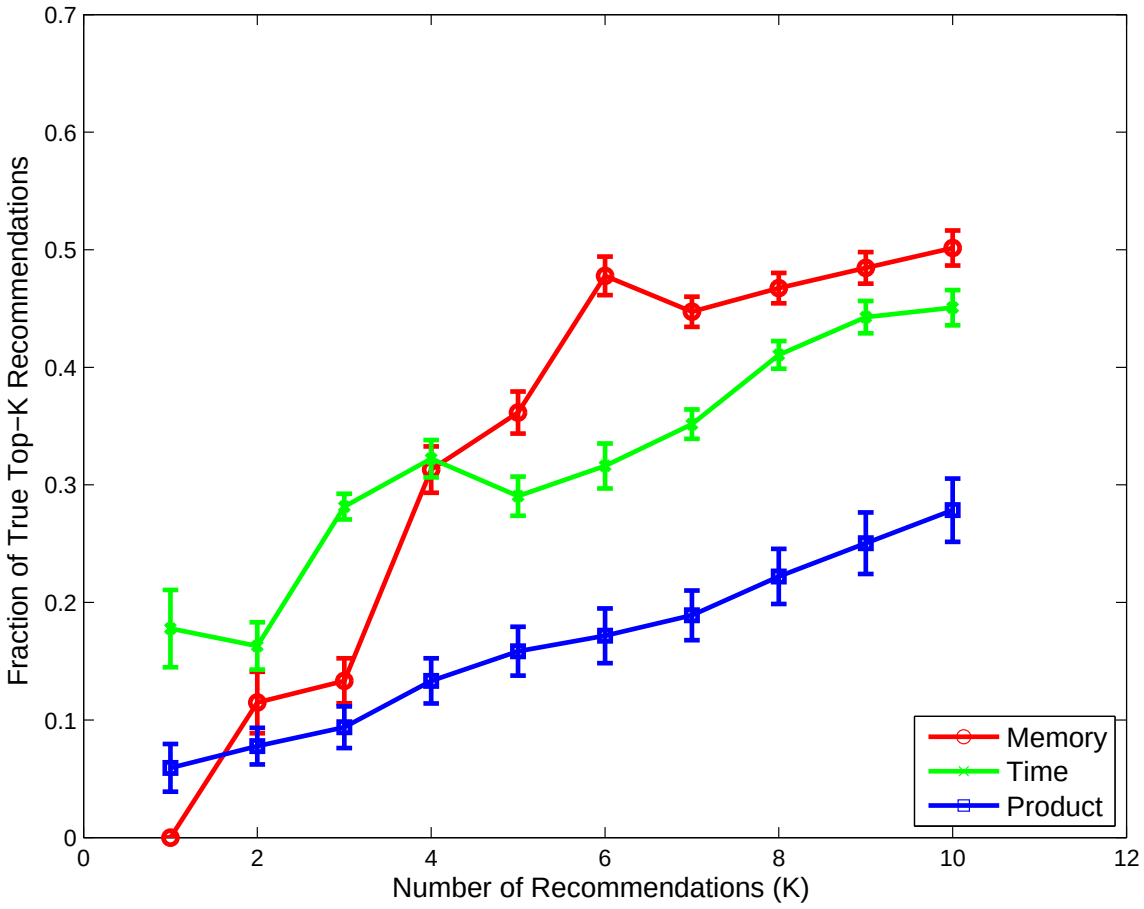


Fig. 64. Fraction of the true best choices for memory, time, and memory-time product that is present in top-k recommendations for Trilinos ML (66 configurations) for the seen matrices. The precision values shown are averaged over the solved problems over 5 runs.

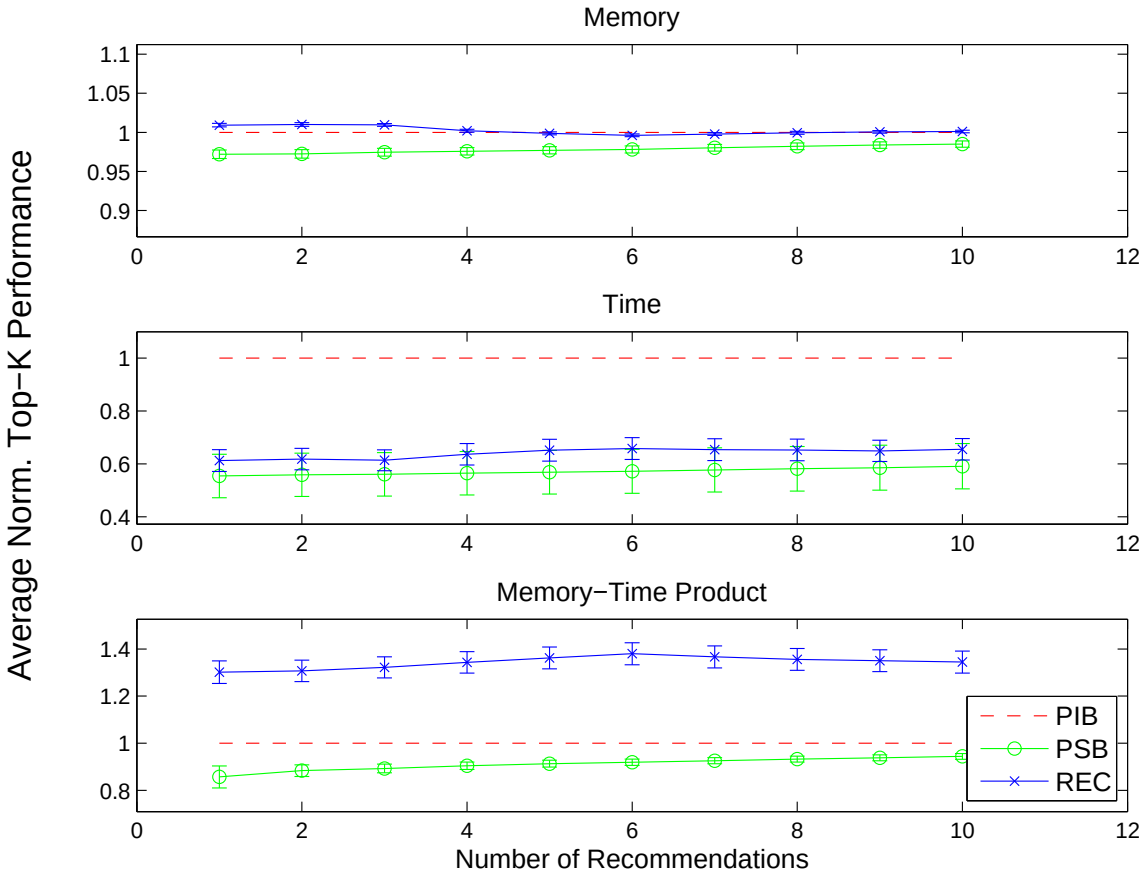


Fig. 65. Average performance with respect to memory, time and memory-time product for the recommended (REC) and problem specific best (PSB) Trilinos ML configurations normalized by the corresponding problem independent best (PIB) values for the seen matrices. For each k, the performance values are averaged over the solved problems for 5 runs.

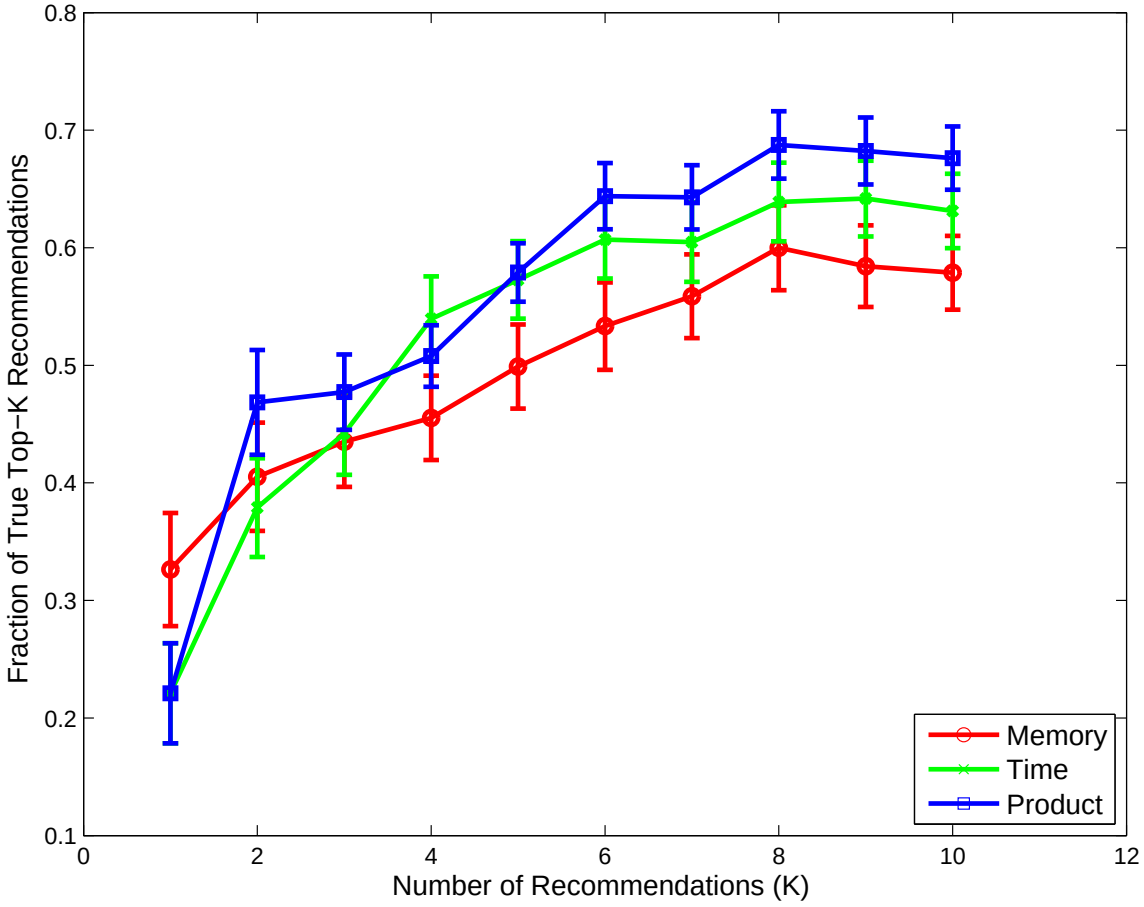


Fig. 66. Fraction of the true best choices for memory, time, and memory-time product that is present in top-k recommendations for Hypre ParaSails (99 configurations) for the seen matrices. The precision values shown are averaged over the solved problems for 5 runs.

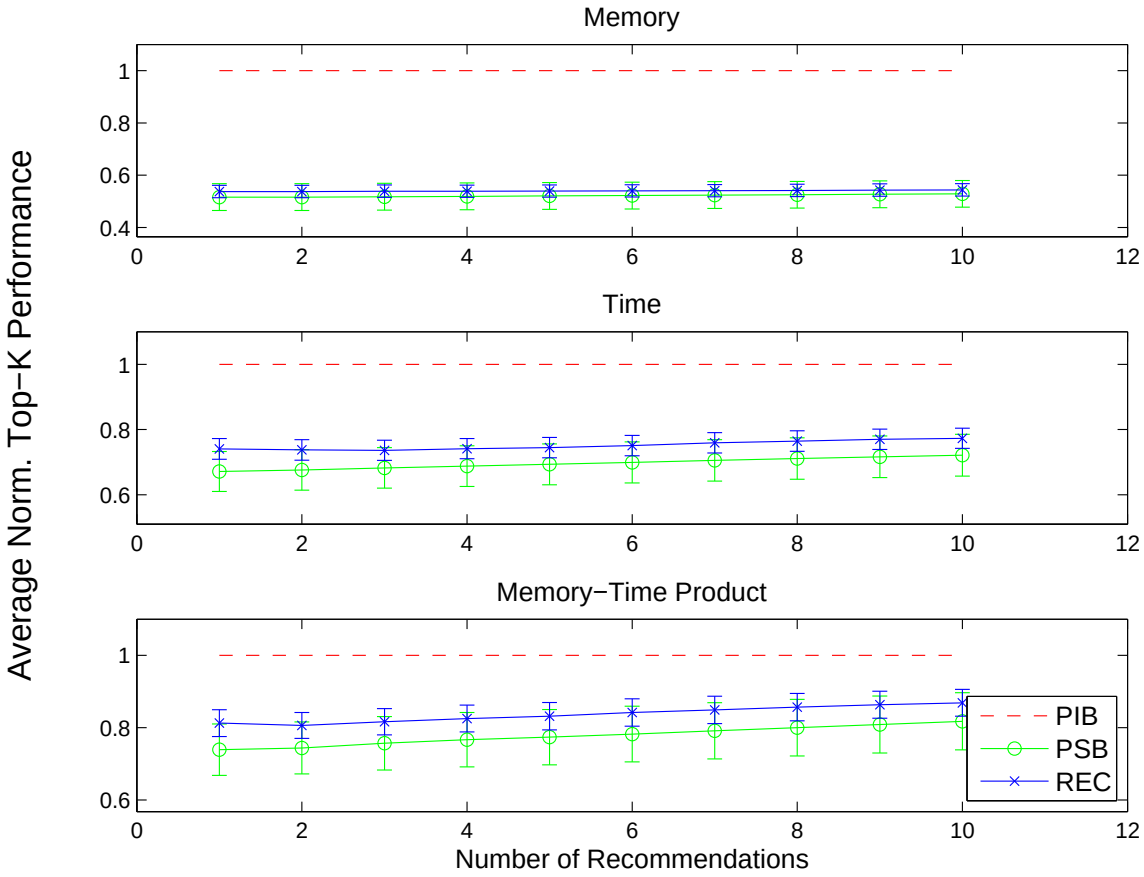


Fig. 67. Average performance with respect to memory, time and memory-time product for the recommended (REC) and problem specific best (PSB) Hypre ParaSails configurations normalized by the corresponding problem independent best (PIB) values for the seen matrices. For each k, the performance values are averaged over the solved problems for 5 runs.

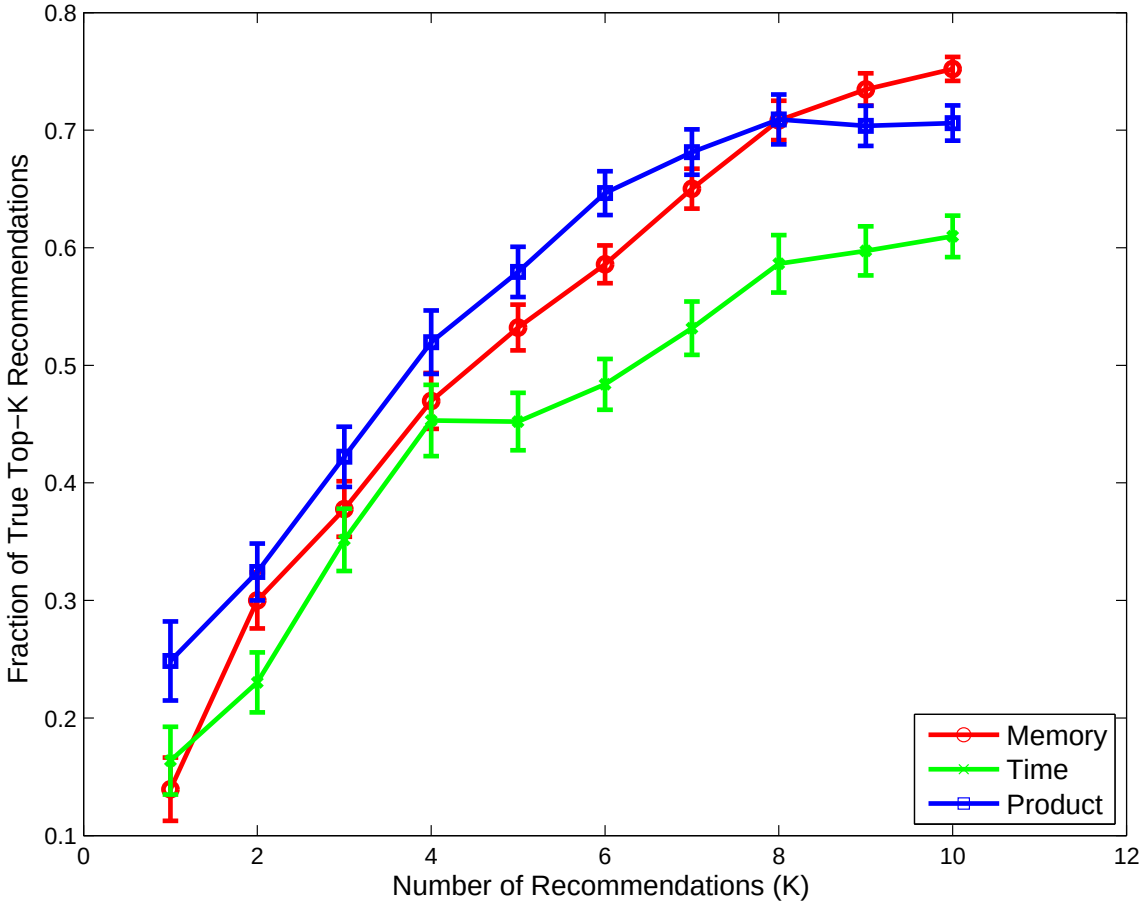


Fig. 68. Fraction of the true best choices for memory, time, and memory-time product that is present in top-k recommendations for WSMP ICT (64 configurations) for the seen matrices. The precision values shown are averaged over the solved problems for 5 runs.

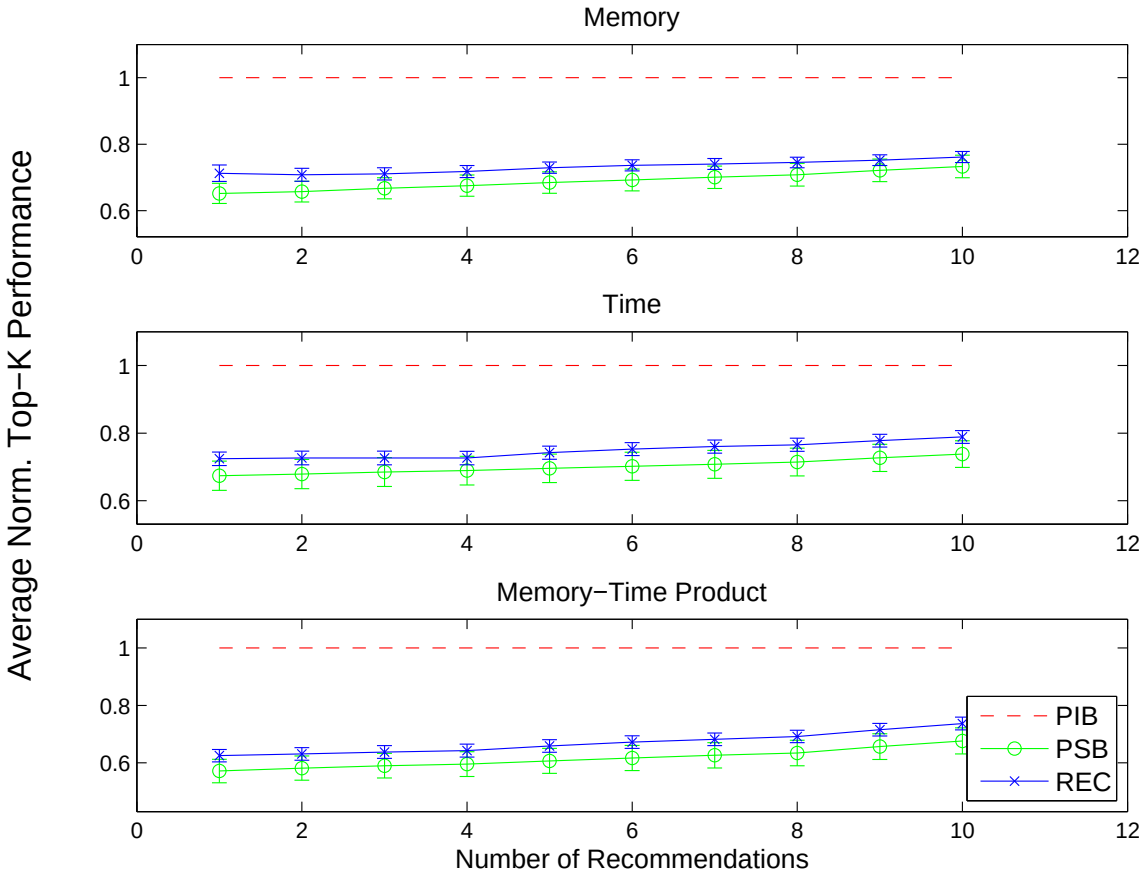


Fig. 69. Average performance with respect to memory, time and memory-time product for the recommended (REC) and problem specific best (PSB) WSMP ICT configurations normalized by the corresponding problem independent best (PIB) values for the seen matrices. For each k, the performance values are averaged over the solved problems for 5 runs.

b. Results on Unseen Test Matrices

Trilinos ML. Figure 70 shows the average top-k precision values for the unseen matrices. The average top-k precisions of recommendations are only slightly lower than those in the case of seen matrices in Figure 64 though the standard deviations are quite high due to the small sample. Individual top-k precisions for four sample matrices (*audikw_1*, *ldoor*, *mstamp-2c*, *90153*) are shown in Figure 71. With the exception of *ldoor*, the performance models are able to predict a reasonable fraction of the top 10 actual best configurations with respect to memory, time, and memory-time product. The average quality of the recommendations is shown in Figure 72 and the individual recommendation quality plots for four sample matrices are shown in Figure 73. The average recommendations for time and memory lie between the PIB and PSB curves though there is a significant variation among the matrices. However, the individual plots in Figure 73 indicate that a decent fraction of the recommendations are comparable to the PIB and PSB values.

Hypre ParaSails. The average top-k precision and individual top-k plots for unseen matrices are shown in Figures 74 and 75 respectively. With the exception of memory, the average precision is low relative to Trilinos-ML, but significantly better than a random ranking since there are 99 configurations to choose from. The corresponding average recommendation quality and individual plots are shown in Figures 76 and 77 respectively. Even though the top-k precision values are low, the actual performance values are still better than the PIB choice and are close to the problem specific best values. The plot for *audikw_1* in Figure 76 is an example of a case where the PIB solver configuration could not solve it. The recommendations for all the three metrics are able to capture the problem specific best choices in the very first few recommendations.

WSMP ICT. Figures 78 and 79 show the average and individual top-k precision plots for WSMP ICT. All the matrices have reasonable precision with respect to memory

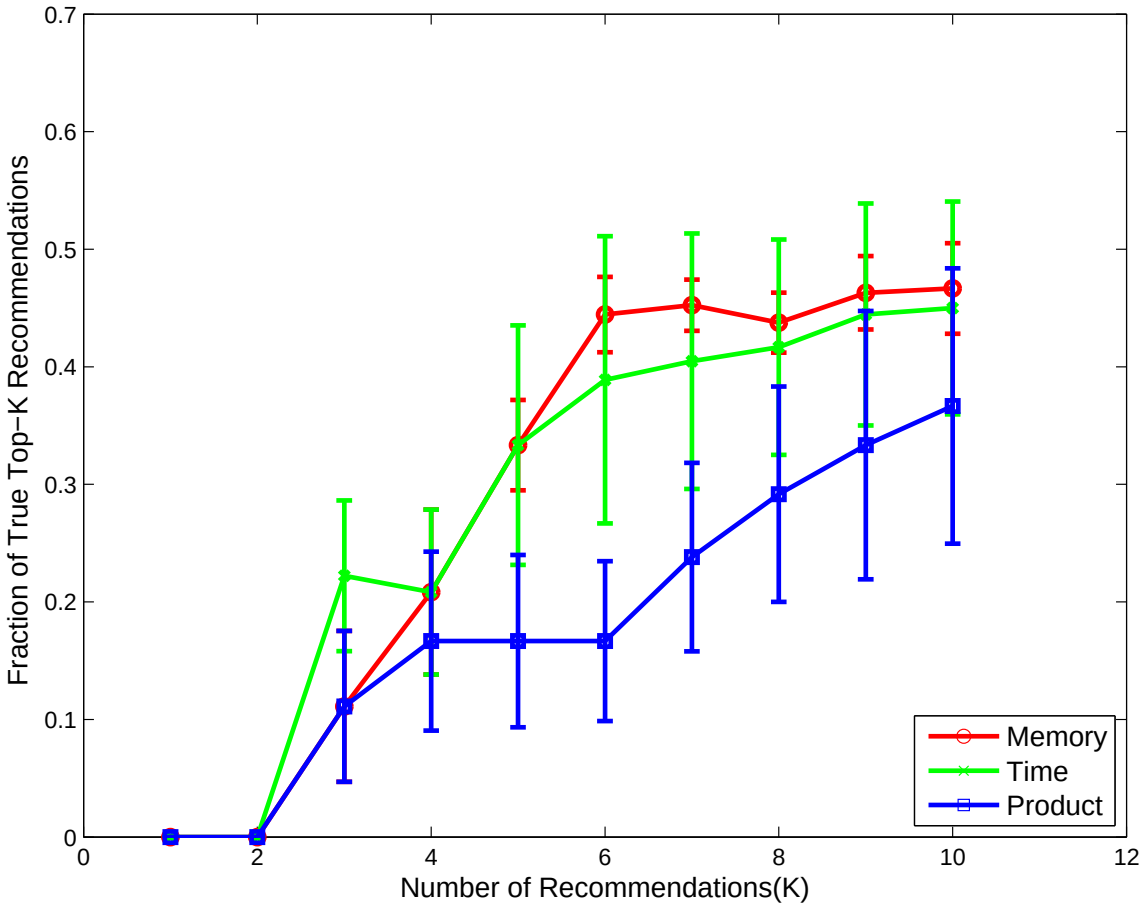


Fig. 70. Fraction of the true best choices for memory, time, and memory-time product that is present in top-k recommendations for Trilinos ML (66 configurations) in the case of unseen matrices. The precision values are averaged over the solved problems.

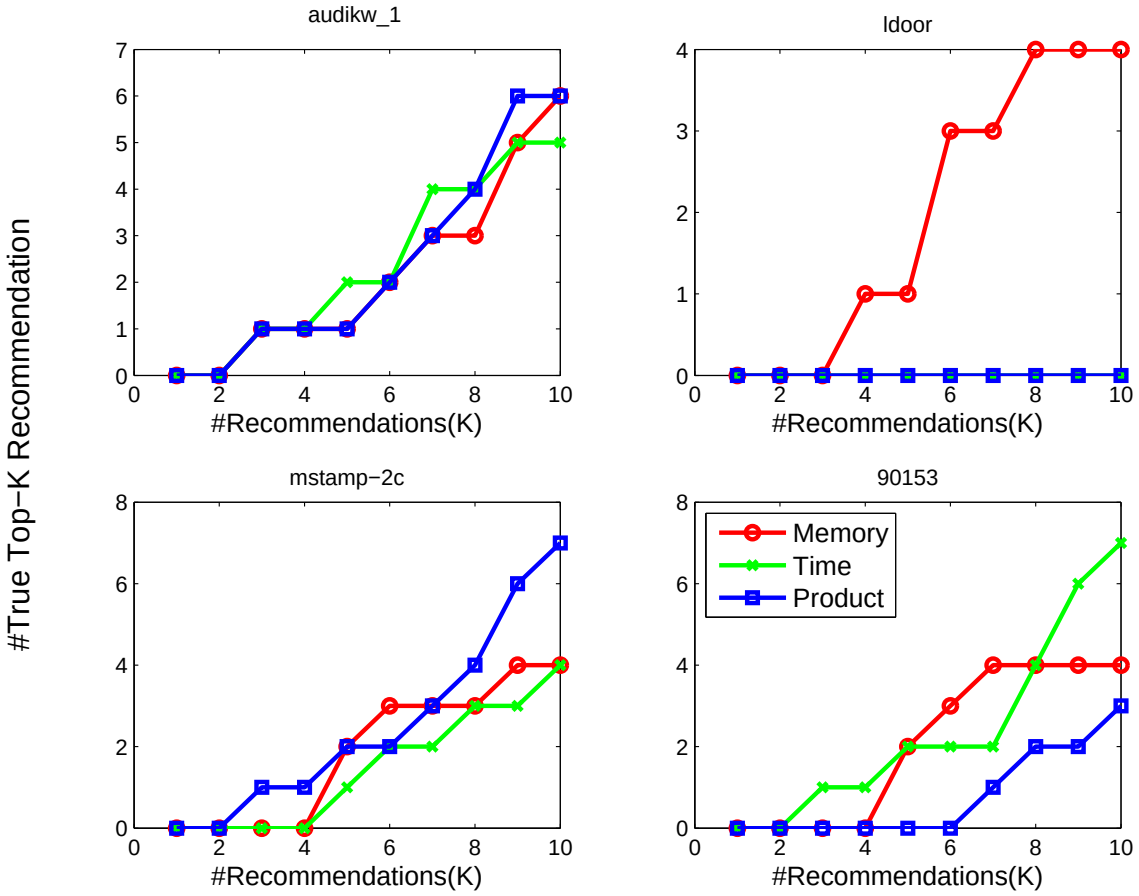


Fig. 71. The number of true best choices for memory, time, and memory-time product that is present in top-k recommendations for Trilinos ML (66 configurations) in the case of *audikw_1*, *ldoor*, *mstamp-2c*, and *90153* matrices.

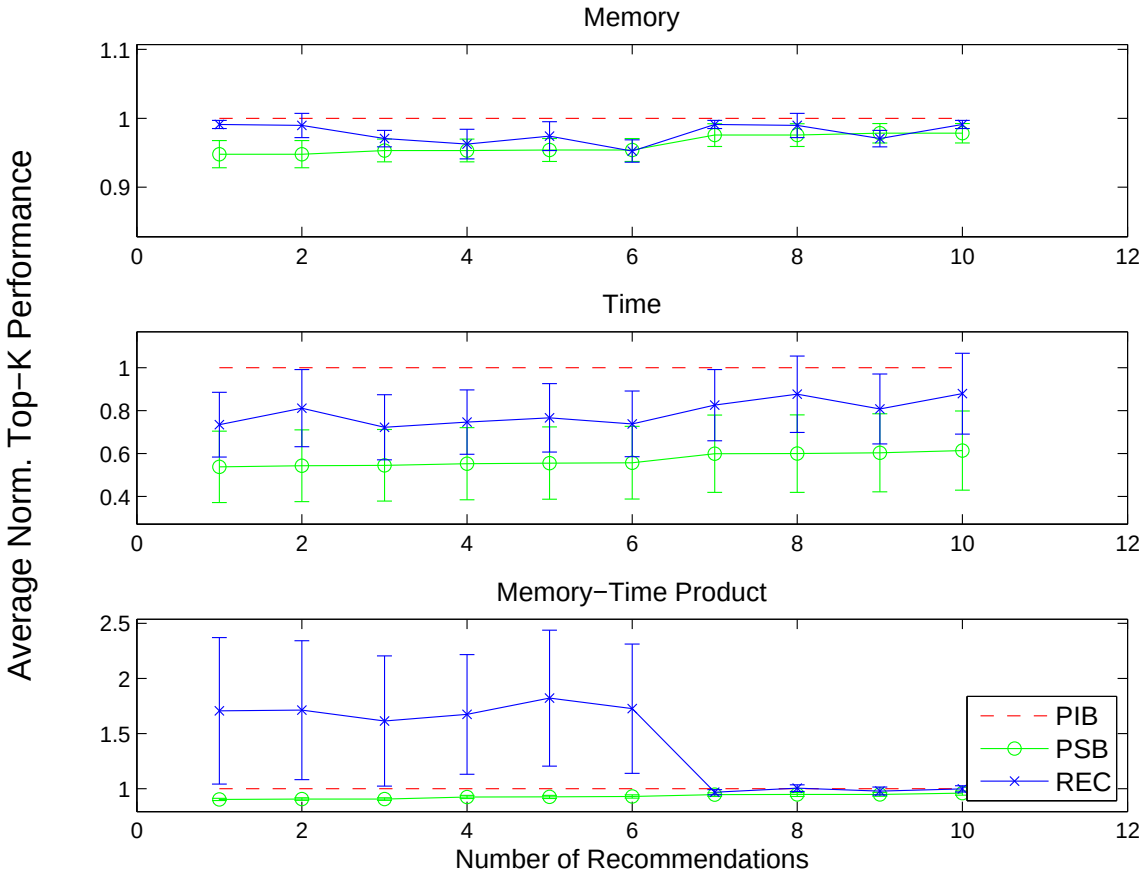
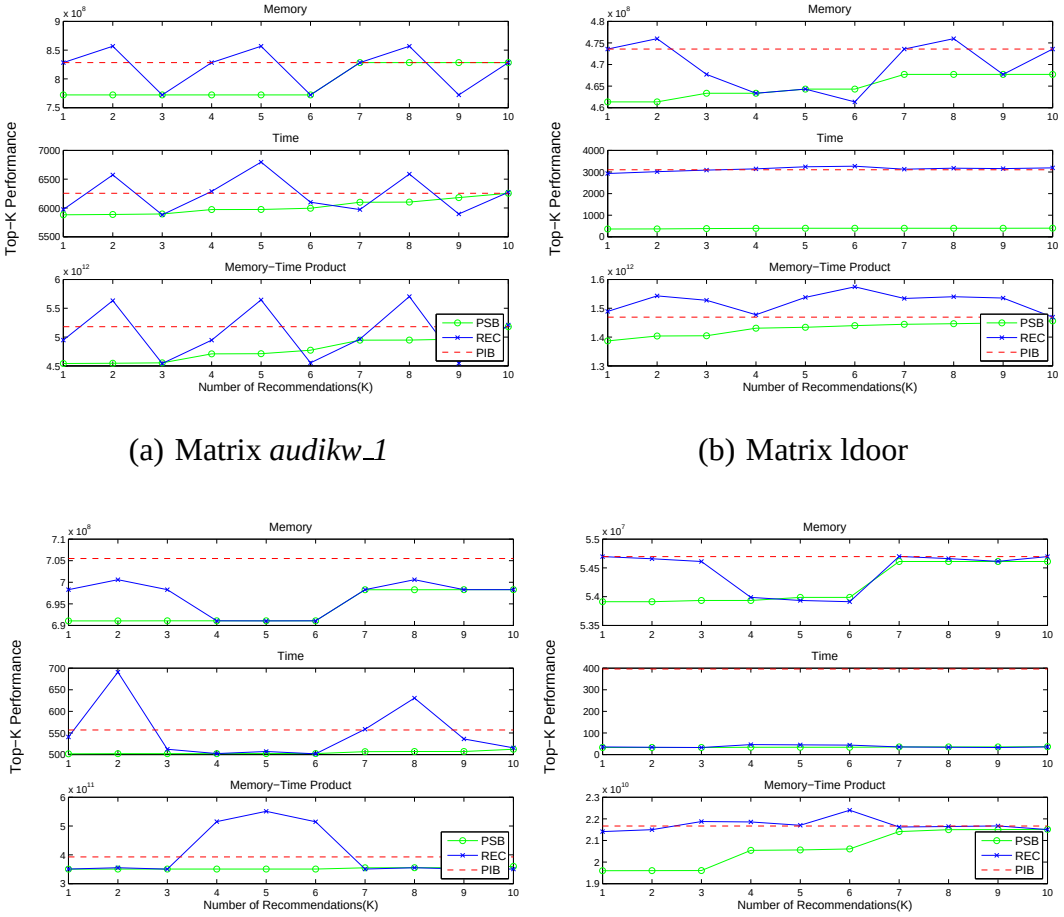


Fig. 72. Average performance with respect to memory, time and memory-time product for recommended (REC) and problem specific best (PSB) Trilinos ML configurations normalized by the corresponding problem independent best (PIB) values for the unseen matrices. For each k, the performance values are averaged over the solved problems.



(a) Matrix *audikw_1*

(b) Matrix ldoor

(c) Matrix mstamp-2c

(d) Matrix 90153

Fig. 73. Performance with respect to memory, time and memory-time product for the k^{th} recommended (REC) and problem specific best (PSB) Trilinos ML configurations for *audikw_1*, *ldoor*, *mstamp-2c*, and *90153* matrices. Where applicable, the problem independent best (PIB) performance values are also plotted.

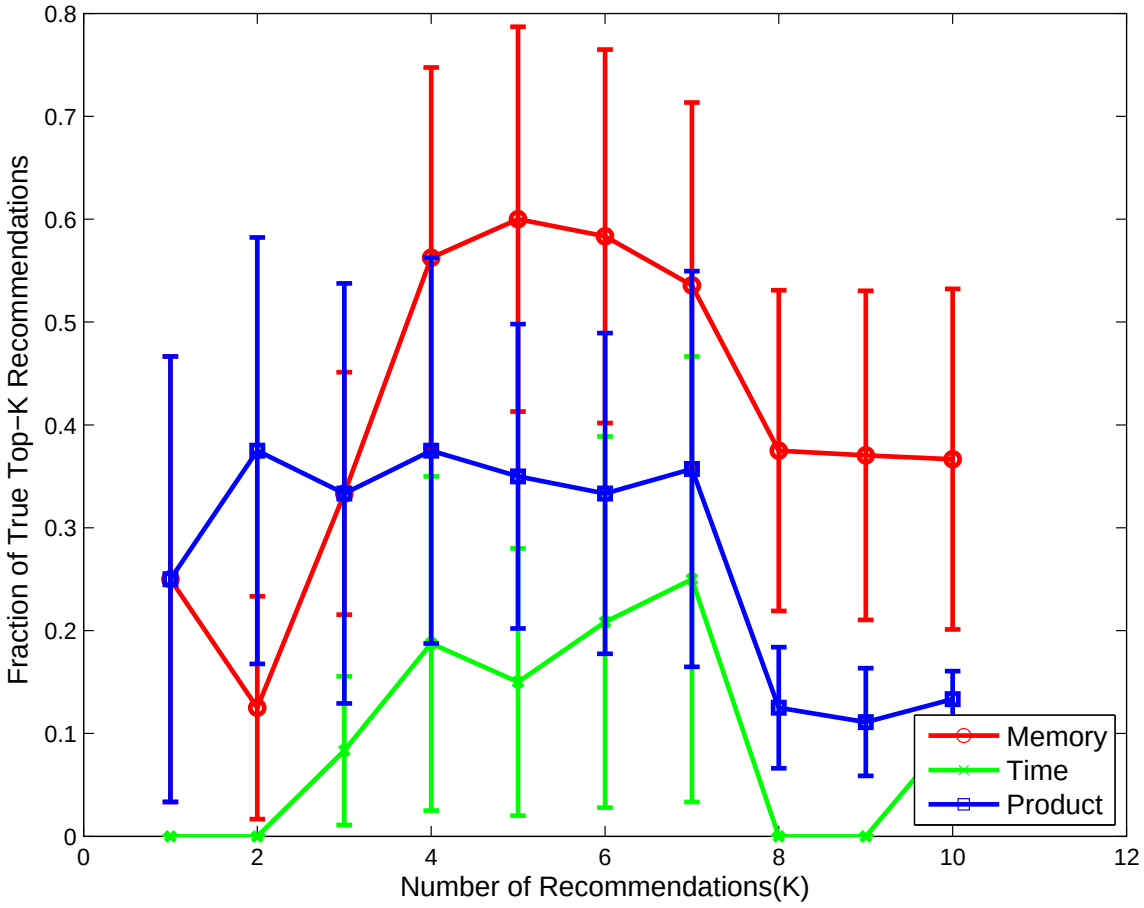


Fig. 74. Fraction of the true best choices for memory, time, and memory-time product that is present in top-k recommendations for Hypre ParaSails (99 configurations) in the case of unseen matrices. The precision values shown are averaged over the solved problems.

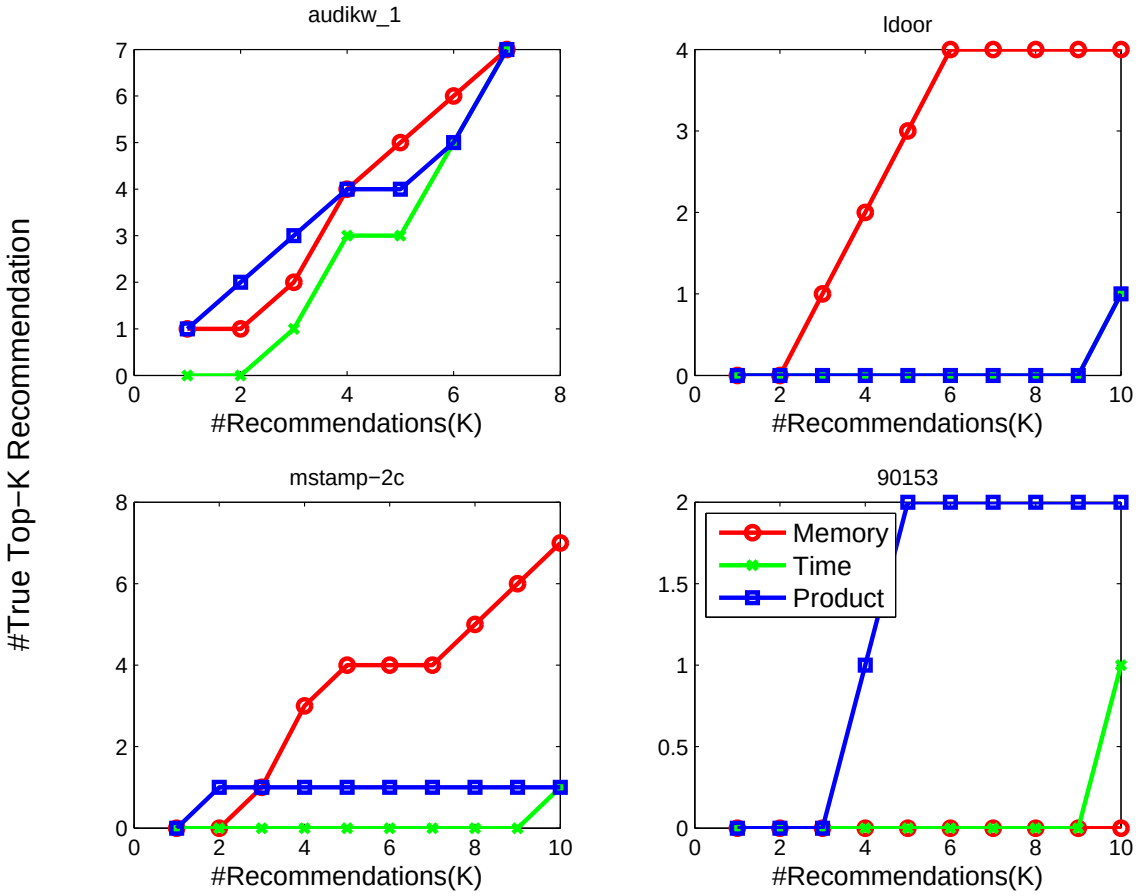


Fig. 75. The number of true best choices for memory, time, and memory-time product that is present in top-k recommendations for Hypre ParaSails (99 configurations) in the case of *audikw_1*, *ldoor*, *mstamp-2c*, and *90153* matrices.

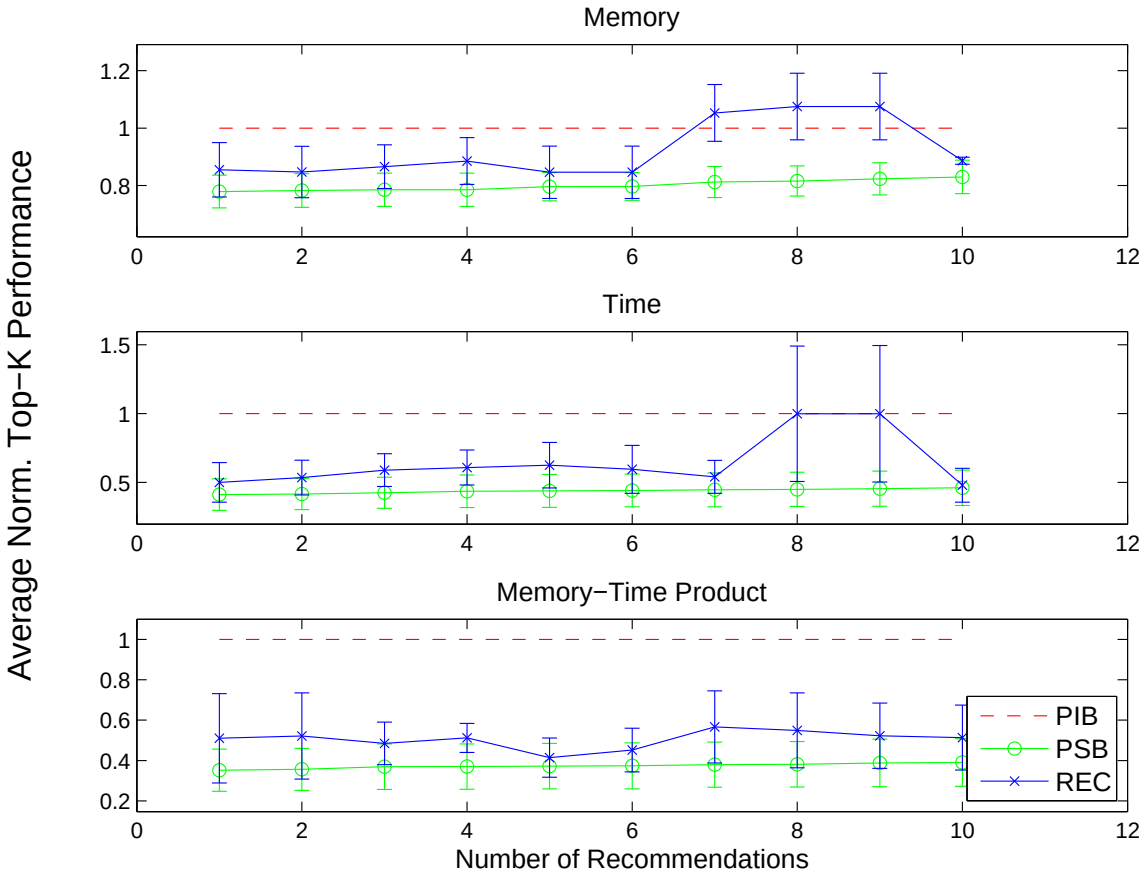
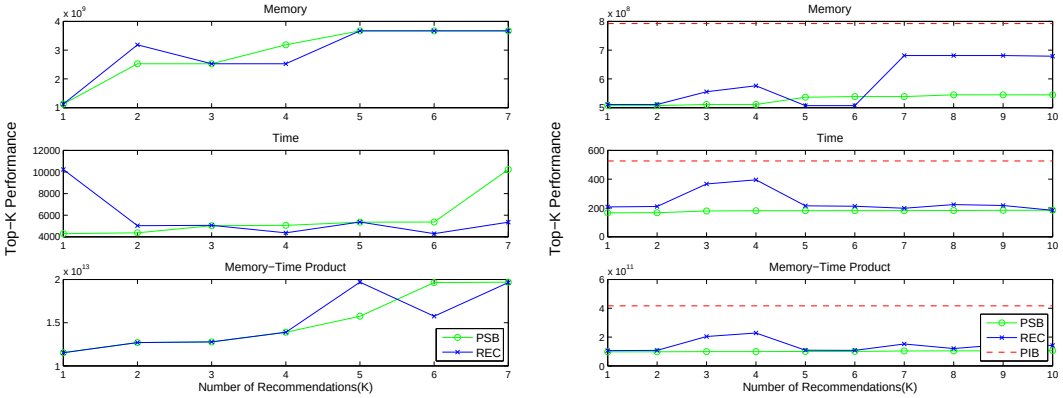


Fig. 76. Average performance with respect to memory, time and memory-time product for recommended (REC) and problem specific best (PSB) Hypre ParaSails configurations normalized by the corresponding problem independent best (PIB) values for the unseen matrices. For each k, the performance values are averaged using the solved problems.



(a) Matrix *audikw_1*

(b) Matrix ldoor

(c) Matrix mstamp-2c

(d) Matrix 90153

Fig. 77. Performance with respect to memory, time and memory-time product for the k^{th} recommended (REC) and problem specific best (PSB) Hypre ParaSails configurations for *audikw_1*, *ldoor*, *mstamp-2c*, and *90153* matrices. Where applicable, the problem independent best (PIB) performance values are also plotted.

and time. The average quality of recommendations in Figure 80 indicates that the recommendations are slightly better than the PIB values. However, some of the representative individual plots in Figure 81 indicate that the recommendations are different for the various matrices based on the metrics. For example, in the case of *mstamp-2c*, the first few memory, time, and memory time product recommendations are pretty close to the problem specific configurations. There are few cases where there is a non trivial gap between the first few recommended configurations and PSB values, however, the recommended configuration curve does intersect the PIB and PSB curves at multiple instances.

5. Discussion

A general trend that is observed in all these results is that it is possible to obtain reasonably good recommendations with respect to core performance metrics as well as hybrid combinations such as memory-time product. These recommendations are even more invaluable in scenarios where the problem independent best configuration is not able to solve a problem or if there are numerous solver configurations to choose from. Even though the predicted values for performance obtained for unseen matrices are not highly accurate, the recommendations based on ordering these predicted values are fairly competitive and most often have a significant overlap with the problem specific best values. These results are encouraging since there are very few feasible trials to accurately capture the matrix effects. The decrease in prediction quality of the solvability and performance models with respect to unseen matrices indicates that there is potential for improvement in our modeling procedure, especially with respect to capturing matrix-specific effects by including more informative matrix characteristics.

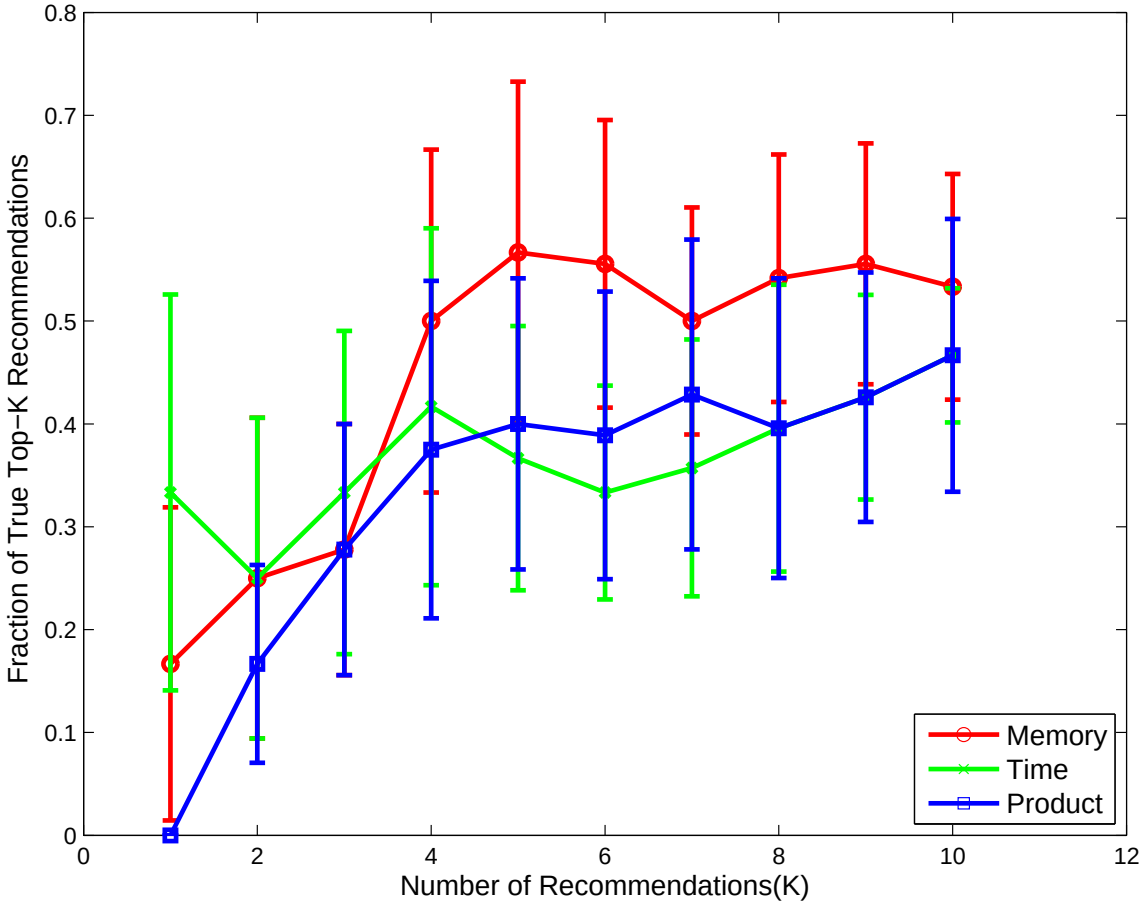


Fig. 78. Fraction of the true best choices for memory, time, and memory-time product that is present in top-k recommendations for WSMP ICT (64 configurations) in the case of unseen matrices. The precision values shown are averaged over the solved problems.

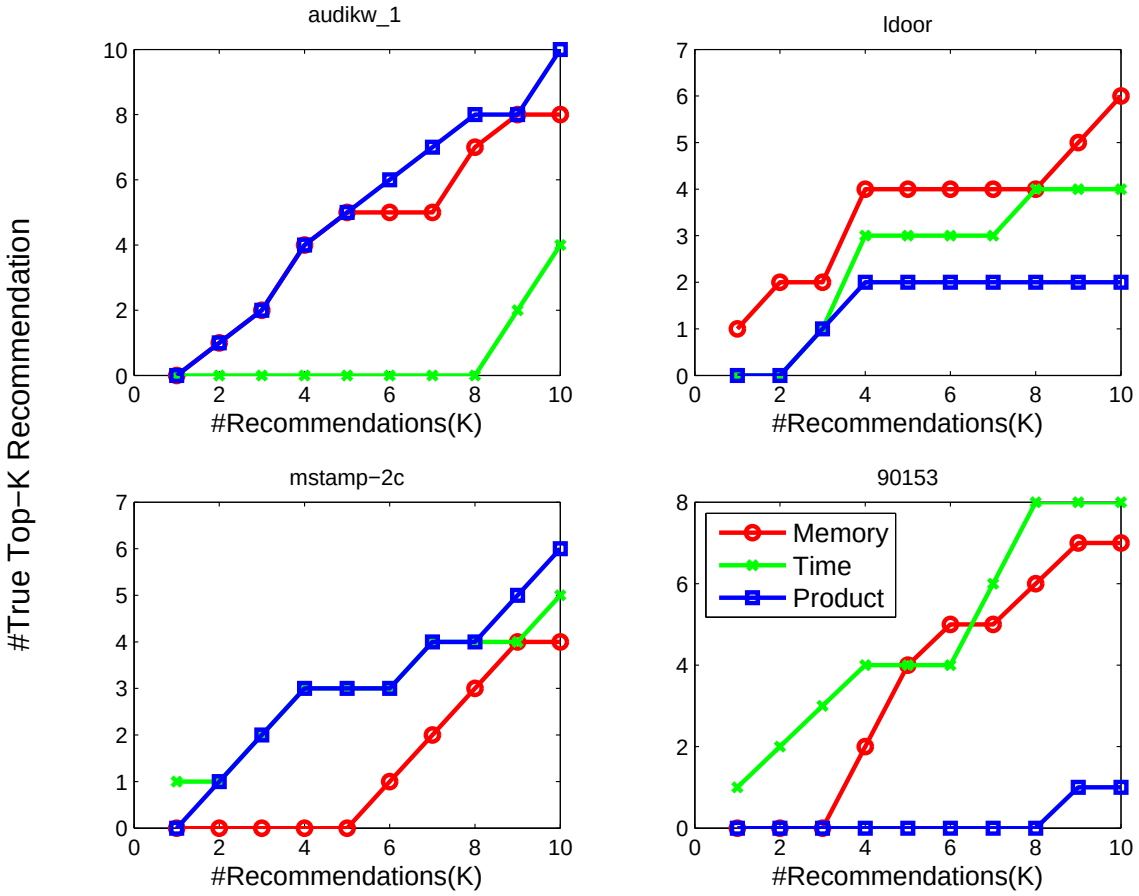


Fig. 79. The number of true best choices for memory, time, and memory-time product that is present in top-k recommendations for WSMP ICT (64 configurations) in the case of *audikw_1*, *ldoor*, *mstamp-2c*, and *90153* matrices.

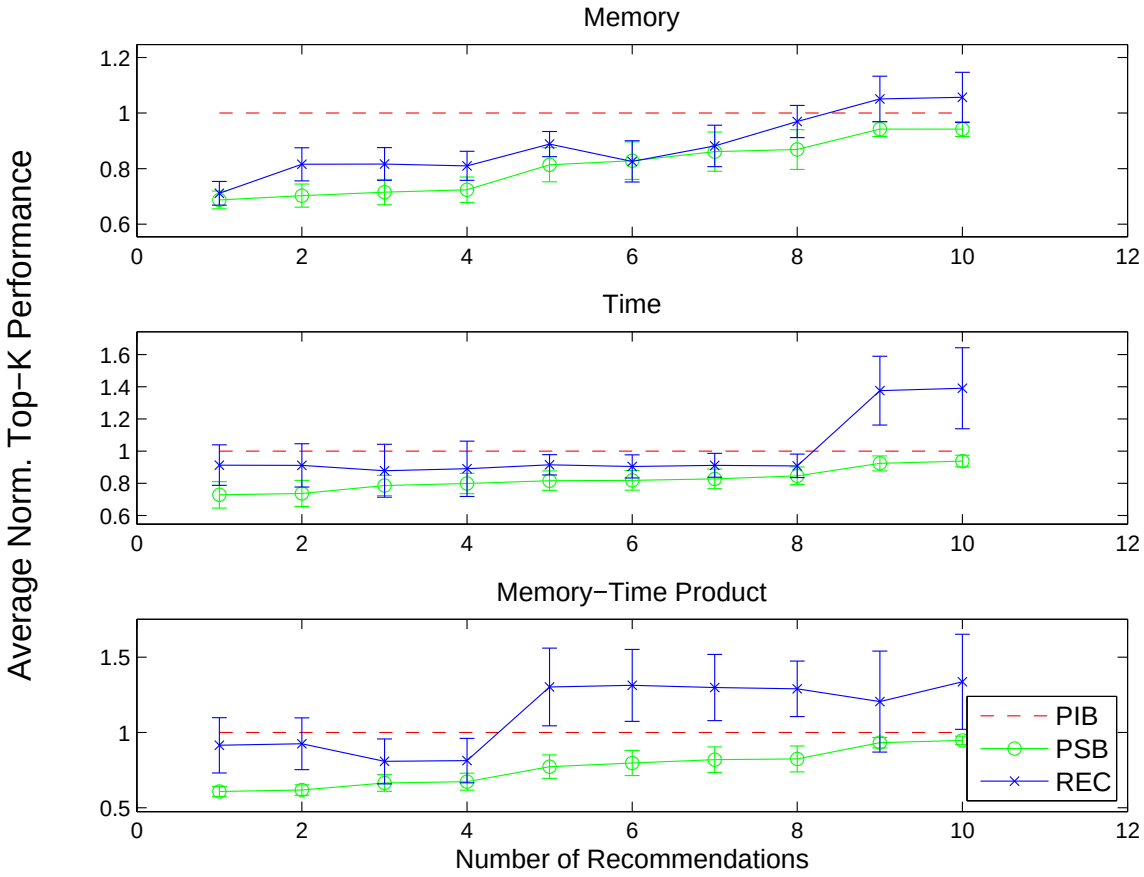
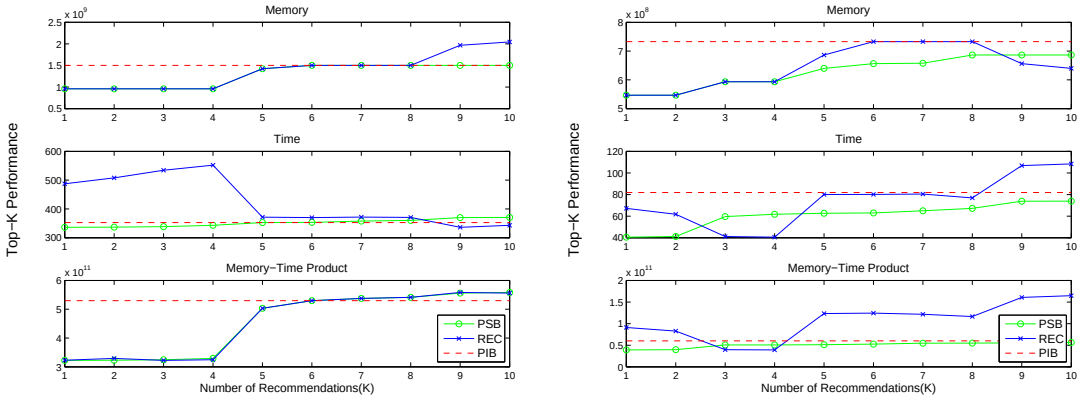
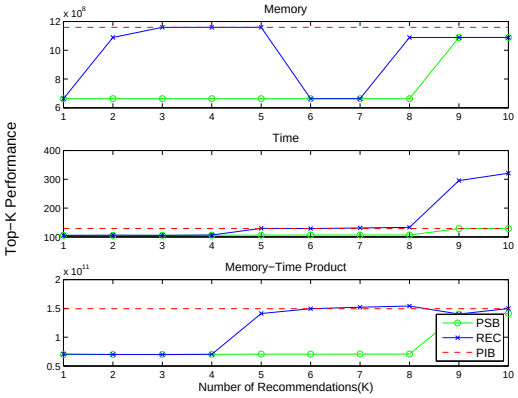


Fig. 80. Average performance with respect to memory, time and memory-time product for recommended (REC) and problem specific best (PSB) WSMP ICT configurations normalized by the corresponding problem independent best (PIB) values for the unseen matrices. For each k, the performance values are averaged over the solved problems.

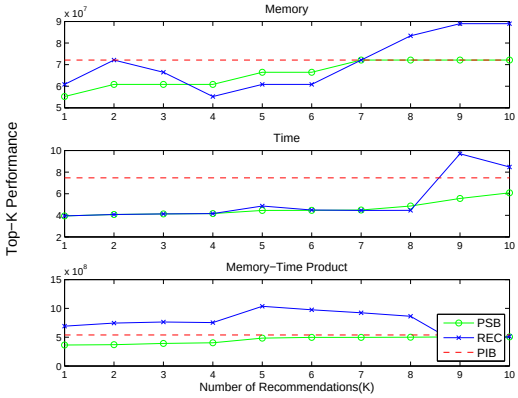


(a) Matrix *audikw_1*

(b) Matrix *ldoor*



(c) Matrix *mstamp-2c*



(d) Matrix *90153*

Fig. 81. Performance with respect to memory, time and memory-time product for the k^{th} recommended (REC) and problem specific best (PSB) WSMP ICT configurations for *audikw_1*, *ldoor*, *mstamp-2c*, and *90153* matrices. Where applicable the problem independent best (PIB) performance values are also plotted.

C. Case Study: Sheet Metal Forming

The results for unseen matrices in Sections B.2 and B.3 highlighted the some of the drawbacks of the learned models in providing accurate predictions with respect to memory and time performance. This was predominantly due to the unseen matrices being vastly different from that of the train matrices. However, if an application requires the solution of a number of matrices, that are very similar, then the learned models could be used for obtaining good performance even on unseen matrices. In order to demonstrate the utility of the recommendation system, we collected performance data on 6 SPD matrices from the UFL collection [1]. All the matrices vary only slightly with respect to structural and numerical properties and are obtained from the application domain of sheet metal forming. The solvability and performance models were learned using three matrices (*af_0_k101*, *af_2_k101*, *af_4_k101*) and tested on the remaining three (*af_1_k101*, *af_3_k101*, *af_5_k101*). In this section, we present the results on solvability prediction, performance prediction, and effectiveness of the top-k recommendations for Trilinos ML and WSMP ICT solver configurations. None of the Hypre ParaSails could solve the problems to the required accuracy in 1000 iterations and are therefore, omitted in this set of experiments.

Figure 82 shows the accuracy, precision, and recall for solvability prediction on the unseen matrices. In the case of WSMP ICT we are able to obtain 100 % accuracy which is much better than the baseline. However, in the case of Trilinos ML the solvability prediction accuracy is slightly lower than the baseline. Figures 83 and 84 show the median relative error and R^2 statistic with respect to memory and time prediction respectively. We are able to predict the exact memory and time usage with very high accuracy (median relative error $< 17\%$) for unseen matrices.

Figures 87 – 92 show the overlap of the top 10 recommendations with the true top

[1] http://www.cise.ufl.edu/research/sparse/matrices/Schenk_AFE/index.html.

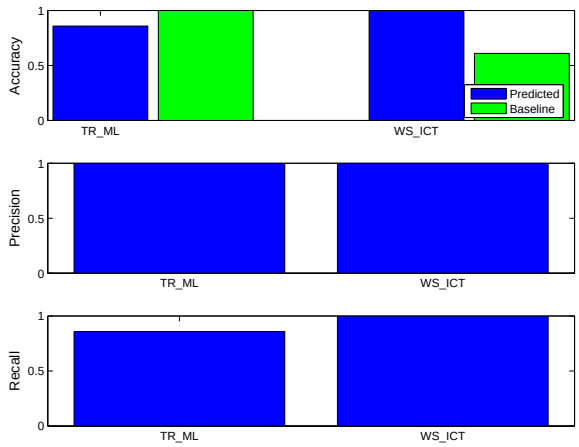


Fig. 82. Classification accuracy, precision, and recall for solvability prediction using the best solvability model on trials comprising of domain specific unseen matrices and solver configurations for Trilinos ML and WSMP ICT.

10 for the three unseen matrices in the case of Trilinos ML and WSMP ICT. In the case of Trilinos ML we observe that the overlap of the top 10 recommendations with that of the true top 10 recommendations is lower than in the case of WSMP ICT expecially in the case of memory. The corresponding performance benefits relative to the PIB and PSB choices are shown in the figures. In the case of Trilinos ML, the recommendations are very close to the PSB values for all the three matrices in the case of time and memory-time product. Even though the overlap of the top recommendations with respect to memory are low, the actual values do not vary much from PIB and PSB choices and the best choice is included in the top 4 recommendations. In the case of WSMP ICT, there are only slight variations in memory and memory time product values between the PIB and PSB choices. However, in the case of time, there is a significant difference in the PIB and PSB performance and the recommendations match the PSB values closely for a large fraction of the top 10 recommendations. These observations suggest that our recommendation approach is highly effective in specialized domains if there is sufficient training data to capture the general behavior of the various matrices.

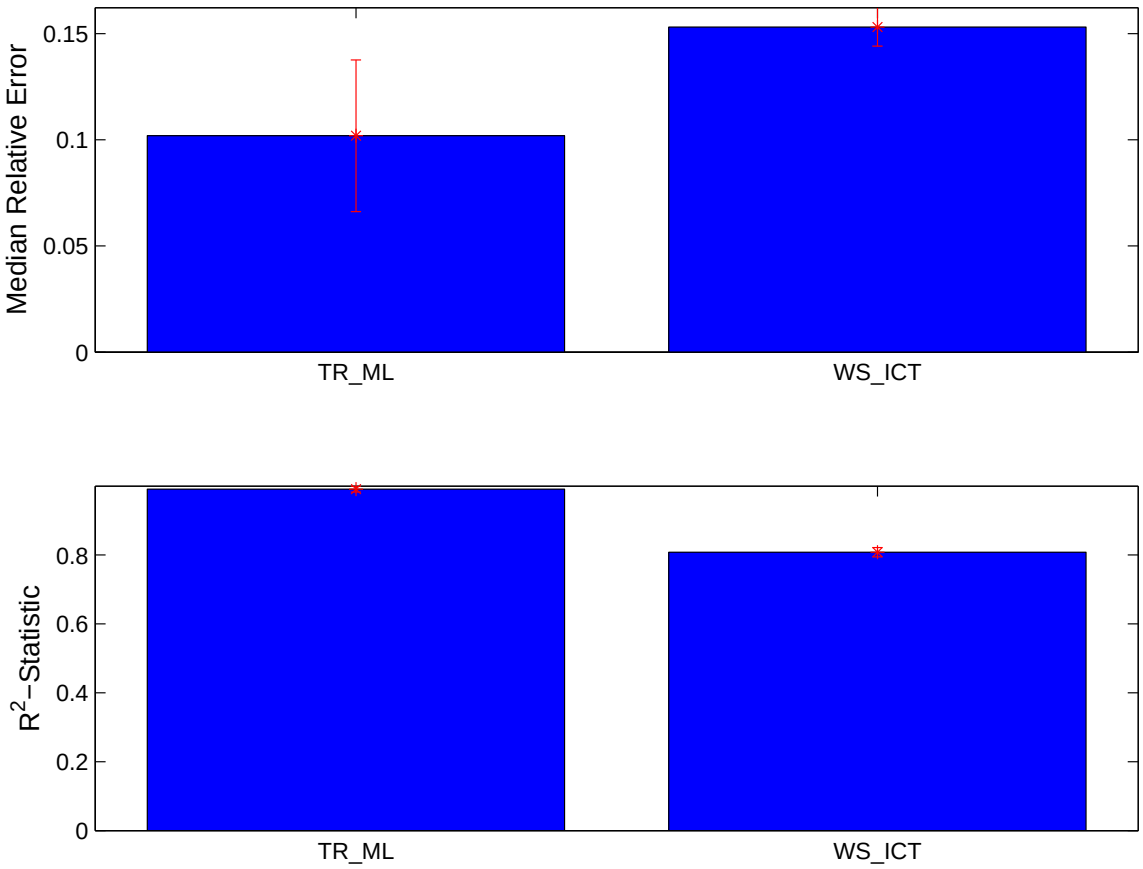


Fig. 83. Median relative error and R^2 statistic for memory prediction using the best memory model on trials comprising of domain specific unseen matrices and solver configurations for Trilinos ML and WSMP ICT.

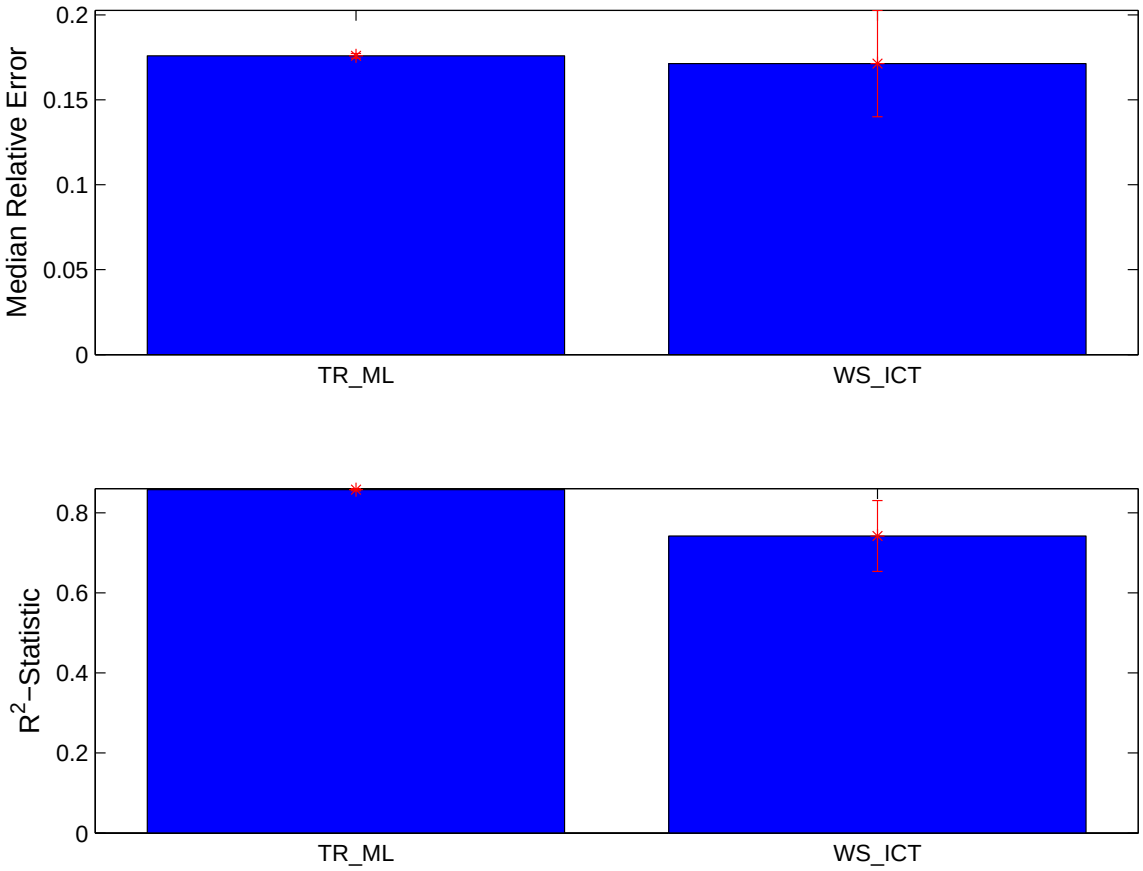


Fig. 84. Mean Relative error and R^2 statistic for time prediction using the best time model on trials comprising of domain specific unseen matrices and solver configurations for Trilinos ML and WSMP ICT.

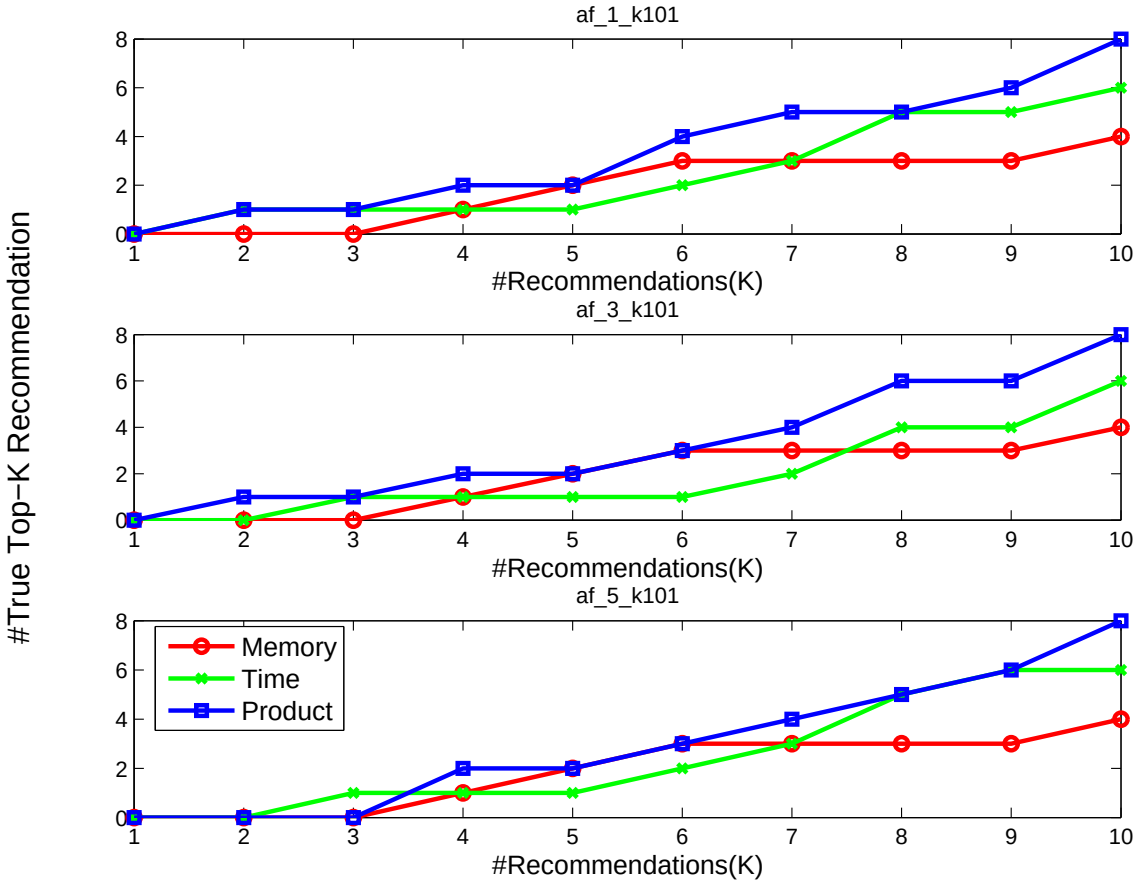


Fig. 85. The number of true best choices for memory, time, and memory-time product that is present in top-k recommendations for Trilinos ML (66 configurations) in the case of *af_1_k101*, *af_3_k101* and *af_5_k101* matrices.

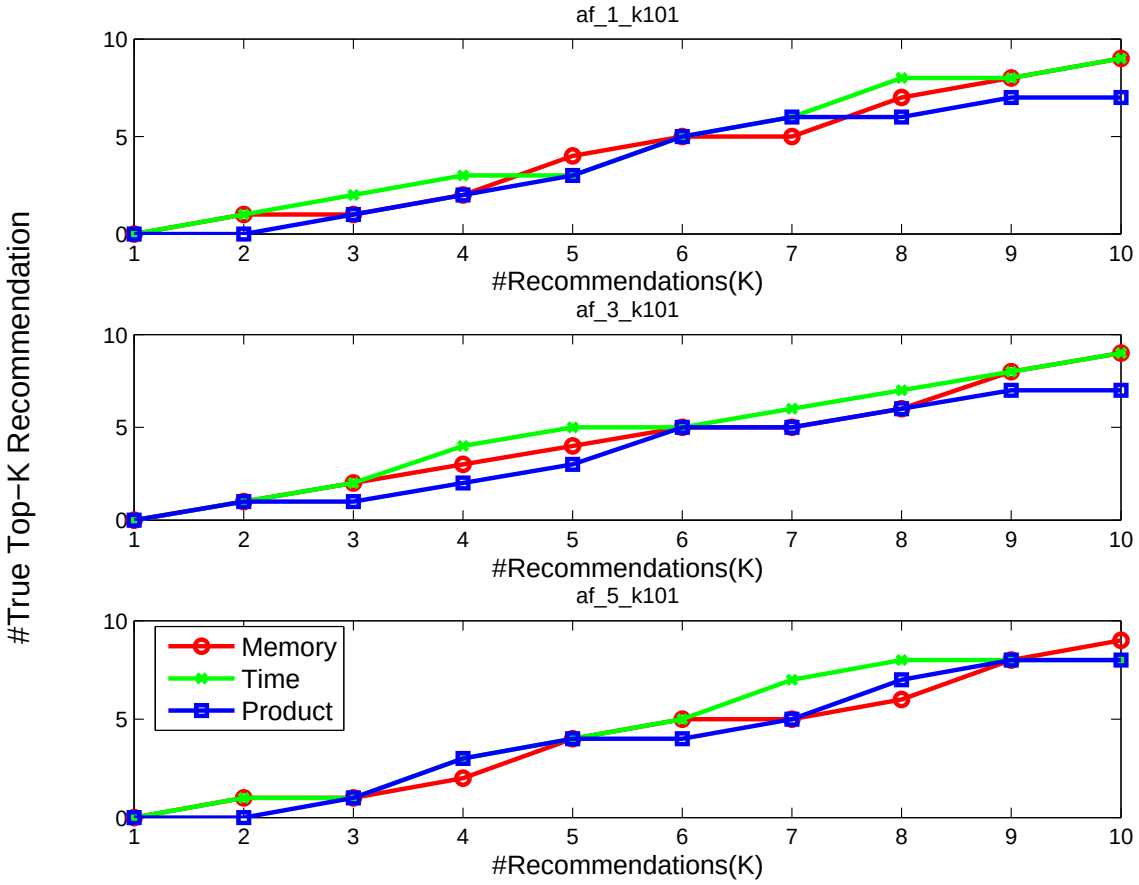


Fig. 86. The number of true best choices for memory, time, and memory-time product that is present in top-k recommendations for WSMP ICT (64 configurations) in the case of *af_1_k101*, *af_3_k101*, and *af_5_k101* matrices.

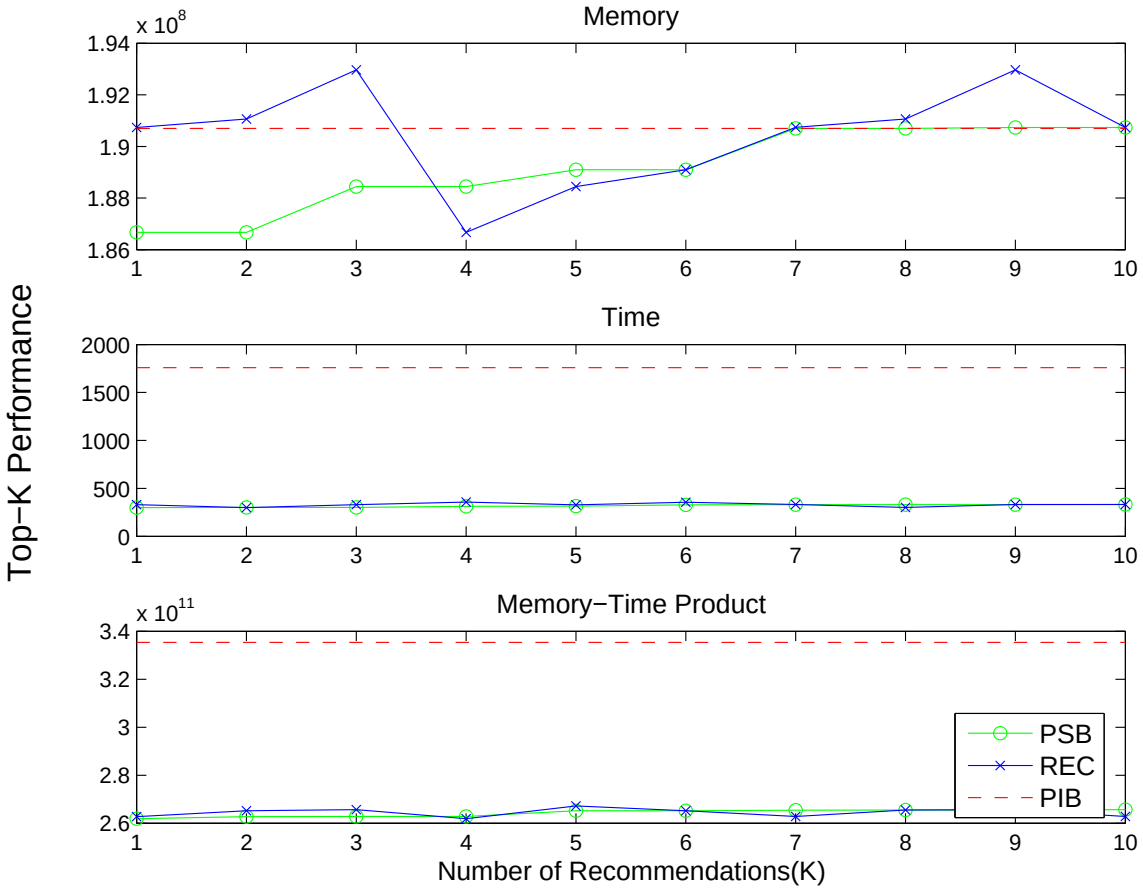


Fig. 87. Performance with respect to memory, time and memory-time product for the k^{th} recommended (REC) and problem specific best (PSB) Trilinos ML configurations for *af_1_k101*. Where applicable the problem independent best (PIB) performance values are also plotted.

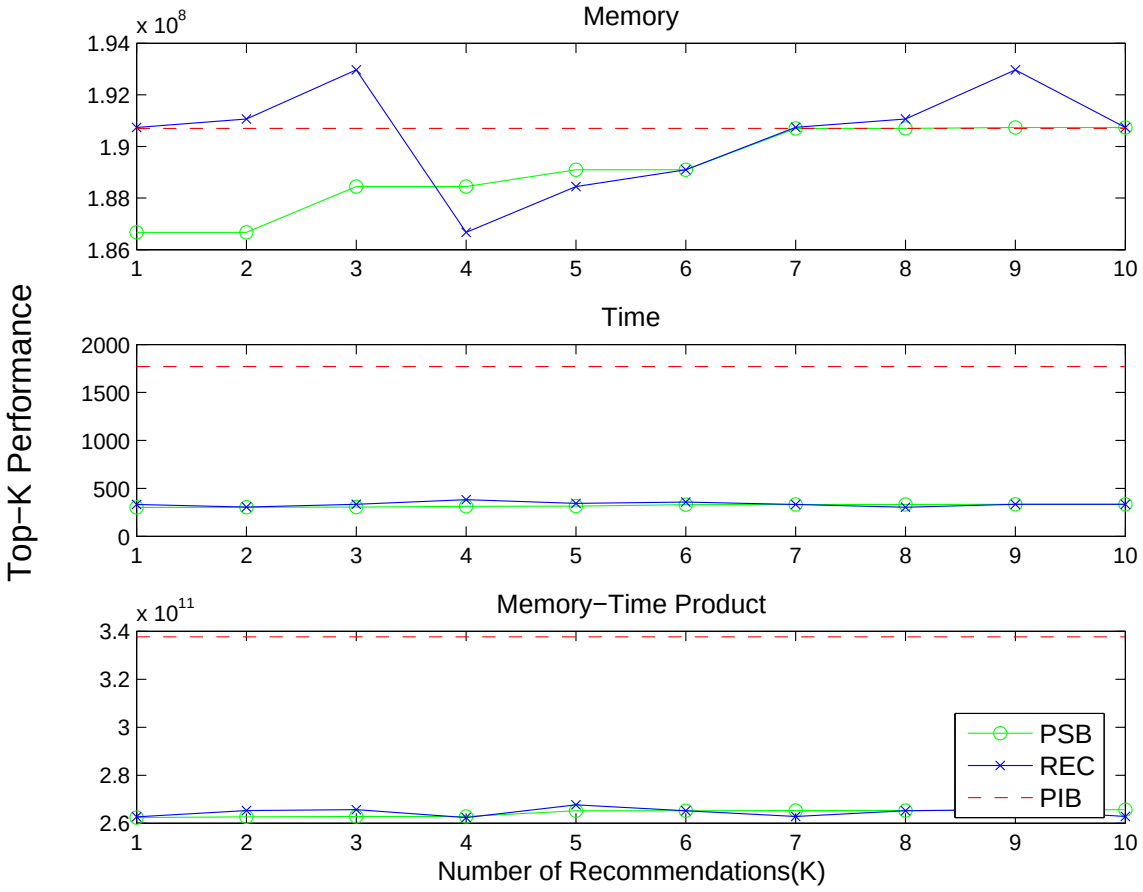


Fig. 88. Performance with respect to memory, time and memory-time product for the k^{th} recommended (REC) and problem specific best (PSB) Trilinos ML configurations for *af_3_k101*. Where applicable the problem independent best (PIB) performance values are also plotted.

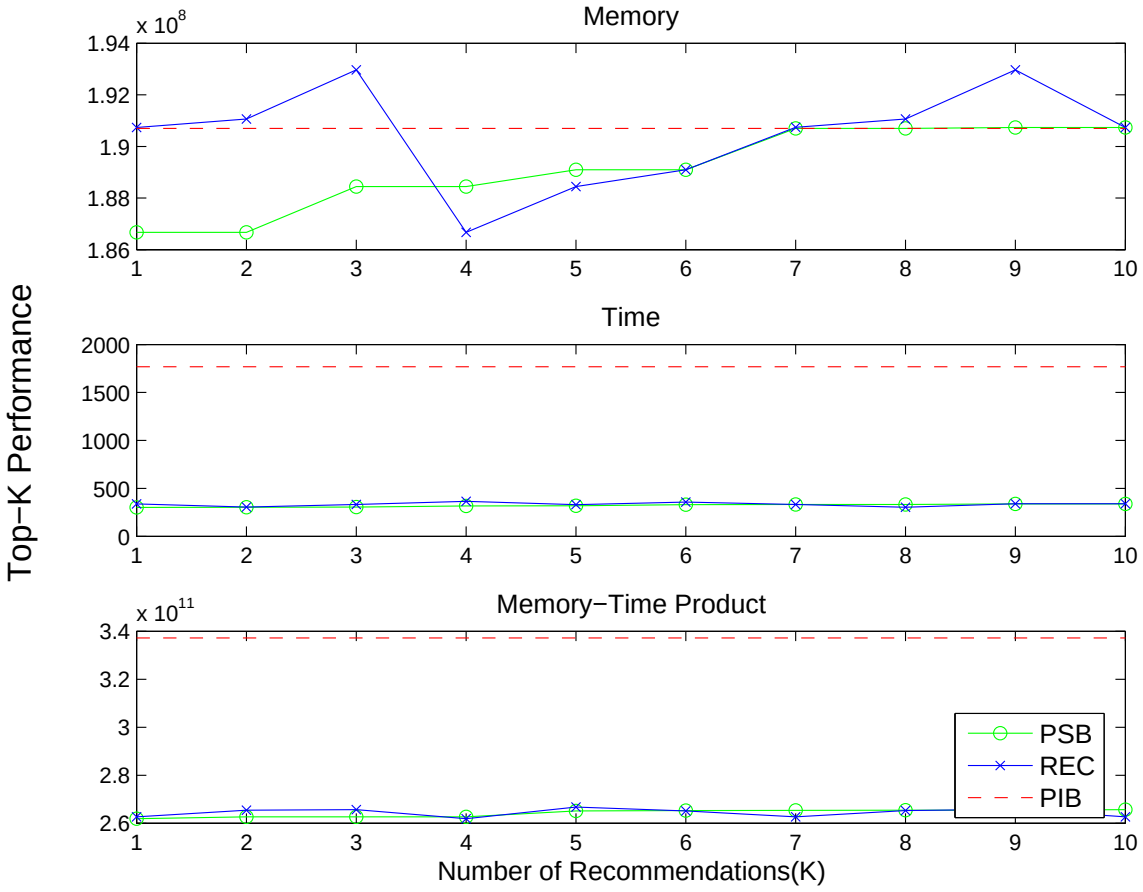


Fig. 89. Performance with respect to memory, time and memory-time product for the k^{th} recommended (REC) and problem specific best (PSB) Trilinos ML configurations for *af_5_k101*. Where applicable the problem independent best (PIB) performance values are also plotted.

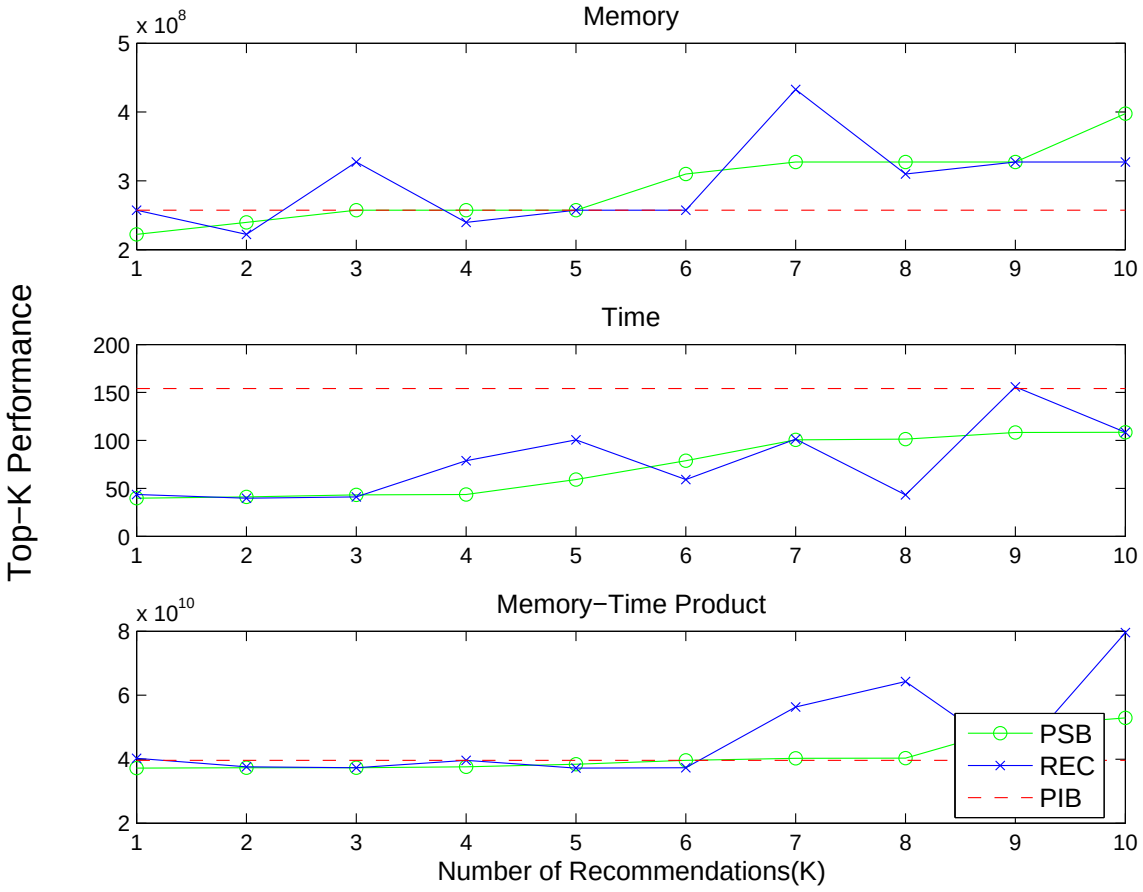


Fig. 90. Performance with respect to memory, time and memory-time product for the k^{th} recommended (REC) and problem specific best (PSB) WSMP ICT configurations for *af_1_k101*. Where applicable the problem independent best (PIB) performance values are also plotted.

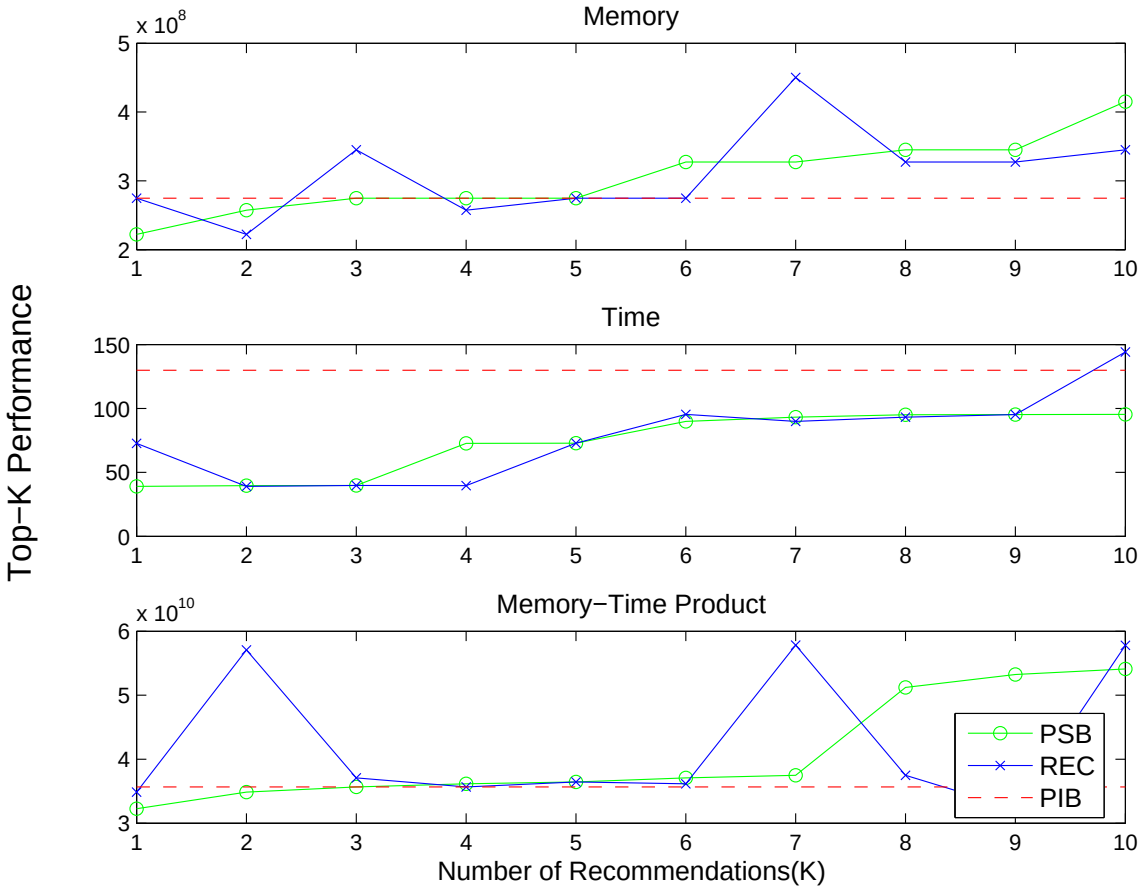


Fig. 91. Performance with respect to memory, time and memory-time product for the k^{th} recommended (REC) and problem specific best (PSB) WSMP ICT configurations for *af_3_k101*. Where applicable the problem independent best (PIB) performance values are also plotted.

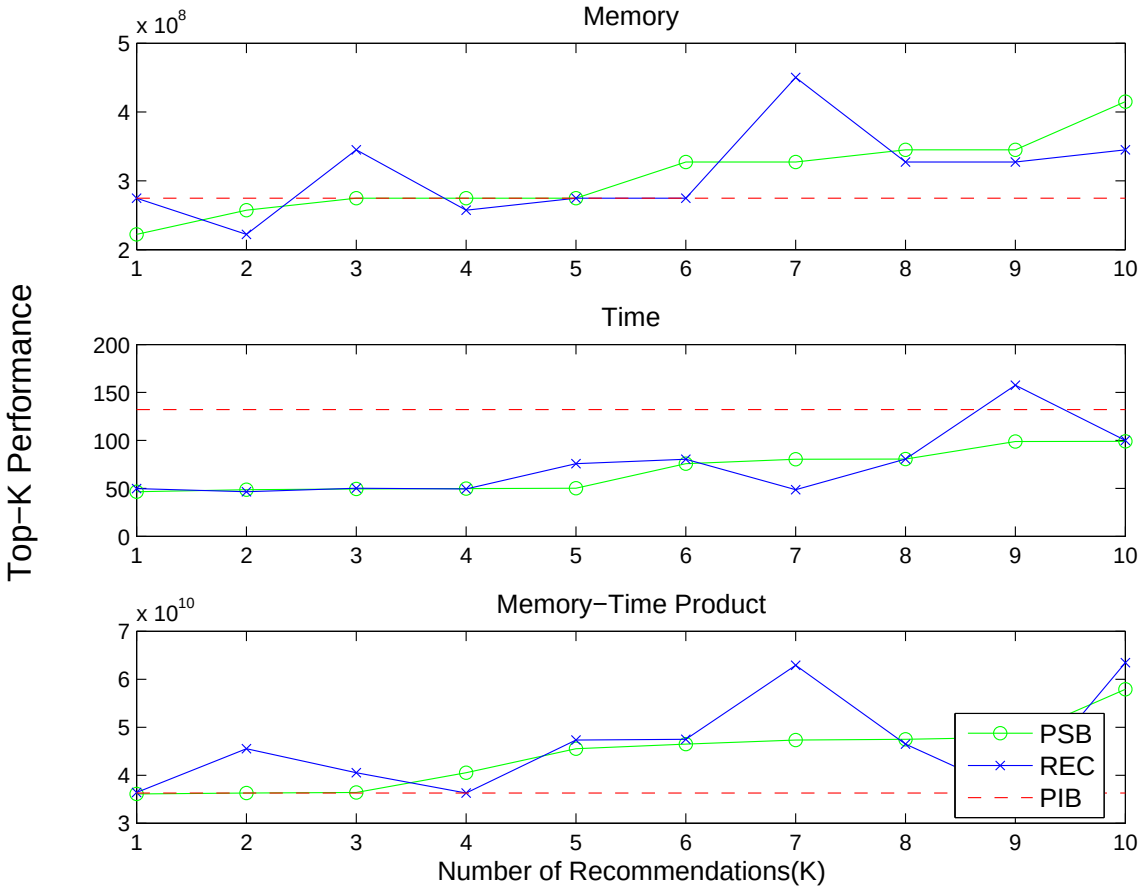


Fig. 92. Performance with respect to memory, time and memory-time product for the k^{th} recommended (REC) and problem specific best (PSB) WSMP ICT configurations for *af_5_k101*. Where applicable the problem independent best (PIB) performance values are also plotted.

CHAPTER VI

CONCLUSION AND FUTURE WORK

Solving complex linear systems lies at the heart of most scientific computing tasks and the increasing prevalence of large scale simulations with huge linear systems has made it extremely critical to develop intelligent strategies for selecting preconditioned iterative solver configurations. The current dissertation attempts to address this challenge by presenting a fairly general practitioner-centric framework both for problem-independent retrospective analysis as well as problem-specific predictive modeling of performance data. Empirical evaluation of the proposed approaches for iterative solver selection clearly demonstrates the potential benefits to practitioners.

A. Contributions

We now summarize the specific contributions of this dissertation.

Performance evaluation methodology: We introduce a principled methodology for a comparative performance evaluation of software options that is motivated by a typical user decision process. This approach addresses practical user concerns such as software failure, representation of software configurations, and parameter fine-tuning by introducing new relatively simple, but powerful metrics. Notable among these are the use of area under performance-profile curves to facilitate a credible ranking of the software options and use of conditional variance measures to identify influential parameters for sequential fine-tuning. We also outline how these metrics can be employed to (a) determine the "best" default configurations among a set of possible choices, (b) compare groups of software options in the presence and absence of fine-tuning along multiple performance criteria and (a) refine parameters to achieve

the desired performance improvements. Though targeted towards the iterative solver selection problem, this methodology has much broader scope and should be readily applicable to other software selection tasks.

Solver performance analysis infrastructure: To apply our performance evaluation methodology for the solver selection problem, we developed software infrastructure tools for the collection, analysis, and visualization of solver performance data. Given a user specified set of linear systems, this infrastructure conducts software trials and collects performance data (time, memory, error norm, etc.) for all combinations generated from the specified linear systems, a set of hardware configurations (number of CPUs and memory limits), and sets of values of various solver and preconditioner parameters. This is achieved via a data collection unit composed of both serial and parallel driver programs and associated scripts for some widely used solver packages. Subsequently, the analysis and reporting unit of the system performs various comparative and sensitivity analyses within and across pre-specified groups of solver configurations using the collected performance data. We intend to publish this software so that application scientists can use it to analyze solver performance with respect to matrices from specific domains that are of interest to them.

Extensive empirical evaluation: We also present performance evaluation of a suite of preconditioners based on incomplete factorization, sparse approximate inverse, and algebraic multilevel schemes available in packages such as PETSc, Trilinos, Hypre, ILUPACK, and WSMP. We compare the robustness, speed, and memory consumption of these preconditioners on a set of benchmark problems and also identify the influential parameters for each solver configuration group. For packages that provide support for parallel execution, we collect and present performance data on multiple processors. To the best of our knowledge, this study provides the most extensive

practitioner-centric empirical evaluation to date of a number of popular and promising general purpose preconditioners available in black-box solver packages, and can be highly beneficial to solver package users and developers.

Predictive performance modeling: We propose a novel multi-stage statistical approach for determining the "best" software option for a given problem with respect to some desired performance criteria that is based on learning predictive models from empirical performance data. The key novelty of our approach lies in our modular formulation that comprises of three sub problems: (a) solvability modeling, (b) performance modeling, and (c) performance optimization, which provides the flexibility to effectively target practical challenges such as software failure and multi-objective optimization using suitable classification, regression and rank-aggregation techniques. To be specific, the solvability model is used to filter out failure-prone configurations before modeling the performance. To accommodate optimization of multiple criteria, separate models are learned for each of the core performance statistics and then combined during the optimization step to identify the top choices. Our choice of instance space consisting of "trials" represented in terms of characteristics of the corresponding "problem", "software configuration" and their interactions, which allows us to directly model solvability and performance as response functions associated with trials, is also a distinguishing feature of our approach relative to existing work that contributes to an elegant formulation, and in fact, better predictions. Further, for the case where the performance models are based on generalized linear regression, we also propose a fast and efficient methodology for identifying the top-k solver choices with respect to various performance criteria (including hybrid combinations) using monotonic rank aggregation techniques over the software configuration feature space.

Iterative solver recommendation system: Lastly, we developed a prototype implementation of a modular self-learning recommendation system for iterative solvers with specialized components dedicated to data collection, feature generation, offline learning, and online recommendation. Using the available domain knowledge as well as solver performance data from our earlier study, we customized the recommendation system by identifying informative properties of the linear systems and iterative solver configurations as well as the data transformations needed to ensure good model fit. Evaluation of our approach on various performance datasets using models specific to a solver package (HYPRE) and various package-preconditioner combinations demonstrates that one can obtain very good quality recommendations that are often close to the ideal choices and better than even the problem-independent best solver configuration. The success of our approach indicates that there is a huge potential for optimizing scientific computing software using statistical models, especially in case of problems that require significant computational effort.

B. Future Work

The current work offers multiple avenues for future exploration:

Application to other software selection tasks. In the current dissertation, we only evaluated our performance analysis and recommendation methodology on iterative solvers, but it can be readily generalized to broader software selection scenarios such as optimization of scientific libraries for different architectures and problem domains. Effective customization of the proposed approaches would require a suitable representation of the space of problems as well as available software options.

Hierarchical representation for heterogeneous solver space. In our current performance analysis methodology, we represent solver configurations as a feature vector

based on constituent parameters. However, certain attributes are, often, meaningful only for a subset of choices, (e.g., restart values is an important parameter for GMRES, but not applicable in case of CG). Separate analysis of performance data from the homogeneous solvers subsets (e.g., package-preconditioner combinations) is often beneficial in such cases, but lack of sufficient data might lead to misleading conclusions and over-fitted models. A potential alternate strategy is to have a multi-level hierarchical representation of the solver space based on the common parameters. Using this representation, one can perform predictive or retrospective analysis with respect to the common parameters at any node in the hierarchy using the performance data that corresponds to all the configurations associated with that node, resulting in more robust results.

Sensitivity analysis of performance models. Studying the variation of the performance model predictions due to changes in the linear system and solver configuration features is important for validating the models and ensuring that one does not arrive at misleading conclusions due to outliers. In the current work, we perform a simplistic model analysis by computing the correlations and the corresponding p-values of the trial characteristics with respect to the response of the performance models and qualitatively verifying that these are compatible with expected behavior. A more rigorous sensitivity analysis involves computing either the partial derivatives or the conditional variance of the response with respect to each of the input features. As discussed in Chapter III, Section D.3, for linear models (including support vector regression with linear kernel), one can obtain a global sensitivity score for each input feature that is a function of the linear coefficient and variance associated the input feature and these can be compared with the empirical estimates. However, in case of SVR with non-linear kernels (e.g., RBF kernel), there is often a significant variation

in sensitivity depending on the value of the input feature and it is beneficial to study this non-linear dependence using the metrics presented in [42].

Optimization of continuous solver parameters. In our current recommendation approach, we assume that there are a finite number of solver configurations and perform ranking over them using the performance models to obtain the top choices. However, in practice, due to continuous parameters, the solver space might not be finite and the problem of choosing the best solver configurations can be posed in terms of an optimization problem over the solver parameter space with the objective function determined by the learned performance models and the specified problem. In case of linear models, this reduces to a mixed integer linear program.

Active collection of performance data. The current studies only involved a small benchmark dataset because of the difficulty in obtaining large complex matrices. However, in an industrial setting, where one has access to a large number of matrices, the computational effort associated with performing an empirical trial would prevent one from exploring all possible matrix-solver combinations. An active learning based approach for identifying potentially informative trials that reduce the uncertainty in the performance models would be extremely valuable in such a situation.

REFERENCES

[1] *Hypre, high performance preconditioners: Users manual*. Available online at *http://acts.nersc.gov/hypre/documents/HYPRE_usr_manual.ps*, 2006.

[2] G. ADOMAVICIUS AND A. TUZHILIN, *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions*, IEEE Transactions on Knowledge and Data Engineering, 17 (2005), pp. 734–749.

[3] D. AGARWAL AND S. MERUGU, *Predictive discrete latent factor models for large scale dyadic data*, in Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2007, pp. 26–35.

[4] S. BALAY, K. BUSCHELMAN, W. D. GROPP, D. KAUSHIK, M. G. KNEPLEY, ET AL., *Portable extensible toolkit for scientific computing web page*, 2001. http://www.mcs.anl.gov/petsc.

[5] A. BANERJEE, I. DHILLON, J. GHOSH, S. MERUGU, AND D. MODHA, *A generalized maximum entropy approach to Bregman co-clustering and matrix approximation*, Journal of Machine Learning Research, 8 (2007), pp. 1919–1986.

[6] R. BARRETT, M. BERRY, T. F. CHAN, J. DEMMEL, J. DONATO, ET AL., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, 1994.

[7] R. BARRETT, M. BERRY, J. DONGARRA, V. EIJKHOUT, AND C. ROMINE, *Algorithmic bombardment for the iterative solution of linear systems: A poly-iterative approach*, Journal of Computational and Applied Mathematics, 74 (1996), pp. 91–109.

[8] M. BENZI, *Preconditioning techniques for large linear systems: A survey*, Journal of Computational Physics, 182 (2002), pp. 418–477.

[9] M. BENZI AND M. TUMA, *A comparative study of sparse approximate inverse preconditioners*, Applied Numerical Mathematics, 30 (1999), pp. 305–340.

[10] S. BHOWMICK, V. EIJKHOUT, Y. FREUND, E. FUENTES, AND D. KEYES, *Application of machine learning to the selection of sparse linear solvers* , International Journal of High Performance Computing Applications. Under Review.

[11] S. BHOWMICK, P. RAGHAVAN, LOIS C. MCINNES, AND BOYANA NORRIS, *Faster PDE-based simulations using robust composite linear solvers*, Future Generation Computer Systems, 20 (2004), pp. 373–387.

[12] B. E. BOSER, I. GUYON, AND V. VAPNIK, *A training algorithm for optimal margin classifiers*, in Proceedings of the 5th Annual Conference on Computational Learning Theory, 1992, pp. 144–152.

[13] G. BOSILCA, Z. CHEN, J. DONGARRA, V. EIJKHOUT, G. E. FAGG, ET AL., *Self-adapting numerical software (SANS) effort*, IBM Journal of Research and Development, 50 (2006), pp. 223–238.

[14] R. BRAMLEY, D. GANNON, T. STUCKEY, J. VILLACIS, J. BALASUBRAMANIAN, ET AL., *The Linear System Analyzer*, Tech. Report TR-511, Indiana University, Bloomington, IN, 1998.

[15] C. J. C. BURGES, *A tutorial on support vector machines for pattern recognition*, Data Mining and Knowledge Discovery, 2 (1998), pp. 121–167.

[16] C. CHANG AND C. LIN, *LIBSVM: A library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[17] E. CHOW, *Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns*, International Journal of High Performance Computing Applications, 15 (2001), pp. 56–74.

[18] E. CHOW AND Y. SAAD, *Experimental study of ILU preconditioners for indefinite matrices*, Journal of Computational and Applied Mathematics, 86 (1997), pp. 387–414.

[19] E. CHOW AND Y. SAAD, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM Journal on Scientific Computing, 19 (1998), pp. 995–1023.

[20] C. CORTES AND V. VAPNIK, *Support-vector networks*, Machine Learning, 20 (1995), pp. 273–297.

[21] E. CUTHILL AND J. MCKEE, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings of the 24th ACM National Conference, 1969, pp. 157–172.

[22] T. A. DAVIS, *The University of Florida sparse matrix collection*, Jan 2007. Available at *http://www.cise.ufl.edu/research/sparse/matrices*.

[23] J. W. DEMMEL, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[24] E. D. DOLAN AND J. J. MORÉ, *Benchmarking optimization software with performance profiles*, Mathematical Programming, 91 (2002), pp. 201–213.

[25] J. DONGARRA AND A. BUTTARI, *Freely available software for linear algebra on the web*, 2009. *http://www.netlib.org/utk/people/JackDongarra/la-sw.html*.

[26] H. DRUCKER, C. J. C. BURGES, L. KAUFMAN, A. J. SMOLA, AND V. VAPNIK, *Support vector regression machines*, in Proceedings of the 9th Annual Conference on Neural Information Processing Systems, 1996, pp. 155–161.

[27] R. FAGIN, A. LOTEM, AND M. NAOR, *Optimal aggregation algorithms for middleware*, Journal of Computer and System Sciences, 66 (2003), pp. 614–656.

[28] R. FREUND, G. GOLUB, AND N. NACHTIGAL, *Iterative solution of linear systems*, Acta Numerica, (1992), pp. 57–100.

[29] Y. FREUND AND L. MASON, *The alternating decision tree learning algorithm*, in Proceedings of the 16th International Conference on Machine Learning, 1999, pp. 124–133.

[30] M.W. GEE, C.M. SIEFERT, J.J. HU, R.S. TUMINARO, AND M.G. SALA, *ML 5.0 Smoothed Aggregation User's Guide*, Tech. Report SAND2006-2649, Sandia National Laboratories, 2006.

[31] A. GEORGE AND J. W. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall Professional Technical Reference, Englewood Cliffs, NJ, 1981.

[32] T. GEORGE, A. GUPTA, AND V. SARIN, *An experimental evaluation of iterative solvers for large SPD systems of linear equations*, in Proceedings of the 10th Copper Mountain Conference on Iterative Methods, 2008.

[33] T. GEORGE, A. GUPTA, AND V. SARIN, *An Empirical Analysis of Iterative Solver Performance for SPD Systems*, Tech. Report RC 24737, IBM T. J. Watson Research Center, Yorktown Heights, NY, 2009.

[34] J. R. GILBERT AND S. TOLEDO, *An assessment of incomplete-LU preconditioners for nonsymmetric linear systems*, Informatica, 24 (2000), pp. 409–425.

[35] A. GREENBAUM, V. PTAK, AND Z. STRAKOUS, *Any non-increasing convergence curve is possible for GMRES*, SIAM Journal of Matrix Analysis and Applications, 17 (1996), pp. 465–469.

[36] A. GUPTA, *Fast and effective algorithms for graph partitioning and sparse-matrix ordering*, IBM Journal of Research and Development, 41 (1997), pp. 171–184.

[37] A. GUPTA, *WSMP: Watson Sparse Matrix Package (Part-I: Direct Solution of Symmetric Sparse Systems)*, Tech. Report RC 21886, IBM T. J. Watson Research Center, Yorktown Heights, NY, 2000.

[38] A. GUPTA, *WSMP: Watson Sparse Matrix Package (Part-III: Iterative Solution of Sparse Systems)*, Tech. Report RC 24398, IBM T. J. Watson Research Center, Yorktown Heights, NY, 2007.

[39] A. GUPTA AND T. GEORGE, *Adaptive techniques for improving the performance of incomplete factorization preconditioning*, SIAM Journal on Scientific Computing. To appear.

[40] I. GUYON AND A. ELISSEEFF, *An introduction to variable and feature selection*, Journal of Machine Learning Research, 3 (2003), pp. 1157–1182.

[41] I. GUYON, J. WESTON, S. BARNHILL, AND V. VAPNIK, *Gene selection for cancer classification using support vector machines*, Machine Learning, 46 (2002), pp. 389–422.

[42] I. GUYON, J. WESTON, S. BARNHILL, AND V. VAPNIK, *Gene selection for cancer classification using support vector machines*, Machine Learning, 46 (2002), pp. 389–422.

[43] V. E. HENSON AND U. M. YANG, *BoomerAMG: A parallel algebraic multigrid solver and preconditioner*, Applied Numerical Mathematics: Transactions of International Association for Mathematics and Computers in Simulation, 41 (2002), pp. 155–177.

[44] J. L. HERLOCKER, J. A. KONSTAN, A. BORCHERS, AND J. RIEDL, *An algorithmic framework for performing collaborative filtering*, in Proceedings of the 22nd International ACM SIGIR Conference on Information Retrieval, 1999, pp. 230–237.

[45] M. A. HEROUX, R. A. BARTLETT, V. E. HOWLE, R. J. HOEKSTRA, J. J. HU, ET AL., *An overview of the Trilinos project*, ACM Transactions on Mathematical Software, 31 (2005), pp. 397–423.

[46] T. HOFMANN, *Latent semantic models for collaborative filtering*, ACM Transactions on Mathematical Software, 22 (2004), pp. 89–115.

[47] P. W. HOLLAND AND R.E. WELSCH, *Robust regression using iteratively reweighted least-squares*, Communications in Statistics: Theory and Methods, A6 (1977), pp. 813–827.

[48] G. HOLMES, A. DONKIN, AND I. H. WITTEN, *WEKA: A machine learning workbench*, in Proceedings of the 2nd Australian and New Zealand Conference on Intelligient Information Systems, 1994, pp. 357–361.

[49] E. N. HOUSTIS, A. C. CATLIN, J. R. RICE, V. S. VERYKIOS, N. RAMAKRISHNAN, AND C. E. HOUSTIS, *PYTHIA-II: A knowledge/database system for managing performance data and recommending scientific software*, ACM Transactions on Mathematical Software, 26 (2000), pp. 227–253.

[50] E. N. HOUSTIS AND J. R. RICE, *Future Problem Solving Environments for Computational Science*, Purdue University, West Lafayette, IN, 2002.

[51] D. HYSOM AND A. POTHEN, *A scalable parallel algorithm for incomplete factor preconditioning*, SIAM Journal of Scientific Computing, 22 (2000), pp. 2194–2215.

[52] T. JOACHIMS, *Training linear svms in linear time*, in Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006, pp. 217–226.

[53] M. T. JONES AND P. E. PLASSMANN, *An improved incomplete Cholesky factorization*, ACM Transactions on Mathematical Software, 21 (1995), pp. 5–17.

[54] G. KARYPIS AND V. KUMAR, *Parallel multilevel k-way partitioning scheme for irregular graphs*, in Proceedings of the 8th ACM/IEEE Conference on Supercomputing, 1996, p. 35.

[55] E. KUEFLER AND T.-YI CHEN, *On using reinforcement learning to solve sparse linear systems*, in Proceedings of the 8th International Conference on Computational Science, vol. 5101, 2008, pp. 955–964.

[56] D. D. LEE AND H. S. SEUNG, *Algorithms for non-negative matrix factorization*, in Proceedings of the 13th Annual Conference on Neural Information Processing Systems, 2000, pp. 556–562.

[57] M. BOLLHÖFER, Y. SAAD AND O. SCHENK, *ILUPACK - preconditioning software package*. Available online at *http://www.math.tu-berlin.de/ilupack*, 2006.

[58] S. C. MADEIRA AND A. L. OLIVEIRA, *Biclustering algorithms for biological data analysis: A survey*, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 1 (2004), pp. 24–45.

[59] P. MCCULLAGH AND J. NELDER, *Generalized Linear Models, 2nd edition*, Chapman & Hall/CRC, Boca Raton, FL, 1989.

[60] T. M. MITCHELL, *Machine Learning*, McGraw-Hill Higher Education, Columbus, OH, 1997.

[61] M. NACHTIGAL, S. C. REDDY, AND L. N. TREFETHEN, *How fast are nonsymmetric matrix iterations*, SIAM Journal on Matrix Analysis and Applications, 13 (1992), pp. 778–795.

[62] R. O.DUDA, P.E. HART, AND D. G. STORK, *Pattern Classification (2nd edition)*, Wiley-Interscience, New York, NY, 2000.

[63] A. POPESCUL, L. H. UNGAR, D. M. PENNOCK, AND S. LAWRENCE, *Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments*, in Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence, 2001, pp. 437–444.

[64] J. R. QUINLAN, *Induction of decision trees*, Machine Learning, 1 (1986), pp. 81–106.

[65] J. R. QUINLAN, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1993.

[66] J. R. QUINLAN, *Bagging, boosting, and C4.5*, in Proceedings of the 13th National Conference on Artificial Intelligence, vol. 1, 1996, pp. 725–730.

[67] N. RAMAKRISHNAN AND A. GRAMA, *Mining scientific data*, Advances in Computers, 55 (2001), pp. 119–169.

[68] N. RAMAKRISHNAN AND C. J. RIBBENS, *Mining and visualizing recommendation spaces for elliptic PDEs with continuous attributes*, ACM Transactions on Mathematical Software, 26 (2000), pp. 254–273.

[69] J. RENNIE AND N. SREBRO, *Fast maximum margin matrix factorization for collaborative prediction*, in Proceedings of the 22nd International Conference on Machine Learning, 2005, pp. 713–719.

[70] J.W. RUGE AND K. STÜBEN, *Multigrid methods*, 3 (1987), pp. 73–130. Frontiers in Applied Mathematics.

[71] Y. SAAD, *Iterative Methods for Sparse Linear Systems, 2nd edition*, SIAM, Philadelphia, PA, 2003.

[72] Y. SAAD AND B. SUCHOMEL, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Numerical Linear Algebra with Applications, 9 (2002), pp. 359–378.

[73] Y. SAAD AND H. A. VAN DER VORST, *Iterative solution of linear systems in the 20th century*, Journal of Computational and Applied Mathematics, 123 (2000), pp. 1–33.

[74] M. SALA AND M. HEROUX, *Robust Algebraic Preconditioners with IFPACK 3.0*, Tech. Report SAND-0662, Sandia National Laboratories, Albuquerque, NM, 2005.

[75] A. SALTELLI, M. RATTO, T. ANDRES, F. CAMPOLONGO, J. CARIBONI, ET AL., *Global Sensitivity Analysis: The Primer*, WileyBlackwell, West Sussex, UK, 2008.

[76] A. SALTELLI AND S. TARANTOLA, *On the relative importance of input factors in mathematical models: Safety assessment for nuclear waste disposal*, Journal of the American Statistical Association, 97 (2002), pp. 702–709.

[77] A. J. SMOLA AND B. SCHÖLKOPF, *A tutorial on support vector regression*, Statistics and Computing, 14 (2004), pp. 199–222.

[78] *Spider: General purpose machine learning toolbox in matlab (version 1.7)*. Available at http://www.kyb.tuebingen.mpg.de/bs/people/spider, 2006.

[79] R. S. TUMINARO AND C. TONG, *Parallel smoothed aggregation multigrid : Aggregation strategies on massively parallel machines*, in Proceedings of the 12th ACM/IEEE conference on Supercomputing, 2000, p. 5.

[80] P. VANEK, *Acceleration of convergence of a two level algorithm by smooth transfer operators*, Applied Mathematics, 37 (1992), pp. 265 –274.

[81] V. N. VAPNIK, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, NY, 1999.

[82] R. S. VARGA, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.

[83] V. S. VERYKIOS, E. N. HOUSTIS, AND J. R. RICE, *A knowledge discovery methodology for the performance evaluation of scientific software*, Neural, Parallel and Scientific Computations, 8 (2000), pp. 115–132.

[84] D. C. WILSON, D. B. LEAKE, AND R. BRAMLEY, *Case-based recommender components for scientific problem-solving environments*, in Proceedings of the 16th International Association for Mathematics and Computers in Simulation World Congress, 2000.

[85] S. XU, E. LEE, AND J. ZHANG, *Designing and building an intelligent preconditioner recommendation system (a progress report)*, in Proceedings of International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Scientific and Industrial Applications, 2003.

[86] S. XU, E. J. LEE, AND J. ZHANG, *An Interim Analysis Report on Preconditioners and Matrices*, Tech. Report 388–03, University of Kentucky, Lexington, KY, 2004.

[87] S. XU AND J. ZHANG, *A Data Mining Approach to Matrix Preconditioning Problem*, Tech. Report 433-05, University of Kentucky, Lexington, KY, 2005.

VITA

Thomas George was born in Trivandrum, India in 1976. He completed his Bachelor of Technology in mechanical engineering from Indian Institute of Technology (IIT), Madras, India in May 1999. He joined the graduate program in the Engineering Research Center (ERC) at Mississippi State University, Starkville, Mississippi in Spring 2000 and earned his Master of Science degree in Computational Engineering in December 2001. He continued to work at ERC as a full time research associate working on several NSF and NASA funded projects till he joined the PhD program in Computer Science at Texas A&M University in the Fall of 2003.

During his stay at Texas A&M University, Thomas George worked as a Graduate Research Assistant for Dr. Vivek Sarin and Graduate Teaching Assistant for Dr. Joseph Hurley and Dr. Teresa Leyk. His research interests are in scientific computing, parallel applications, and scientific data mining. He also published several articles in international journals and conferences such as: Journal of Functional Programming, SIAM Journal of Scientific Computing, Supercomputing, and International Conference on Data Mining among others. His address is: IBM India Research Lab, 4 Block C, Institutional Area, Vasant Kunj, New Delhi, 110070, India.