Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 1523 – 1532

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

Dynamic linear solver selection for transient simulations using multi-label classifiers

Paul R. Eller, Jing-Ru C. Cheng, Robert S. Maier

Information Technology Laboratory
Engineer Research and Development Center, Vicksburg, MS

Abstract

Many transient simulations spend a significant portion of the overall runtime solving a linear system. A wide variety of preconditioned linear solvers have been developed to quickly and accurately solve different types of linear systems, each having options to customize the preconditioned solver for a given linear system. Transient simulations may produce significantly different linear systems as the simulation progresses due to special events occurring that make the linear systems more difficult to solve or move the model closer to a state of equilibrium with easier to solve linear systems.

Machine learning algorithms provide the ability to dynamically select the preconditioned linear solver for each linear system produced by a simulation. We test both single-label and multi-label classifiers, demonstrating that multi-label classifiers achieve the best performance due to associating multiple fast linear solvers with each tested linear system. For more difficult simulations, these classifiers produce significant speedups, while for less difficult simulations these classifiers achieve performance similar to the fastest single preconditioned linear solvers. We test classifiers generated using limited attribute sets, demonstrating that we can minimize overhead while still obtaining fast, accurate simulations.

Keywords: machine learning, linear solvers, multi-label classifiers, ADH, MULAN, WEKA

1. Introduction

Many numerical models use transient simulations to accurately model how natural or manmade systems change over time and how different events or designs may affect these systems. For many transient simulations, the largest amount of running time is spent solving a linear system. Many preconditioners and solvers have been developed to quickly solve different types of linear systems. As the linear systems produced by the transient simulations change, the best preconditioned solver to solve each linear system also changes. Using the best preconditioned solver at each point in the simulation will allow us to get the lowest possible running times.

Machine learning algorithms provide the ability to generate predictive models, allowing us to create classifiers capable of taking a set of linear system attributes as input and outputting a preconditioned linear solver as the output.

Email addresses: Paul.R.Eller@usace.army.mil (Paul R. Eller), Ruth.C.Cheng@usace.army.mil (Jing-Ru C. Cheng), Robert.S.Maier@usace.army.mil (Robert S. Maier)

1877-0509 © 2012 Published by Elsevier Ltd.
doi:10.1016/j.procs.2012.04.167

We test both single-label classifiers that associate a single fast linear solver with each linear system and multi-label classifiers that associate multiple fast linear solvers with each linear system.

We can generate databases by computing attributes for each linear system, physical attributes for the transient simulation, computational attributes, and running times for a set of preconditioned solvers on each linear system. Machine learning algorithms can then use these databases to generate classifiers capable of dynamically selecting a preconditioned solver for each linear system given a set of attributes. This allows us to use different preconditioned solvers throughout the simulation and provides the potential to produce speedups in comparison with using a single preconditioned solver for an entire simulation.

2. Related Work

Previous studies have used machine learning algorithms to predict the best solver for a given linear system. Bhowmick et al. 2006 [1] use alternating decision trees and boosting methods to generate single-label classifiers to choose the best linear solver for a driven cavity flow model and a three-dimensional plasma simulation code. Bhowmick et al. 2009 [2] expand on this work by evaluating the performance of multiple single-label classifiers with matrices from a sparse matrix collection. Holloway and Chen [3] used neural networks to predict if a combination of preconditioner and iterative method will correctly solve a given linear system from a sparse matrix collection. Kuefler and Chen [4] use reinforcement learning to select the best preconditioned solver for sparse linear systems from a sparse matrix collection.

In this work, we focus specifically on using both single-label and multi-label classifiers to improve the performance of a numerical model using real-world problems instead of using a matrix collection. The linear systems generated by the ADaptive Hydraulics (ADH) problems of interest are larger than the linear systems typically found in these collections. We are interested in studying how physical attributes of the simulated system affect the solver selection and how the solver selection must change over the course of a transient simulation in order to produce the best results. This work further builds on the work of Nguyen et al. [5], who studied the performance of sparse linear solvers on ADH for a model of the John Day Lock. They studied matrix attributes over the course of the simulation to better understand the performance of linear solvers. They experiment with a number of preconditioned solvers, concluding that the BiCGStab(ℓ) solver is able to efficiently and reliably simulate the John Day Lock model.

3. Background

3.1. ADaptive Hydraulics Modeling System

ADaptive Hydraulics (ADH) provides users with the capability to simulate saturated and unsaturated groundwater flow, overland flow, three-dimensional Navier-Stokes flow, and two- or three-dimensional shallow-water problems. This work focuses on using the 3-D Navier-Stokes numerical flow solver to simulate free-surface flow in complex 3-D structures for the evaluation of navigation locks [6, 5]. ADH uses the Galerkin least-squares finite element method for solving the Reynolds-averaged incompressible turbulent 3-D Navier-Stokes equations. Turbulence is modeled with an adverse pressure gradient eddy viscosity technique. ADH uses the Newton algorithm to solve the nonlinear problem.

The Reynolds-average Navier-Stokes (RANS) equations for the conservation of mass and momentum to compute the pressure and velocity components throughout the flow domain are

$$\frac{\partial}{\partial x_i}\overline{u_i} = 0 \text{ and } \frac{\partial \rho \overline{u_i}}{\partial t} + \frac{\partial \rho \overline{u_j u_i}}{\partial x_j} = -\frac{\partial \overline{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\mu \frac{\partial \overline{u_i}}{\partial x_i} - \rho \overline{u_i' u_j'}\right), \tag{1}$$

where ρ is density, p is mean time-averaged pressure, μ is molecular viscosity, $\overline{u}$ is mean time-averaged velocity, and u' is the fluctuating component, and $-\rho u_i' u_j'$ is Reynolds stresses. Reynolds stresses are modeled to achieve closure by the eddy viscosity model to correctly account for turbulence. These equations are discretized using the Galerkin least-squares finite element algorithms on tetrahedral elements with first-order accuracy.

3.2. Numerical and Database Software

PETSc [8] provides users with access to a suite of data structures and routines for parallel scientific applications, including a wide variety of fast, scalable linear solvers and preconditioners. ADH has been interfaced with PETSc, providing ADH users with access to these fast linear solvers and preconditioners. We use the AnaMod library to help compute numerical metadata. AnaMod is a part of the Self-Adapting Large-scale Solver Architecture (SALSA) [9] software project, which aims to assist applications in finding suitable linear and nonlinear solvers based on analysis of the application-generated data.

We generate Web Ontology Language (OWL) [10] databases using the OWL API [11, 12]. OWL provides a language for developing ontology documents for use by applications that need greater information-processing capabilities. Many tools exist for creating, editing, and reasoning with OWL ontologies, providing us with tools to access the OWL ontology within ADH or to view and process the database on our own using an ontology editor. We use the OWL API to access and modify the OWL ontologies from within the numerical model. The OWL API contains a set of interfaces for inspecting, manipulating, and reasoning with OWL ontologies.

3.3. Machine Learning

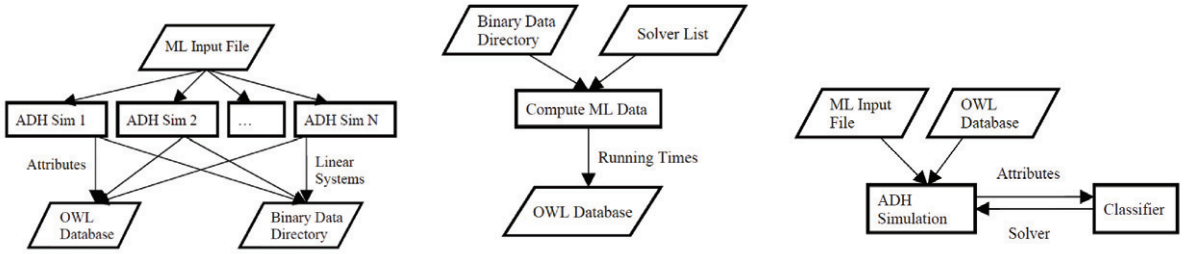
We use the WEKA and MULAN data mining software packages to access a wide variety of single-label and multi-label machine learning algorithms. These classifiers are generated by passing the machine learning algorithm a collection of instances, with each instance containing a set of attribute values and one or more labels. Once the classifier has been generated, we can pass the classifier a set of attributes as input and the classifier will return a single label as the class value. In this case, the instances are linear systems produced by ADH, the attributes are the properties of the linear system, and the labels are preconditioned linear solvers.

The WEKA [13] data mining software provides access to a comprehensive collection of machine learning algorithms and data processing tools. WEKA provides tools for regression, classifications, clustering, association rule mining, and attribute selection. WEKA provides access to single-label classifiers such as nearest neighbor classifiers (IBk, KStar), decision trees (J48, RandomTree, RandomForest), support vector classification (SMO), clustering (SimpleKMeans, EM, FarthestFirst), as well as boosting methods such as AdaBoost, each with a number of options to customize the classifier. For WEKA classifiers, a single label is listed as the class value for each instance. In this case, we use the fastest solver for each linear system as the label.

MULAN [14] is a data mining software library that provides access to a wide variety of machine learning algorithms for multi-label classifiers. MULAN provides tools for classification, ranking, feature selection, and evaluation. MULAN provides access to multi-label classifiers such as nearest neighbor classifiers (MLkNN, BRkNN, IBk), neural network learners (BPMLL), as well as meta classifiers like RAkEL (RAndom k-labELsets) and RAkELd (RAndom k Disjoint labELsets), each with a number of options to customize the classifier. MULAN also provides access to transformation algorithms such as the LabelPowerset learner capable of using WEKA single-label classifiers for multi-label learning problems. For multi-label classifiers, we can list one or more labels as the class value for each instance, allowing us to set a threshold to determine which solvers are fast for each linear system, at times resulting in a single solver being selected for the label and at times resulting in many solvers being selected for the label. This allows us to provide the machine learning algorithms with more examples of which linear systems each linear solver is capable of solving quickly and accurately.

4. Machine Learning Interface

The machine learning interface uses PETSc, AnaMod, and OWLAPI in addition to the machine learning software to dynamically select the solver to use for each linear system produced during a simulation. In order to use machine learning, we must generate a database containing simulation attributes and running times for a set of linear systems tested against a set of solvers. This dataset is used to generate a classifier at the beginning of each machine learning simulation. A number of command line options are provided to simplify this process.



(a) Generate data (b) Update database (c) Run ADH with machine learning

Figure 1: Process to use machine learning with ADH. First we generate a database and linear system data, then test solvers with linear systems and update database with running times, and finally use ADH with machine learning to dynamically select the solvers for each linear system.

Table 1: Matrix attributes computed by AnaMod

Category	Attributes
Slow	trace-asquared, n-nonzero-diags, avg-diag-dists, sigma-diag-dist
Non-Unique	col-variability, n-dummy-rows, dummy-rows-kind, trace-abs, diagonal-sign, diag-definite diag-zerostart, norm1, left-/right-bandwidth, positive-fraction, symmetry, nnzdia, nnzup, nnzlow, upband
Ritz-values	ritz-values-r, ritz-values-c
Ritz-based	ellipse-ax/-cx/-ay/-cy, kappa, sigma-max/-min, lambda-max/min-by-mag-re, lambda-max-by-real/im-part-re, lambda-max/min-by-mag-im, lambda-max-by-real/im-part-im, ruhe75-bound
Unique	row-variability, diagonal-average, diagonal-variance
Fast Unique	trace, normInf, normF, diagonal-dominance, nrows, nnzeros, max-/min-nnzeros-per-row, blocksize, avgdistfromdiag, nnz, avgnnzprow, loband

4.1. Generating Database

In order to use machine learning, we must first create a database containing training data for the machine learning algorithm. Figure 1(a) shows how to generate a database and save the linear systems. We run simulations using input options to save attributes to a database with a safe choice for the preconditioned solver that we are confident will converge in a reasonable time. For this work, we use the BCGSL solver with the block-Jacobi preconditioner using two search directions.

The -save_stats or -save_all_stats commands create an OWL database containing attributes for each linear system produced during the simulation. Using AnaMod, we compute attributes for each linear system for the categories simple (normlike quantities), variance (heuristics estimating how different elements in the matrix are), normality (estimates of the departure from normality), structure (nonzero structure properties), and spectrum (eigenvalue and singular value estimates produced using GMRES iterations). Table 1 shows the attributes we compute based on uniqueness and compute time. Using the full set of attributes results in high overhead, so we want to use a minimal number of attributes while still generating accurate classifiers. Attributes related to the physical system and computational methods (Table 2) are also added to the database. These attributes are passed directly from ADH, requiring minimal additional computation. When generating the database, we use the input option -save_systems to save the full linear systems produced by the simulation to a binary data directory for further processing later.

4.2. Testing Preconditioned Solvers

Once we have created a database with attributes for many linear systems, we must determine the best preconditioned solvers for each linear system. Figure 1(b) shows how to update the database with the running times for the

Table 2: Simulation attributes computed by ADH

Category	Attributes
Physical	eddy viscosity, inflow velocity, inflow velocity change
Residual	residual-l2-norm, residual-max-norm
Adaptation	adapted, ref/unref-cycles, ref/unref-max-node-change, ref/unref-percent-change
Computational	3d-max/min-area, max/min-nodes, nonlin-iteration, percent-complete, procs, sim-time, time-step

preconditioned solvers. We can use a separate program to solve each saved linear system with many different preconditioned solvers, adding the running times for each solver for each linear system to the database. The user passes the program a list of PETSc preconditioned solvers that may perform well at some point during the simulation.

4.3. Using Machine Learning

Once the full database has been generated, simulations can be run using machine learning to select a solver for each linear system. Figure 1(c) shows how to use machine learning with ADH. The machine learning command line option -use_ml <number>, using 1 for WEKA classifiers and 2 for MULAN classifiers, sets the numerical model to use machine learning, while the command line option -ml_classifer <classifier name> sets the machine learning classifier to use.

A function to generate each machine learning classifier must be written and compiled in the machine learning section of the code. This code must use WEKA or MULAN functions to create, build, and return a classifier. The user can also define the attributes they want to compute in the input file ml.data. This allows the user to limit the number of attributes computed for each linear system. Additional attributes specific to the simulation code can also be used by the classifier. The user must add code to compute these values and pass the name of the attribute and its value to the machine learning interface. This functionality allows the user to control which attributes are computed during the simulation and choose the best classifier for their problem.

At the beginning of an ADH simulation with machine learning, the machine learning interface will access the database and create a dataset. This dataset is used by the machine learning algorithm to create a classifier. A machine learning input file is passed as input to the machine learning interface to determine which attributes and solvers are used during the simulation.

5. Experimental Setup

5.1. Test Model

We test the effectiveness of the machine learning algorithms with the Watts Bar Lock model. ADH models are frequently used to simulate locks filling with water, resulting in difficult to compute transient simulations due to rapid changes in flow velocity and pressure at the beginning of the simulation. Many linear solvers have difficulty solving the resulting linear systems, making linear solver selection an important factor in getting a fast, accurate solution. The Watts Bar Lock model has a long culvert with a slope at the beginning that leads to a tainter valve with a valve well. There are bulkheads before and after the valve well. The finite element mesh for this model uses 1,635,510 elements to simulate a 23.0 x 0.888 x 3.778 ft area.

We change the values of the eddy viscosity parameter and inflow speed to create 12 variations on this model. Eddy viscosity refers to the resistance or “thickness” of fluids in the model. Larger values for the eddy viscosity result in simulating with less turbulence, allowing the user to focus on larger trends occurring in the model. We test eddy viscosities of $2ft^2/s$, $5ft^2/s$, $10ft^2/s$, and $50ft^2/s$. We change the inflow speed by increasing or decreasing hydrostatic pressure at the inflow boundary. We test inflow pressures of 725Pa, 740Pa, and 755Pa. Using lower eddy viscosities and faster flows increases the turbulence of the fluid in the model, resulting in more difficult to compute transient simulations.

5.2. Preconditioned Solvers

We tested many PETSc solvers, preconditioners, and subpreconditioners against ADH-produced linear systems and selected 70 preconditioned solvers that produced fast running times for a significant number of linear systems. Many of the tested preconditioned solvers did not perform well for the tested ADH linear systems. Due to the large amount of time needed to generate the data for each solver to add to the database, we must limit the number of solvers in order to generate the full database in a reasonable amount of time. We test BiCGStab(ℓ) (BCGSL) with 2, 4, 6, 8, and 10 search directions and the -ksp_bcgsl_cxpoly option. We test the flexible generalized minimal residual method (FGMRES) with 10, 25, 50, 100, and 200 search directions. We test the additive Schwarz method, block-Jacobi, and Jacobi preconditioners. We test the incomplete LU subpreconditioner with 0 and 1 factor levels and the LU subpreconditioner. If none of the tested solvers are able to solve a linear system, then the best solver for that linear system is listed as "none". If the classifier selects "none" as the solver, then the numerical simulation will skip solving the linear system, causing the nonlinear iteration to fail and the time-step to be reduced.

5.3. Test Setup

We test the accuracy of the classifiers by generating full classifiers with knowledge of all variations of a model and test classifiers with knowledge of all but one variation of a model. We test the full classifiers against all variations of the model. This demonstrates the performance of the classifiers when they have prior knowledge of the transient simulation being run and other similar transient simulations. We test each test classifier against the variations of the model of which the test classifier does not have prior knowledge. This demonstrates the performance of the classifiers when they do not have prior knowledge of the transient simulation being run, allowing us to better predict classifier performance against new variations of a model.

We performed experiments testing classifiers without any knowledge of the model being run, but knowledge of other models. However, we do not have access to enough models to generate linear systems that fully cover the space of linear systems generated by ADH. These tests produced inaccurate results since the classifiers often did not have knowledge of any linear systems similar to the ones produced by the model being run. In practice, scientists run many variations of each model hundreds of times. Allowing them to get fast, accurate results after running some preprocessing routines can greatly accelerate their work.

We perform tests using the WEKA J48, J48 AdaBoost, J48 AdaBoost with 50 iterations, IBk (1, 2, and 4 nearest neighbors), IBk AdaBoost (1 nearest neighbor), KStar, RandomForest, RandomTree, and SMO classifiers. We perform tests using the MULAN BPMLL (0.05, 0.10, and 0.20 learning rate), MLkNN (5, 20, and 40 nearest neighbors), BRkNN with (5, 20, and 40 nearest neighbors), and MLkNN ClusteringBased classifiers with SimpleKMeans, EM (Expectation Maximization), and FarthestFirst clusterers. We also generate multi-label classifiers for the WEKA classifiers using LabelPowerset classifiers and the RAkEL and RAkELd meta classifiers.

In order to reduce the overhead of dynamic linear solver selection, we test classifiers generated with multiple sets of attributes (Table 3). The normal attribute set eliminates non-unique and slow attributes, while reducing the number of ritz-values. The reduced attribute set eliminates the ritz-values, but still uses the ritz-based values. The minimal attribute set uses only attributes that are both fast and unique in an attempt to reduce the attribute computation time as much as possible.

Tests are performed on Garnet, a Cray XT6 with 1260 compute nodes. Each node contains a 2.4-GHz AMD Opteron 64-bit 16-core processor and 32 GB of dedicated memory. The nodes are connected using a Cray Gemini Ethernet interconnect. We use 16 nodes (256 cores) for our tests to allow the most difficult simulations to finish in a reasonable amount of time.

6. Results and Analysis

6.1. Classifier Performance

This section compares the performance of ADH using classifiers produced using WEKA and MULAN machine learning algorithms with the performance using a single PETSc BCGSL and FGMRES solver for a full simulation. First we determine the best performing PETSc solvers. Figure 2 shows the performance of the tested PETSc preconditioned linear solvers on the Watts Bar Lock simulations. The fastest BCGSL solver outperforms the fastest FGMRES solver for the more difficult simulations, while the fastest FGMRES solver outperforms the fastest BCGSL solver for

Table 3: Attribute Sets

Name	Attribute Categories
Full	Non-Unique, 60 Ritz-values, 60 Ritz-based, Unique, Fast Unique
Normal 30	30 Ritz-values, 30 Ritz-based, Unique, Fast Unique
Reduced 30	30 Ritz-based, Unique, Fast Unique
Minimal	Fast and Unique

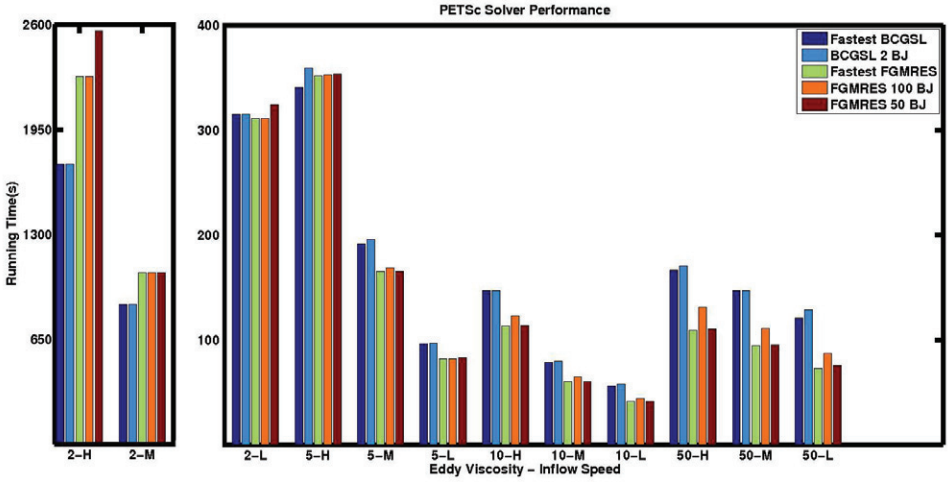


Figure 2: Comparison of the fastest PETSc solver for each model variation with three individual PETSc solvers for ADH Watts Bar Lock simulations with varying eddy viscosities ($2ft^2/s$, $5ft^2/s$, $10ft^2/s$, and $50ft^2/s$) and inflow speeds ((L)ow, (M)edium, or (H)igh hydrostatic pressure at inflow boundary).

the less difficult simulations. We see that BCGSL block-Jacobi with 2 search directions is the fastest or near fastest BCGSL solver. For FGMRES, we see that FGMRES block-Jacobi with 100 search directions is the fastest or near fastest FGMRES solver for the more difficult simulations, while FGMRES block-Jacobi with 50 search directions is the fastest or near fastest solver for the less difficult simulations. This suggests that for the Watts Bar Lock simulations, BCGSL block-Jacobi with 2 search directions is best for the more difficult simulations, while FGMRES block-Jacobi with 50 search directions is best for the less difficult simulations.

Once we have determined the fastest PETSc solvers, we compare these solvers to the fastest solvers using machine learning for dynamic linear solver selection. Figure 3 compares the fastest BCGSL and FGMRES solvers to the fastest single-label and multi-label classifiers for both the test and full classifiers. We generate single-label classifiers using WEKA. We see that the fastest single-label classifiers for WEKA and WEKA Full perform well for some model variations, but do not perform as well for other model variations, producing slower running times than the fastest BCGSL and FGMRES solvers. We also see that the test classifiers outperform the full classifiers in some situations. The additional information that the full classifiers are given should allow them to outperform the test classifiers. This suggests that the single-label solvers do not have enough data to accurately predict the best linear solvers in some situations. Additional tests demonstrated that individual WEKA and WEKA Full classifiers would perform well for some model variations, but would produce significantly slower running times than the best BCGSL and FGMRES solvers for other model variations. At times, the WEKA and WEKA Full classifiers would fail to solve some linear systems, preventing the ADH simulation from running to completion.

Therefore, we test the multi-label classifiers using MULAN. We see that the fastest multi-label classifiers outperform the fastest BCGSL and FGMRES solvers for every model variation and that the fastest MULAN Full classifiers outperform the fastest MULAN classifiers in every case. The MULAN classifiers produce significant speedups for the more difficult simulations, while the MULAN classifiers produced running times slightly faster than the best BCGSL

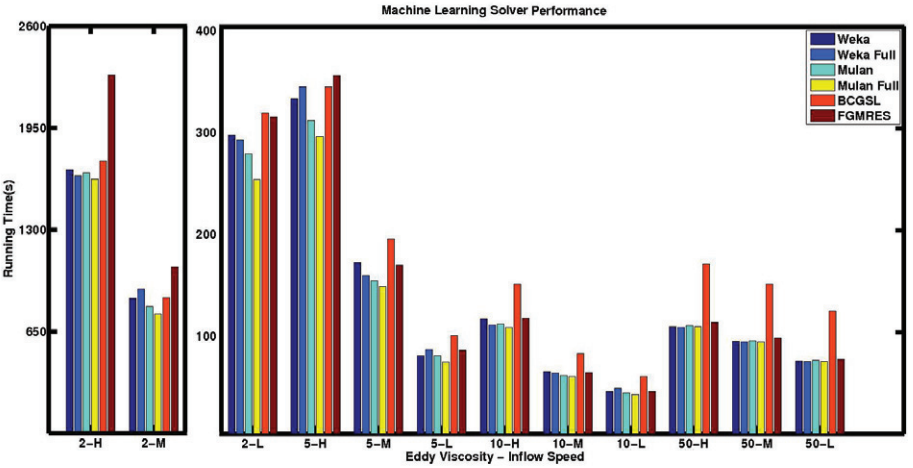


Figure 3: Comparison of fastest PETSc BCGSL and FGMRES solvers for each model variation with the fastest WEKA single-label and MULAN multi-label classifiers for each model variation for ADH Watts Bar Lock simulations with varying eddy viscosities ($2ft^2/s$, $5ft^2/s$, $10ft^2/s$, and $50ft^2/s$) and inflow speeds ((L)ow, (M)edium, or (H)igh hydrostatic pressure at inflow boundary).

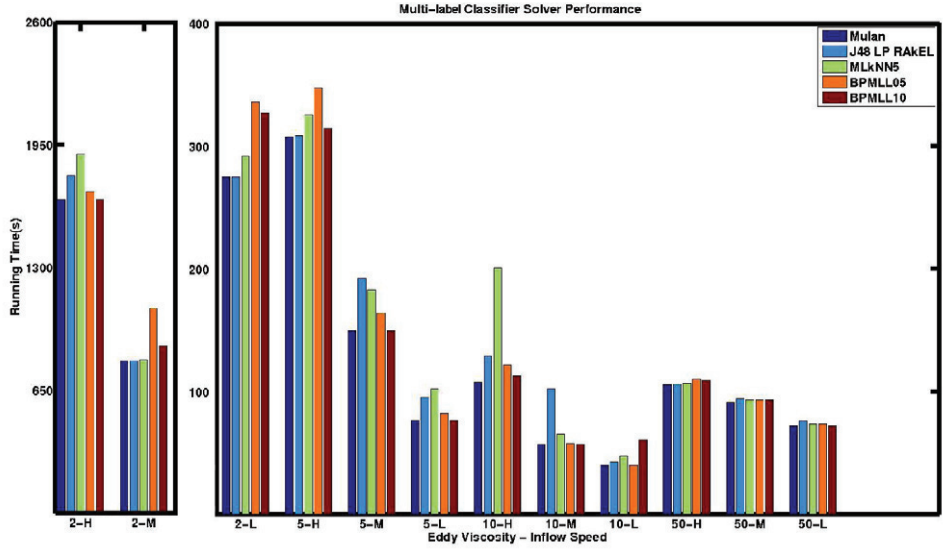


Figure 4: Comparison of the fastest MULAN multi-label classifier for each model variation with the J48 LP RAkEL, MLkNN with 5 nearest neighbors, BPMLL with a 0.05 learning rate, and BPMLL with a 0.10 learning rate MULAN multi-label classifiers for ADH Watts Bar Lock simulations with varying eddy viscosities ($2ft^2/s$, $5ft^2/s$, $10ft^2/s$, and $50ft^2/s$) and inflow speeds ((L)ow, (M)edium, or (H)igh hydrostatic pressure at inflow boundary).

and FGMERS solvers for the simpler simulations. This suggests that the additional information provided by listing multiple fast preconditioned linear solvers for each linear system provides enough information to accurately predict fast linear solvers for each linear system encountered by an ADH simulation.

Next we look at some specific multi-label classifiers and compare them to the fastest multi-label classifiers for each model variation. In Figure 4 we see that for each classifier, there are some model variations where the classifier performs well and others where it does not. In most cases, the classifiers do not perform significantly worse than the fastest multi-label classifier, but there are some cases where we see significant slowdowns. Further experiments with the machine learning algorithms are necessary to produce more consistent individual classifiers.

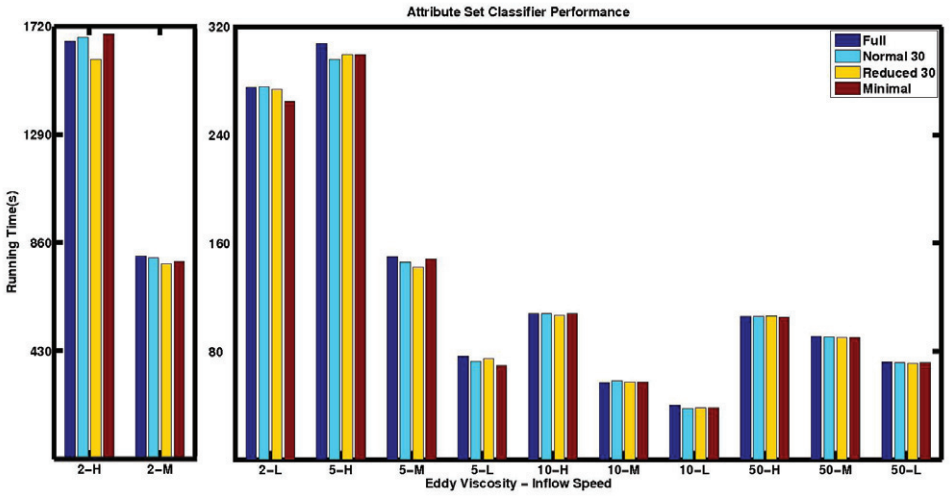


Figure 5: Comparison of the fastest MULAN multi-label classifier for each model variation with the fastest MULAN multi-label classifiers generated using the Normal 30, Reduced 30, and Minimal attribute sets for ADH Watts Bar Lock simulations with varying eddy viscosities ($2ft^2/s$, $5ft^2/s$, $10ft^2/s$, and $50ft^2/s$) and inflow speeds ((L)ow, (M)edium, or (H)igh hydrostatic pressure at inflow boundary).

Table 4: ADH machine learning classifier generation, solver selection, and attribute computation running times for the J48 Labelset RAkEL, MLkNN with 5 nearest neighbors, and BPMLL with a learning rate of 0.10 classifiers for the Watts Bar Lock simulation with eddy viscosity 2 and medium flow rate.

Classifier	Attribute Set	Classifier Generation	Solver Selection	Attribute Computation
J48 LP RAkEL	Full	1231.08s	124.99s	921.81s
	Normal 30	748.32s	80.65s	37.81s
	Reduced 30	457.14s	49.67s	38.80s
	Minimal	322.73s	35.43s	5.75s
MLkNN5	Full	24.97s	40.00s	867.51s
	Normal 30	20.78s	25.41s	41.96s
	Reduced 30	24.04s	11.27s	39.41s
	Minimal	22.63s	11.07s	5.81s
BPMLL10	Normal 30	261.60s	57.19s	801.36s
	Normal 30	181.40s	31.34s	44.45s
	Reduced 30	197.91s	10.27s	42.79s
	Minimal	187.88s	7.18s	5.77s

6.2. *Classifier Overhead*

Using machine learning to dynamically select linear solvers results in a significant amount of overhead. Therefore, we want to limit the amount of overhead as much as possible. We first want to see if reducing the number of computed attributes affects the ability of the classifiers to accurately select linear solvers. Figure 5 shows that the normal 30, reduced 30, and minimal attribute sets result in classifiers that are about as fast as the classifiers generated by the full attribute set. We see that in some cases we obtain a small speedup, while in other cases we get a slight reduction in speed. This suggests that we can reduce the number of attributes we compute while still accurately predicting the best linear solver for each linear system.

Next we want to see how much overhead using machine learning creates. Computing a large number of attributes for each linear system encountered by a simulation has the potential to significantly increase the total running time of

the simulation, eliminating any speedups obtained by the classifiers. However, by carefully selecting which attributes we compute, we can limit this overhead. Table 4 demonstrates that we can significantly reduce the overhead for the more difficult Watts Bar Lock simulation with eddy viscosity 2 and medium flow rate by removing a couple of attributes that are time-consuming to compute. Using the minimal number of attributes also results in some additional speedups. We also see that reducing the number of attributes also reduces the time needed to generate the classifier and select the solver. However, to obtain the best performance, we will need parallel versions of these machine learning algorithms. At the moment, both WEKA and MULAN only run on a single processor when generating the classifier and selecting the linear solver. Creating parallel versions of WEKA and MULAN classifiers would significantly reduce the time spent generating the classifiers and selecting the solvers.

7. Conclusions

This work demonstrates that dynamic linear solver selection using multi-label classifiers allows us to outperform the fastest BCGSL and FGMRES solvers for transient simulations. The single-label classifiers are not able to consistently produce fast running times due to only being able to associate one solver with each linear system. The multi-label classifiers are able to obtain fast running times due to their ability to associate multiple solvers with each linear system, providing the machine learning algorithms with more examples of the linear systems each solver can quickly and accurately solve.

Using machine learning can create a significant amount of overhead, but this work demonstrates that limited attribute sets can be used to greatly reduce the amount of time spent computing attributes, as well as reducing the amount of time needed to generate the classifier and select the linear solvers. However, in order to obtain the best performance, we will need parallel versions of these machine learning algorithms.

Acknowledgment

We would like to thank Allen Hammack at the U.S. Army Engineer Research and Development Center Coastal and Hydraulics Laboratory for providing us with ADH models. This study was supported by the U.S. Army Engineer Research and Development Center Civil Works Basic Research Program and an allocation of computer time from the DoD High Performance Computing Modernization Program.

References

[1] S. Bhowmick, V. Eijkhout, E. Fuentes, D. Keyes, Application of machine learning to the selection of sparse linear solvers, in: Int. J. of High Performance Computing Applications, 2006.
[2] S. Bhowmick, B. Toth, P. Raghavan, Towards low-cost, high-accuracy classifiers for linear solver selection, in: Proceedings of the 9th Int. Conf. on Computational Science, 2009.
[3] A. Holloway, T.-Y. Chen, Neural networks for predicting the behavior of preconditioned iterative solvers, in: Proceedings of the 7th Int. Conf. on Computational Science, 2007.
[4] E. Kuefler, T.-Y. Chen, On using reinforcement learning to solve sparse linear systems, in: Proceedings of the 8th Int. Conf. on Computational Science, 2008.
[5] H. V. Nguyen, J.-R. C. Cheng, E. A. Hammack, R. S. Maier, Parallel newton-krylov solvers for modeling of a navigation lock filling system, in: Proceedings of the 10th Int. Conf. on Computational Science, 2010.
[6] E. A. Hammack, R. L. Stockstill, 3d numerical modeling of john day lock tainter valves, in: Proceedings of World Environmental and Water Resources Congress, 2009.
[7] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, H. Zhang, Portable, extensible toolkit for scientific computation, http://www.mcs.anl.gov/petsc (2009).
[8] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, H. Zhang, PETSc users manual, Tech. Rep. ANL-95/11 - Revision 3.0.0, Argonne National Laboratory (2008).
[9] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vudoc, R. C. Whaley, K. Yelick, Self adapting linear algebra algorithms and software, Proceedings of the IEEE 93 (2).
[10] D. L. McGuinness, F. V. Harmelen, Owl web ontology language overview, w3C recommendation, http://www.w3.org/TR/owl-features (2004).
[11] M. Horridge, S. Bechhofer, The owl api: A java api for working with owl 2 ontologies, in: OWLED 2009, 6th OWL Experienced and Directions Workshop, 2009.
[12] S. Bechhofer, P. Lord, R. Volz, Cooking the semantic web with the owl api, in: 2nd International Semantic Web Conference, 2003.
[13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, The weka data mining software: An update, SIGKDD Explorations 11 (1).
[14] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, I. Vlahavas, Mulan: A java library for multi-label learning, Journal of Machine Learning Research 12 (2011) 2411–2414.