RC24600 (W0807-048) July 9, 2008
Computer Science

IBM Research Report

A Recommendation System for Preconditioned Iterative Solvers

Thomas George
Department of Computer Science
Texas A&M University
College Station, TX 77843
tgeorge@cs.tamu.edu

Anshul Gupta
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
anshul@watson.ibm.com

Vivek Sarin
Department of Computer Science
Texas A&M University
College Station, TX 77843
sarin@cs.tamu.edu

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties). Copies may be requested from IBM T. J. Watson Research Center , P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home .

A Recommendation System for Preconditioned Iterative Solvers

Thomas George
Dept. of Computer Science
Texas A&M University
tgeorge@cs.tamu.edu

Anshul Gupta
Mathematical Sciences Dept.
IBM T. J. Watson Research Center
anshul@watson.ibm.com

Vivek Sarin
Dept. of Computer Science
Texas A&M University
sarin@cs.tamu.edu

Abstract

Preconditioned iterative methods are often used to solve very large sparse systems of linear systems that arise in many science and engineering applications. The performance and robustness of these solvers is extremely sensitive to the choices of multiple preconditioner and solver parameters. Users of iterative methods often encounter an overwhelming number of combinations of choices for solvers, matrix preprocessing steps, preconditioners, and their parameters. The lack of a unified theoretical analysis of preconditioners coupled with limited knowledge of their interaction with linear systems makes it highly challenging for practitioners to choose good solver configurations. In this paper, we propose a novel, multi-stage learning based methodology for determining the best solver configurations to optimize the desired performance behavior for any given linear system. Our solver configuration recommendation system involves three steps of modeling, namely (a) solvability modeling, (b) performance modeling, and (c) performance optimization. We model solvability and performance metrics as response functions associated with dyads of linear systems and solver configurations using classification and regression techniques. The best solver choices are then determined using a fast and efficient technique that performs rank aggregation over the learned performance models. Empirical results over real performance data for the Hypre *iterative solver package demonstrates the efficacy and flexibility of the proposed approach.*

1. Introduction

A fundamental step in many scientific and engineering applications is the solution of a system of linear equations of the form $Ax = b$, where A is the sparse coefficient matrix, b is the right hand side vector, and x is the vector of unknowns. Iterative methods have the potential to solve these large systems with memory and time costs that are almost linear in the size of A. However, in practice, they are often fraught with problems such as poor or no convergence and poor CPU utilization rates. Therefore, in practice, iterative solvers almost always use preconditioners to improve their performance and robustness. A preconditioner broadly refers to an explicit or implicit scheme that modifies the original linear system (e.g., $M^{-1}Ax = M^{-1}b$) such that it is easier to solve using an iterative method while still consuming moderate amount of resources. For example, an important class of preconditioners is based on the incomplete factorization $M = LU$, where L and U are significantly sparse approximations to the actual lower and upper triangular factors of A. The effectiveness of incomplete factorization is highly dependent on the manner in which the nonzero entries in L and U are selected. Other promising preconditioners include those based on directly computing approximate inverses of A (e.g., ParaSails [10]) and those using algebraic multigrid methods (e.g., BoomerAMG [19]). In addition to preconditioning, certain preprocessing steps such as diagonal scaling and re-ordering the linear system are also known to help in the case of harder problems [5]. Empirical studies [9, 15, 16] indicate that fine-tuning the preconditioner parameters and matrix preprocessing options is critical for ensuring convergence as well as for providing significant benefits in terms of computation time, memory usage, and accuracy of the solution.

Choosing the best preprocessing options and 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. The diversity of the preconditioners makes it difficult to analyze them in a unified theoretical model. The large number of tunable parameters, both discrete and continuous, associated with each preconditioner along with their mutual interaction effects further exacerbates this situation. There is also often a significant variability in the different implementations for the same preconditioner, which limits the utility of a purely theoretical analysis. The enormous computational resources required for solving large linear systems make it extremely expensive for practitioners to adopt a simple trial and error strategy for 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 highly desirable to have automated tools for recommending the most suitable solver configuration(s) based on the linear system properties and user requirements.

The solver recommendation problem is somewhat analogous to a product recommendation problem with solver configurations and linear systems corresponding to products and users, respectively. The main focus of the solver recommender is to optimize the expected performance metrics while that of a product recommender is to optimize user-preference ratings. However, a few key differences make the solver selection problem more challenging. Solver failure and the large range of possible values for performance metrics (e.g., seconds to hours) are critical issues to be addressed during any statistical modeling of solver performance metrics. Simple models used in most product recommendation systems, where the user preference ratings typically lie in a small fixed range (e.g., 1–10), would not suffice. The main potential 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. The huge requirements of memory and time resources also severely limit the collection of such empirical data entailing learning from highly sparse data.

In this paper, we propose a novel multi-stage learning based methodology for determining the "best" solver configuration(s) with respect to some desired performance criteria for any given sparse linear system. Our work addresses the challenges described above and makes the following main contributions:

1. We propose a formulation of the solver recommendation problem in terms of three key subproblems: (a) solvability modeling, (b) performance modeling, and (c) performance optimization. This allows us to readily address issues arising from solver failure as well as multi-objective optimization. Specifically, the solvability model is used to filter out failure-prone configurations before modeling the performance. Further, to accommodate optimization of multiple criteria, we separately learn models for each of the core performance statistics. The optimization step involves combining the learned performance models to identify the top solver choices for the specified performance criteria.

2. We model both solvability and performance metrics as response functions associated with trials, which are pairs of linear systems and solver configurations. The goal is to predict the unknown response values in terms of the trial characteristics gleaned from domain knowledge as well as the observed response values from empirical data. We achieve this by using standard classification and regression techniques.

3. Assuming generalized linear models for the core performance statistics, we propose a fast and efficient methodology for identifying the top-k solver choices for multiplicative combinations of the core performance statistics using monotonic rank aggregation techniques such as the Fagin's threshold algorithm [14].

4. We describe a prototype implementation for a modular self-learning solver recommendation system with specialized components dedicated to data collection, feature generation, offline learning, and online recommendation. Using this system, we evaluate the key aspects of our proposed approach on performance data obtained using *Hypre*, an iterative solver package with a diverse set of preconditioners.

The remainder of the paper is organized as follows. We describe the solver recommendation problem in more detail and provide a formal definition in section 2. In section 3, we discuss our multi-step learning based approach as well as the main algorithmic components. Section 4 provides an overview of the solver recommendation system. In section 5, we provide a detailed empirical evaluation of the proposed recommendation approach on a real data set. Section 6 contains a discussion of related work followed by conclusion and directions for future work in section 7.

2. Solver Recommendation Problem

In this section, we present some background of the iterative solver selection problem and provide formal definitions of each of the three subproblems in our formulation.

2.1. Background

Linear System Features	Memory	Time	Failure
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

Table 2.1. 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: 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. Table 2.1 lists the set of features of matrix A 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 2.1 suggest that these features have a correlation with the performance metrics that is significantly different from zero.

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." Table 2.2 contains a list of the solver features that we used in our experiments. The total number of feasible combinations of preconditioner parameters, orderings, preconditioners, and solvers in Table 2.2 is 317.

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.

2.2. 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 ith 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 subproblems in our formulation.

Solvability Prediction: Since iterative solvers are known to have a high rate of failure [9, 15, 16] and the performance metrics obtained for failed trials can often be misleading, an important first requirement is to predict and filter out the unsuccessful 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.

Performance Estimation: A desirable feature of a solver recommender is to provide performance estimates of a solver configuration for a particular linear system. In general, for a given linear system, there is no single solver configuration that performs best with respect to time, memory, and accuracy. A fast solver configuration may consume significantly more memory than slower configurations, and vice versa. A practitioner in this case might prefer a solver that performs reasonably with respect to the memory-time product or some other hybrid additive and/or multiplicative combinations. The second subproblem 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.

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

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, 0.005, -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

Table 2.2. Description of the components of solver configuration.

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 rth 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 combination of linear system and solver configuration.

3. Multi-Stage Learning Approach

In this section, we describe the key algorithmic components of our approach for addressing the three subproblems described in section 2.2.

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. Hence, a natural choice for trial features includes the attributes of the linear system and solver configuration along with the product interactions [24]. 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 [18] 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 [4, 21]. 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 [4] 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. Algorithm 1 shows the detailed steps.

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 high variance in the modeling process. To handle this problem, we normalize the actual metrics by the performance of a specific default solver. Since it is also desirable to have better sensitivity for lower performance values, we also 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.

Algorithm 1 Solvability Modeling

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 $\hat{s}$

To model the performance metrics, we again use three types of approaches just as in the case of solvability modeling. The first option involves directly learning each performance model from the raw trial features using standard regression techniques. The second one involves unsupervised matrix-imputation techniques that ignore the trial features. The third alternative is to adopt a hybrid approach. For our empirical evaluation, we pick a representative technique (linear regression, Euclidean co-clustering [4] and a variant of Gaussian-PDLF [3]) for each of these three classes and study all of them. Our variant of Gaussian-PDLF algorithm follows the hybrid solvability modeling approach outlined in Algorithm 1. 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.

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 one 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 rth performance metric. The $\exp(\cdot)$ 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 a simple aggregation over the features, i.e., $g(\mathbf{z}_{ij}) = \prod_r (z_{ij}^{(r)})^{\alpha_r} \Rightarrow g(\hat{\mathbf{z}}_{ij}) = \exp\left((\sum_r \alpha_r \boldsymbol{\beta}_r)^T \mathbf{u}_{ij}\right)$. Since 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, $g(\hat{\mathbf{z}}_{ij}) = \exp(\boldsymbol{\delta}_i^T \mathbf{v}_j)$, where $\mathbf{v}_j$ 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 $\mathbf{v}_j$ 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 [14], 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. Algorithm 2 shows the detailed steps.

4. Recommendation System Prototype

In this section, we provide an overview of a modular self-learning solver recommender system based on the multi-step approach described in Sec 3. We have implemented this system using a combination of Matlab code and drivers in C. Figure 4.1 shows the various components of the proposed system and their interactions. The functional units

Algorithm 2 Top-k Performance Ranking

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 2.3

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 pth solver feature $(1 \leq p \leq P)$

Compute feature coefficients and sign for specified linear system

Choose $\boldsymbol{\delta}_i$ s.t. $\boldsymbol{\delta}_i^T \mathbf{y}_j = (\sum_r \alpha_r \boldsymbol{\beta}_r)^T \mathbf{u}_{ij}$

$\mathbf{w}_{ip} \leftarrow sign(\delta_{ip}),\ \delta_{ip} \leftarrow |\delta_{ip}|,\ \ (1 \leq p \leq P)$

Initialize candidate solver configuration set

$C_M \leftarrow \emptyset$

repeat

Access top element in the sorted feature lists in the appropriate direction

$M^{(p)} \leftarrow pop(L_p, w_{ip})\ \ (1 \leq p \leq P)$

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 $g(A_i, :) \leq \tau$

return top k list of C_M in terms of $g(A_i, :)$

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

The *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 single CPU of an IBM HPC cluster 1600 based on 1.9 GHz Power5+ processors. The *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. The *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 using classification algorithms such as decision trees and support vector machines as well as multiple feature selection techniques based on information gain, L1 norm, etc. The *Performance Modeling Unit* learns predictive models for the performance metrics of interest in the solvable re-

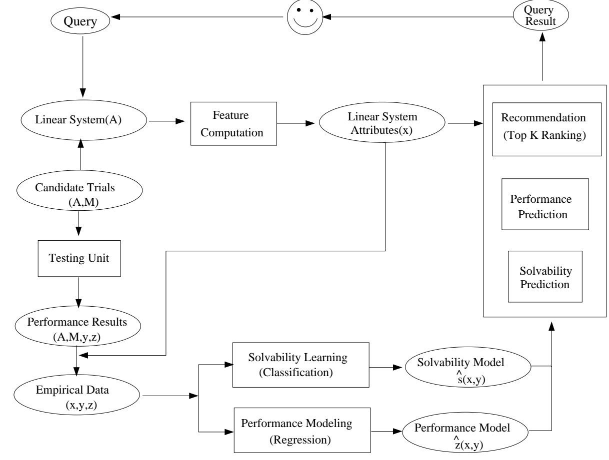

Figure 4.1. Recommender System Overview

gion, 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 Matlab tool kit [27]. The *Solvability Prediction Unit* predicts whether a given linear system can be solved using a particular solver configuration using the learned solvability model. The *Performance Prediction Unit* provides predictions on the expected performance for user specified combinations of linear systems and solver configurations using the learned performance model. The *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.

5. Experimental Evaluation

In this section, we present empirical evaluation of the various aspects of our solver recommendation approach.

Performance Data Set: For our empirical study, we considered 317 solver configurations drawn from a publicly available iterative solver package Hypre (release version 2.0.0) [1] that has implementations of multiple general purpose preconditioners. Based on the description of the solver options in Table 2.2, we identified 15 attributes such as solver, restart, preconditioner, level of fill, drop tolerance etc. Using the above solver configurations, we generated performance data on a test set of large sparse SPD matrices obtained mostly from the University of Florida sparse matrix collection [11].

For each trial, we obtained the memory usage, time taken, relative error norm and also recorded solver failure where applicable.

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 a 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) [28], (2) decision tree (J48) [24], and (3) K-nearest neighbor (KNN) [24]. 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 5.2 shows the misclassification 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 misclassification error (45.6% using the majority classification). Including the interaction features leads to even better classification accuracy. Figure 5.3 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 ICK and ILUT preconditioners as well as many SAI and AMG based solver configurations while the first one contains matrices that could not be solved by the ICK and ILUT preconditioners, but were solved by most configurations of SAI and AMG 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 5.3 lists the top 5 interaction features that were selected for classification.

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 profile curves [12] 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 [15].

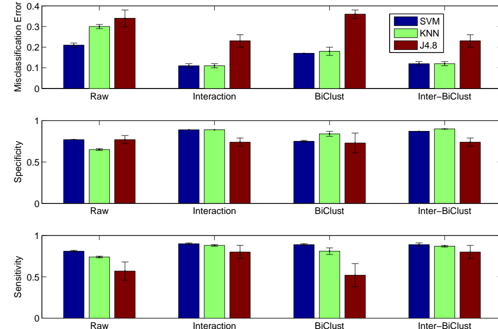

Figure 5.2. Classification error, sensitivity and specificity on test set for solvability prediction for SVM, KNN and J48 classifiers on a trial with 20% training split averaged over 5 runs.

Classification	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

Table 5.3. 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.

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.

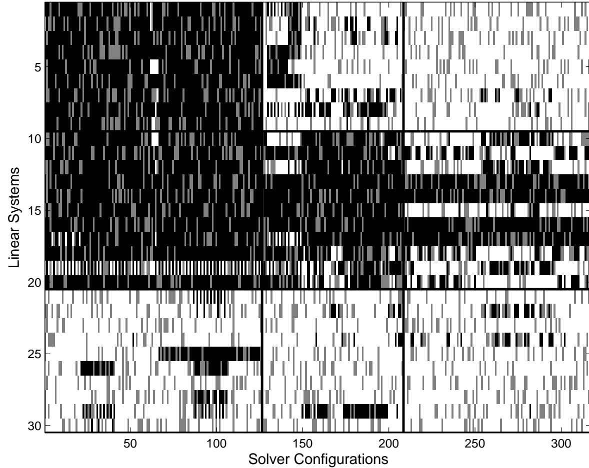

Figure 5.3. Linear system-solver configuration bi-cluster for solvability. Black indicates solver failure, white indicates solver successes and grey indicates missing values.

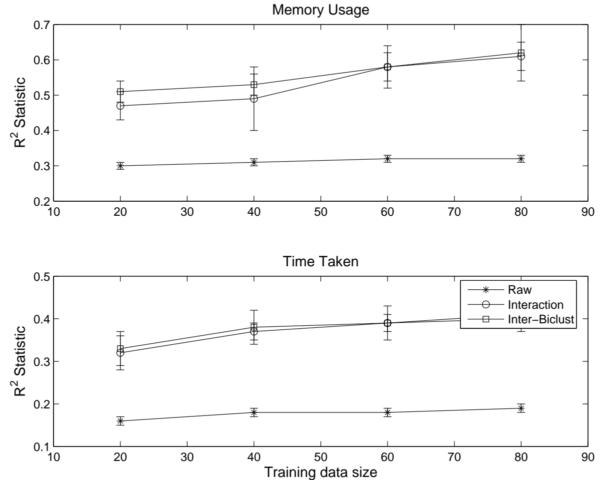

Figure 5.4. R^2 statistic for memory and time prediction with varying training data size averaged over 5 runs for multiple feature sets.

Figure 5.4 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 5.3 shows the top 5 predictive interaction features for memory and time, respectively.

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 default choice in terms of average quality value of the top-k recommendations.

Figure 5.5 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. For comparison, the expected overlap with the actual top-25 with random selection is $< 8\%$. Figure 5.6 (a-c) shows the performance improvement that can be obtained using the generated recommendations over the default choice. In this case, the default solver configurations (dotted lined) were chosen based on the overall best performance on the entire test suite using performance profile areas [15]. The ideal 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 default choice for all the three criteria and fairly close to the ideal curve. The recommendations for memory-time product indicate that our approach can be quite effective even for optimizing a hybrid performance criterion.

6. Related Work

In this section, we briefly discuss how the proposed solver recommendation approach is related to existing literature on preconditioned iterative solvers, expert systems for choosing scientific software, and machine learning based recommendation systems.

Adaptive Iterative Solvers: Over the past few years, there have been a number of empirical studies [9, 15, 16] on iterative solvers that highlight the importance of problem specific fine tuning for improving solver performance. This has prompted research on adaptive solvers [7, 17] where the solver parameters are dynamically chosen at the beginning

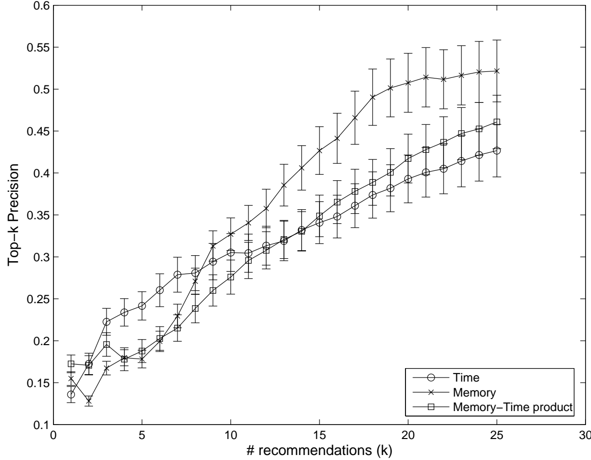

Figure 5.5. Fraction of the true best choices for memory, time and memory-time product that is present in top-k recommendations.

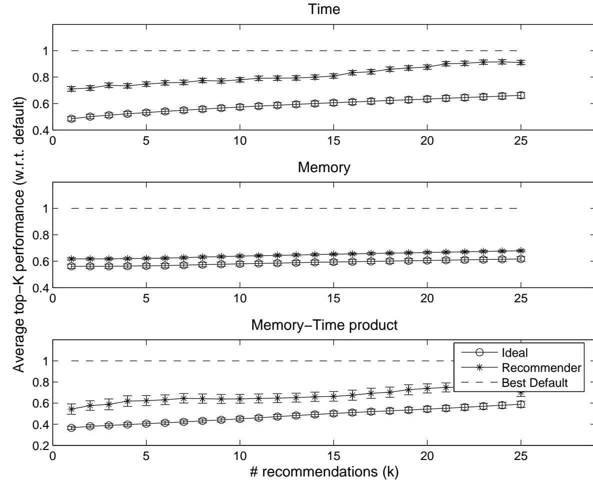

Figure 5.6. Average improvement in the memory, time and memory-time product due to fine-tuning over that of the default choice for multiple values of k.

of each iteration based on the characteristics of the linear system as it changes during the iterative solution process. Recent empirical studies [15] indicate that adaptive solutions are highly effective.

Expert systems for software selection: Our proposed approach is closely related to existing techniques on employing data mining for software selection. One of the early influential works in this area is the recommendation portal PYTHIA-II [22], which provides users with the data management infrastructure and data mining tools to make suitable software choices. In recent years, the emergence of problem solving environments (PSEs) for analyzing linear systems [8] has resulted in a renewed interest in developing intelligent systems for recommending solvers/preconditioners [6, 13, 29]. 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 [6] 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 [23] attempts to use reinforcement learning for solvability prediction, however, with limited success in obtaining good results in comparison to more expensive supervised learning techniques. The use of linear system and solver configuration interaction features as well as estimating and optimizing actual and hybrid combinations of performance metrics is a unique contribution of our work.

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 [4, 20, 26] 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, 25] 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. Our current work maps the solver selection problem to the abstract recommendation scenario and employ the state-of-the-art learning techniques for estimating solvability and performance values, while paying attention to practical aspects such as the variability in the performance values, feature selection as well as the final application goals.

7. Conclusion and Future Work

We have proposed a new and effective solver recommendation approach that takes into account user requirements and attributes of the linear systems. Our methodology effectively addresses key challenges specific to the solver recommendation setting such as robustness to solver failures and multiple performance criteria by a novel decomposition of the recommendation problem into three independent subproblems. The first two subproblems are solved by adapting existing classification/regression techniques whereas, for the top-k ranking problem we propose

a fast and efficient solution based on Fagin's threshold algorithm. Empirical results indicate that our approach can provide highly accurate predictions for solvability (89%) as well as performance estimates (reduction in quadratic loss of 61% for memory and 41% for time). The suggested top-25 recommendations are significantly better than the default and has reasonable overlap with the actual top-25 configurations (53% for memory and 43% for time).

Our current study also highlights new directions of research that could be explored in future. Specifically, we find that the performance predictions tend to be more accurate and the features are more interpretable when limited to homogeneous groups of solver configurations. Hence, a hierarchical representation of the solver configuration space and a suitable modeling might yield better recommendations even if we were to incorporate multiple iterative solver package implementations as well as the direct solver. The high acquisition cost for performance data as well as linear system features also points to the need for a more economical active learning strategy for generating empirical data that balances the information gained and the acquisition costs.

References

[1] Hypre, High Performance Preconditioners. https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html.

[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 TKDE*, 17(6):734–749, 2005.

[3] D. Agarwal and S. Merugu. Predictive discrete latent factor models for large scale dyadic data. In *KDD*, pages 26–35, 2007.

[4] A. Banerjee, I. Dhillon, J. Ghosh, S. Merugu, and D. Modha. A generalized maximum entropy approach to Bregman co-clustering and matrix approximation. *JMLR*, 8:1919–1986, 2007.

[5] M. Benzi. Preconditioning Techniques for Large Linear Systems: A Survey. *JCP*, 182(2):418–477, 2002.

[6] S. Bhowmick, V. Eijkhout, Y. Freund, E. Fuentes, and D. Keyes. Application of Machine Learning to the Selection of Sparse Linear Solvers . *IJHPCA, Submitted Sept.*, 2006.

[7] S. Bhowmick, P. Raghavan, L. C. McInnes, and B. Norris. Faster PDE-based simulations using robust composite linear solvers. *Future Gen. Comp. Syst.*, 20(3):373–387, 2004.

[8] R. Bramley et al. The Linear System Analyzer. Technical Report TR-511, C S Dept, Indiana University., 1998.

[9] E. Chow and Y. Saad. Experimental Study of ILU Preconditioners for Indefinite Matrices. *Journal of Comp. and App. Math.*, 86(2):387–414, 1997.

[10] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM Journal on Scientific Computing*, 19(3):995–1023, 1998.

[11] T. A. Davis. The University of Florida Sparse Matrix Collection. Technical report, CS Dept, University of Florida, Jan 2007.

[12] E. D. Dolan and J. J. Moré. Benchmarking Optimization Software with Performance Profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[13] J. Dongarra et al. Self-adapting Numerical Software (SANS) Effort. *IBM J. Res. Dev.*, 50(2/3):223–238, 2006.

[14] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[15] T. George, A. Gupta, and V. Sarin. An Experimental Evaluation of Iterative Solvers for Large SPD Systems of Linear Equations. In *10th Copper Mountain Conference on Iterative Methods*, 2008.

[16] J. R. Gilbert and S. Toledo. An assessment of incomplete-LU preconditioners for nonsymmetric linear systems. *Informatica*, 24:409–425, 2000.

[17] A. Gupta. WSMP: Watson sparse matrix package (Part-III: iterative solution of sparse systems). Technical Report RC 24398, IBM T. J. Watson Research Center, 2007.

[18] I. Guyon and A. Elisseeff. An Introduction to Variable and Feature Selection. *JMLR*, 3:1157–1182, 2003.

[19] V. E. Henson and U. M. Yang. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics: Transactions of IMACS*, 41(1):155–177, 2002.

[20] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *SIGIR*, pages 230–237, 1999.

[21] T. Hofmann. Latent semantic models for collaborative filtering. *ACM TOMS*, 22(1):89–115, 2004.

[22] 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. *TOMS*, 26(2):227–253, 2000.

[23] E. Kuefler and T.-Y. Chen. On using reinforcement learning to solve sparse linear systems. to appear in proceedings of the international conference on computational science. In *LNCS*, volume 5101, pages 955–964, 2008.

[24] T. M. Mitchell. *Machine Learning*. McGraw-Hill Higher Education, 1997.

[25] 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 *UAI*, pages 437–444, 2001.

[26] J. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *ICML*, pages 713–719, 2005.

[27] Spider: General Purpose Machine Learning Toolbox in Matlab. http://www.kyb.tuebingen.mpg.de/bs/people/spider.

[28] V. N. Vapnik. *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer, November 1999.

[29] S. Xu, E. J. Lee, and J. Zhang. An Interim Analysis Report on Preconditioners and Matrices. Technical report, University of Kentucky, Lexington, 2004.