Automating Sparse Linear Solver Selection with Lighthouse

Kanika Sood, Boyana Norris
Computer and Information Science
University of Oregon
Eugene, OR 97403
[kanikas,norris]@cs.uoregon.edu

Pate Motter, Elizabeth Jessup
Department of Computer Science
University of Colorado
Boulder, CO 80309
[pate.motter,elizabeth.jessup]@colorado.edu

1. INTRODUCTION

Solving large sparse linear systems is a fundamental problem in high performance scientific and engineering computing. Application developers face the challenge of predicting which solver will converge fastest or at all for a given linear system.

The time to solve different linear systems depends on various factors. The main challenge is to predict which solution approach performs best for a given linear system. We apply several supervised machine learning (ML) techniques to help make this decision based on relatively few, easily computable properties of the input linear system. We achieve up to 93% accuracy.

2. BACKGROUND AND RELATED WORK

There have been several attempts to enable automated solver selection and configuration. None of them has produced usable software infrastructure that enables users to apply the methods for their own applications. Hence, our goal is to produce an extensible methodology for classifying algorithms and software that applies as solvers evolve.

We classify solvers based on linear system features and select the solver configuration that performs best on an extensive training set of problems.

Researchers have previously used ML to identify *good* solvers in the context of parallel nonlinear PDE solution [7]. This ML framework was successful in identifying solvers with expensive preconditioners (such as BoomerAMG) should be used as well as identifying when very simple solvers (such as Jacobi) would suffice. The success of this limited-domain exploration motivated the work presented in this poster, which is not limited to a particular domain and considers a greater number of ML methods.

Lighthouse [4] is the first framework that offers an organized taxonomy of software components for linear algebra. It currently offers support for sequential dense linear algebra computations provided by LAPACK [1] and for sequential and parallel sparse linear algebra computations provided by PETSc [2] and SLEPc [3]. We are in the process of adding the Trilinos [5] parallel sparse linear algebra library as well.

3. APPROACH

Creating the Training Dataset: The sparse linear systems used are derived from input matrices (1,015 matrices for PETSc and 1,429 matrices for Trilinos) in the University of Florida Sparse Matrix Collection [6] with unit right-hand sides. We used 154 and 72 solver and preconditioners configurations to solve these systems with PETSc and Trilinos, respectively.

Training Data Preparation: The data was converted into Weka's [9] Attribute-Relation File Format. Each data point includes a list of feature values, the solver identifier unique to each pair of Krylov method and preconditioner, and a label. We labeled each solver as *good* or *bad* based on its performance compared to best performing method.

Feature Computation: We used Anamod [8] to extract 68 features of the coefficient matrices. We also computed a 38-feature set with Trilinos.

Feature Set Reduction: We reduced the number of features by using Weka's RemoveUseless filter. We completed the feature selection with Weka by combining five attribute evaluators with two search methods.

Solver Classification: We used Weka to compare the performance of several classification algorithms: Bayes Net, k-nearest neighbor, Alternate Decision Trees, Random Forests, J48 , VFI Support Vector Machines, Decision Stump, and LADtree.

ML methods comparison: We measured the prediction accuracy by using the confusion matrix produced by Weka We compared the performance of the ML methods in terms of their sensitivity and the cost of building the classifier.

4. RESULTS

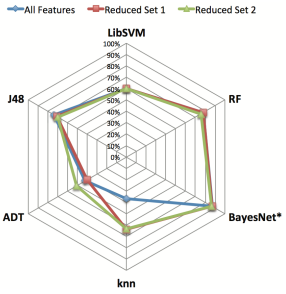
Figure 1 summarizes the best solver configurations we identified along with their occurrence rate indicating as to how often that configuration was used. For PETSc, the figure shows that there is no clear winner among the various combinations available. For PETSc solvers, we consider sequential runs, while the Trilinos results reflect small-scale (12 MPI tasks) parallel runs. The classification was performed on an Intel Core i5 MacBook Pro.

4.1 PETSc Solver Analysis

We collected solver performance information with PETSc version 3.5.3 on a Blue Gene/Q supercomputer. BayesNet showed the best performance in all cases; 87.6% using all

PETSc		Trilinos	
3.8%	LSQR, Jacobi	17.2%	Hybrid Block GMRES, Chebyshev
3.6%	BiCG, ASM(0)	9.4%	Hybrid Block GMRES, Diagonal
3.4%	BCGS, ILU(0)	8.8%	Hybrid Block GMRES, Jacobi
3.2%	GMRES, ASM(2)	3.9%	TFQMR, Chebyshev
3.0%	BCGS, ASM(0)	3.7%	Hybrid Block GMRES, ILUT
2.8%	BICG, BJacobi	2.8%	Hybrid Block GMRES, RILUK
2.3%	Chebyshev, Jacobi	2.8%	Block CG, Jacobi
2.2%	BCGS, ASM(3)	2.7%	Block CG, Diagonal
2.1%	iBCGS, ASM(2)	2.4%	Pseudoblock CG, Jacobi
2.0%	CG, ASM(0)	2.4%	TFQMR, Jacobi

Figure 1: Top 10 "good" solvers for PETSc and Trilinos and their occurrence rate.



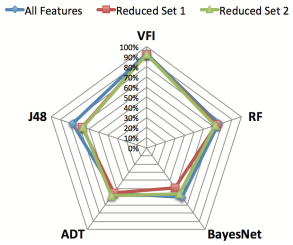
Method	T_{All}	T_{RS1}	T_{RS2}
LibSVM	6.88	1.94	1.79
RF	2.67	2.40	1.43
BayesNet	0.10	0.02	0.01
knn	0.001	0.001	0.001
ADTree	0.74	0.18	0.09
J48	0.24	0.19	0.06

(a) Accuracy and timing

Feature name	Reduced Feature Set 1 ($RS1$)	Reduced Feature Set 2 ($RS2$)
avg-diag-dist	X	X
nnz	X	
norm1	X	X
col-variability	X	X
min-nnzeros-per-row	X	X
row-variability	X	X
n-nonzero-diags	X	
kappa	X	X

(b) Reduced feature sets

Figure 2: PETSc solver classification and classifier build time (in seconds) with Anamod features.



Method	T_{All}	T_{RS1}	T_{RS2}
VFI	0.09	0.02	0.02
RF	14.33	6.83	6.09
BayesNet	0.30	0.08	0.07
ADTree	5.83	1.41	0.77
J48	1.05	0.26	0.18

(a) Accuracy and timing

Feature name	Reduced Feature Set 1 ($RS1$)	Reduced Feature Set 2 ($RS2$)
Dummy rows	X	X
Trace	X	X
Column diagonal dominance	X	X
Row diagonal dominance	X	X
Diagonal nonzeros	X	
Diagonal mean	X	X
Total DNE %	X	X

(b) Reduced Feature Sets

Figure 3: Trilinos solver classification and classifier build time (in seconds) with Trilinos features.

68 features, 86.91% with Reduced Feature Set 1 (RS1) and 86.4 % with Reduced Feature Set 2 (RS2) . RS1 and RS2 are shown in figure 2b. The performance of other ML methods can be seen in figure 2a along with the time taken to build the classifiers. We tested more methods than are included in the figure, but none performed better than the best method shown in the figure.

4.2 Trilinos Solver Analysis

Trilinos experiments were performed using the Janus supercomputer at the University of Colorado. Figure 3a shows the accuracy and time taken to build the classifiers for Trilinos with all Trilinos features.

The best TPR was shown by VFI in all three cases, 93 % with all features, 92% for RS1 and 92.5% for RS2. Figure 3b shows the reduced feature sets.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (NSF) awards CCF-1219089, CCF-155063 and CCF-1550202. It used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. It also used the Janus supercomputer, which is supported by NSF award CNS-0821794 and the University of Colorado Boulder.

5. REFERENCES

[1] LAPACK - Linear Algebra PACKage. http://www.netlib.org/lapack/, 2015.

[2] Portable, Extensible Toolkit for Scientific Computation (PETSc). http://www.mcs.anl.gov/petsc/, 2015.

[3] Scalable Library for Eigenvalue Problem Computations (SLEPc). http://www.grycap.upv.es/slepc/, 2015.

[4] The Lighthouse Project. http://lighthousehpc.github.io/Uolighthouse/, 2015.

[5] The Trilinos Project. http://trilinos.sandia.gov/, 2015.

[6] The University of Florida Sparse Matrix Collection. http://www.cise.ufl.edu/research/sparse/matrices/, 2015.

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

[8] V. Eijkhout and E. Fuentes. A proposed standard for matrix metadata. Technical Report ICL-UT 03-02, University of Tennessee, 2003.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11, 2009.