Application of Machine Learning for Solver Selection

Sanjukta Bhowmick

Department of Applied Physics and Applied Mathematics, Columbia University

Mathematics and Computer Science Division

Argonne National Laboratory

In collaboration with

Victor Eijkhout (TACC), Yoav Freund (UCSD),Erika Fuentes (University of Tennessee) and David Keyes (Columbia University)

Motivation: Extensive Use of Linear Solvers

- The execution time in many PDE-based simulations is dominated by the time to solve linear systems ***Ax=b***
  - MHD codes *M3D*, *Nimrod*, *AORSA* in U.S. DOE
  - Aerodynamics code *TRANAIR* at Boeing
- Different simulations/applications may have different solution requirements:
  - robustness, accuracy, fast execution time , low memory requirement , parallel scaling,…..
- The linear solution time (in turn total simulation time) can be reduced if solvers are chosen to match problem attributes
- Automating solver selection and composition is one of the challenges in scientific computing

Motivation: Difficulty of Method Selection

- There exists a vast range of choices for tuning linear solvers
  - Solvers: direct *(order),* iterative *(tolerance, restart value),* multilevel/multigrid *(number of levels, cycles)*
  - Preconditioner loop: method type, fill parameters, overlap parameters, number of levels, coarsening type, etc.

- Given methods {A, B, C} it is not hard to find problems arising naturally from PDE discretizations {x, y, z}, such that :
  - On x, A > B > C
  - On y, B > C > A
  - On z, C > A > B

“>” is an order relation for better performance

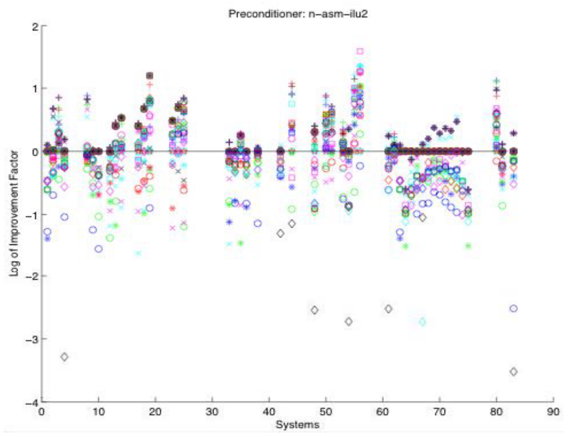
“The impossibility of finding the best linear solver for a given problem…is widely appreciated.” Towards Polyalgorithmic Linear System Solvers for Nonlinear Elliptic Problems (SIAM Journal of Scientific Computing)

Opportunity For Solver Tuning

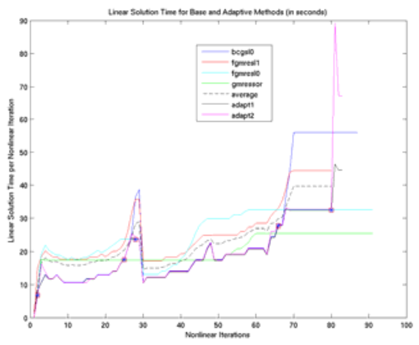
Example of performance variations of differently tuned Krylov solvers over a sequence of about 85 systems

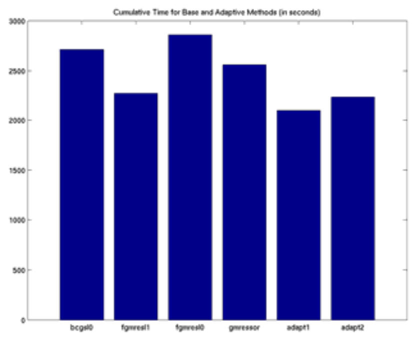
Vertical axis is log of runtime improvements

Opportunity is potentially orders of magnitude



Using Multiple Solvers





Adapt1: (1) 1st order: BCGS / BJacobi with ILU(0)
(25) 1st order: FGMRES(30) / BJacobi with ILU(0)
(28) 2nd order: BCGS / Bjacobi with ILU(0)
(66) 2nd order: FGMRES(30) / BJacobi with ILU(0)
(80) 2nd order: FGMRES(30) / BJacobi with ILU(1)

Adapt2: (1) 1st order: GMRES(30) / Bjacobi with SOR
(2) 1st order: BCGS / BJacobi with ILU(0)
(25) 1st order: FGMRES(30) / BJacobi with ILU(0)
(28) 2nd order: BCGS / Bjacobi with ILU(0)
(66) 2nd order: FGMRES(30) / BJacobi with ILU(0)
(80) 2nd order: FGMRES(30) / BJacobi with ILU(1)

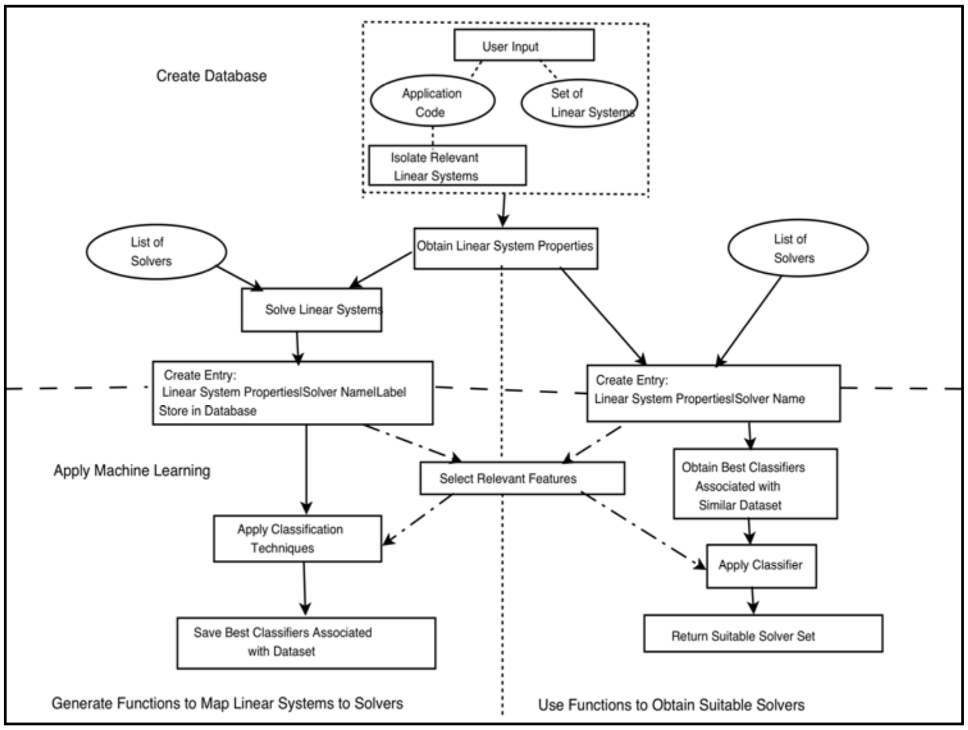
Simulation of Petsc-Fun3d code

Multisolver Research

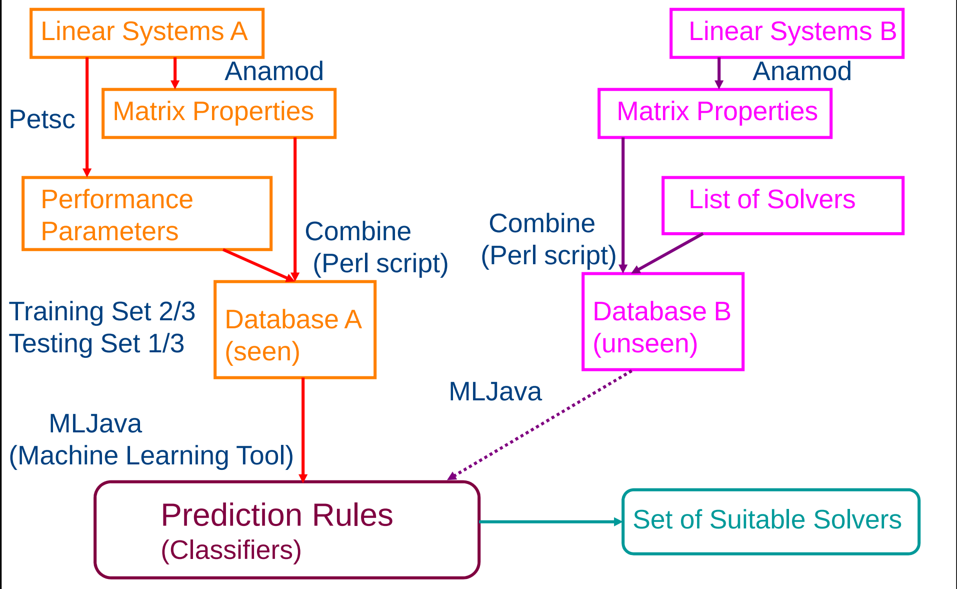
- Performance of several preconditioned Krylov iterations and stationary methods in nonlinear elliptic PDEs [Ern et. al. 1994]
- Polyalgorithmic parallel linear system solution [Barret et al. 1996]
- Linear System analyzer [Bramley et. al. 2000]
- Multimethod Solvers (Composites and Adaptive) [B. et al. 2003]
- SALSA – Self-Adapting Large-scale Software Architecture [Dongarra, Eijkhout et. al. 2003]
- Support Vector Machines for predicting precondtioners [Xue et. al. 2005]
- Neural networks for predicting precondtioners [Holloway et. al. 2007]

Applying Machine Learning

- Machine Learning has been used to harvest large datasets to associate features of an event with outcomes
  - medical diagnoses, retail purchases, hand-written characters, etc.
- Many applications are solved over code lifetime of decades
  - We can obtain a vast amount of information about the systems and corresponding solvers
  - This information (mountains of performance data) can give us pointers for selecting solvers for future runs
- Machine learning methods are used for selecting a method based on problem features
  - application parameters, matrix condition number or sparsity pattern, symmetry, etc.
- A "trained" classifier can reliably decide how to respond to such features in subsequent encounters
  - Input: Database of matrix characteristics X performance with candidate solvers
  - Output: Mechanism to predict a set of suitable solvers for a given matrix

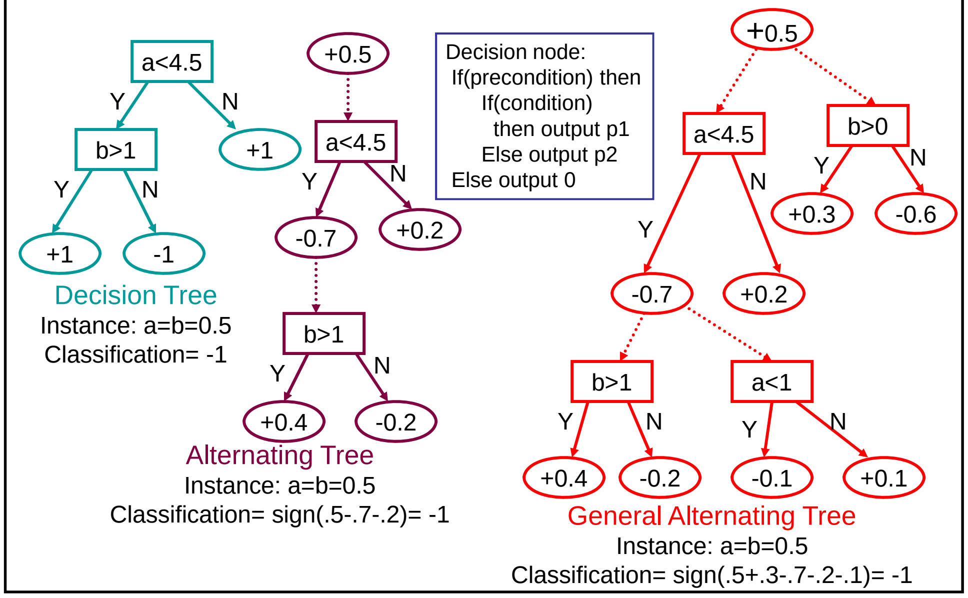

Create Database
User Input
Application Code
Set of Linear Systems
Isolate Relevant Linear Systems
List of Solvers
Obtain Linear System Properties
List of Solvers
Solve Linear Systems
Create Entry: Linear System Properties|Solver Name|Label Store in Database
Create Entry: Linear System Properties|Solver Name
Apply Machine Learning
Select Relevant Features
Obtain Best Classifiers Associated with Similar Dataset
Apply Classification Techniques
Apply Classifier
Save Best Classifiers Associated with Dataset
Return Suitable Solver Set
Generate Functions to Map Linear Systems to Solvers
Use Functions to Obtain Suitable Solvers

Applying Machine Learning



Alternating Decision Trees

- An alternating decision tree is a mapping from instances to real numbers which is defined in terms of a set of base rules

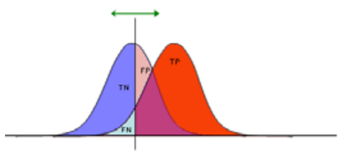


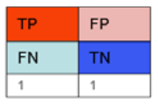
Boosting Algorithm: AdaBoost

- Boosting refers to a general and provably effective method of producing a very accurate prediction by combining rough and moderately inaccurate rules of thumb---A Short Introduction to Boosting
- AdaBoost implements the boosting algorithm as follows;
  - Given database $(x_1,y_1),\ldots,(x_m,y_m)$ where $x_i \in X$ and $y_i \in Y=\{-1,+1\}$
  - Initialize distribution $D_1(i)=1/m$
  - For iterations $t = 1,\ldots,T$
    - Train weak learner using distribution D_t
    - Get weak hypothesis h_t: $X \rightarrow \{-1,+1\}$
    - Update distribution $D_{t+1}(i)$ giving more weight to wrongly classified entries
  - Final hypothesis H(x) is formed by combining the all the h_t
- Applying Boosting to ADTrees:
  - Alternating decision trees can be defined as a sum of base rules
  - It is easy to apply boosting to ADTrees setting base rules as weak learners
  - MLJava implements AdaBoost on ADTrees

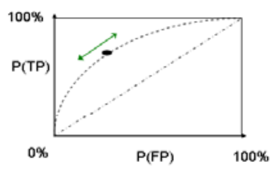
Accuracy:ROC Curves

- ROC (Receiver Operator Characteristic) curves give the accuracy of the classifier.
-

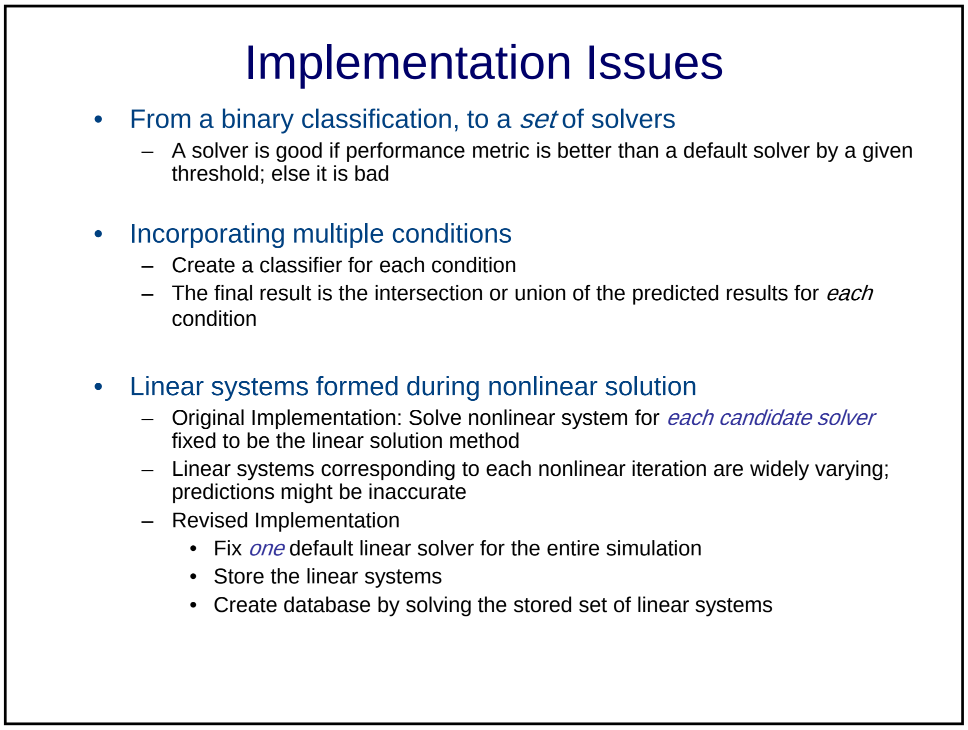




TP	FP
FN	TN
1	1

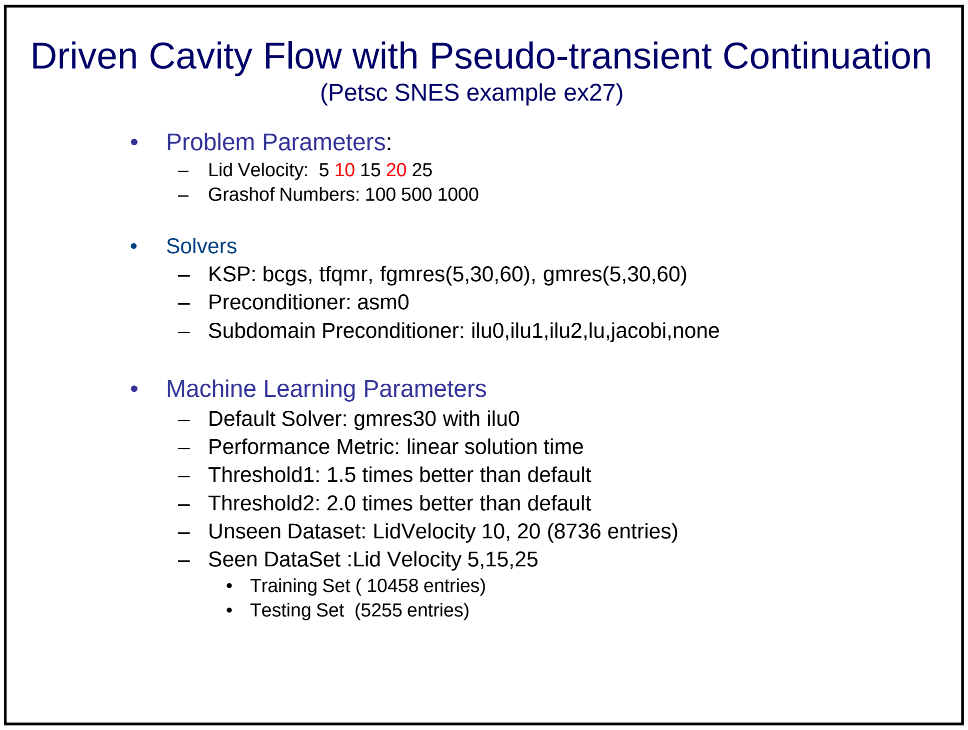


The closer the curve follows the left-hand border and then the top border the more accurate the test.
The x=y line represents random guessing; anything below it is worse than random guesses



Implementation Issues

- From a binary classification, to a *set* of solvers
  - – A solver is good if performance metric is better than a default solver by a given threshold; else it is bad
- Incorporating multiple conditions
  - – Create a classifier for each condition
  - – The final result is the intersection or union of the predicted results for *each* condition
- Linear systems formed during nonlinear solution
  - – Original Implementation: Solve nonlinear system for *each candidate solver* fixed to be the linear solution method
  - – Linear systems corresponding to each nonlinear iteration are widely varying; predictions might be inaccurate
  - – Revised Implementation
    - Fix *one* default linear solver for the entire simulation
    - Store the linear systems
    - Create database by solving the stored set of linear systems

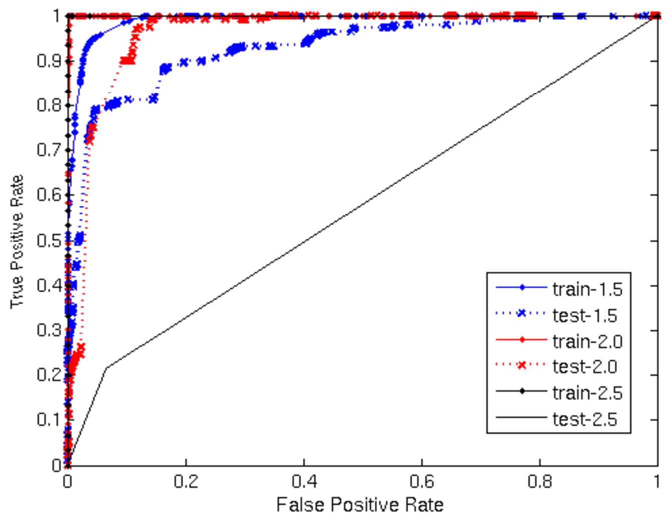


Driven Cavity Flow with Pseudo-transient Continuation

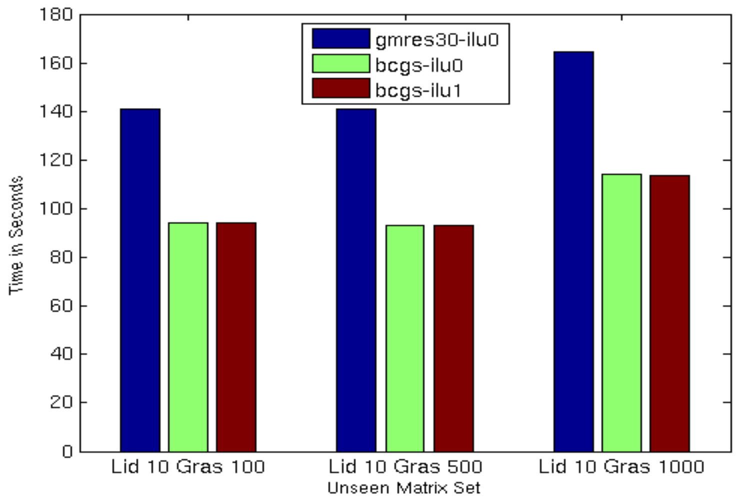
(Petsc SNES example ex27)

- Problem Parameters:
  - – Lid Velocity: 5 10 15 20 25
  - – Grashof Numbers: 100 500 1000
- Solvers
  - – KSP: bcgs, tfqmr, fgmres(5,30,60), gmres(5,30,60)
  - – Preconditioner: asm0
  - – Subdomain Preconditioner: ilu0,ilu1,ilu2,lu,jacobi,none
- Machine Learning Parameters
  - – Default Solver: gmres30 with ilu0
  - – Performance Metric: linear solution time
  - – Threshold1: 1.5 times better than default
  - – Threshold2: 2.0 times better than default
  - – Unseen Dataset: LidVelocity 10, 20 (8736 entries)
  - – Seen DataSet :Lid Velocity 5,15,25
    - Training Set (10458 entries)
    - Testing Set (5255 entries)

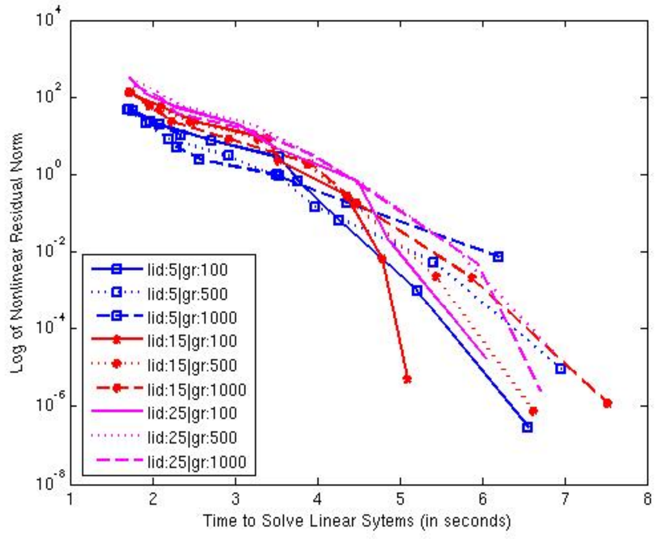
Results: ROC Curves



Results: Performance



Results: Most Prominent Feature (Nonlinear Residual Norm)



Linear Systems from M3D code

(w3.pppl.gov/~jchen)

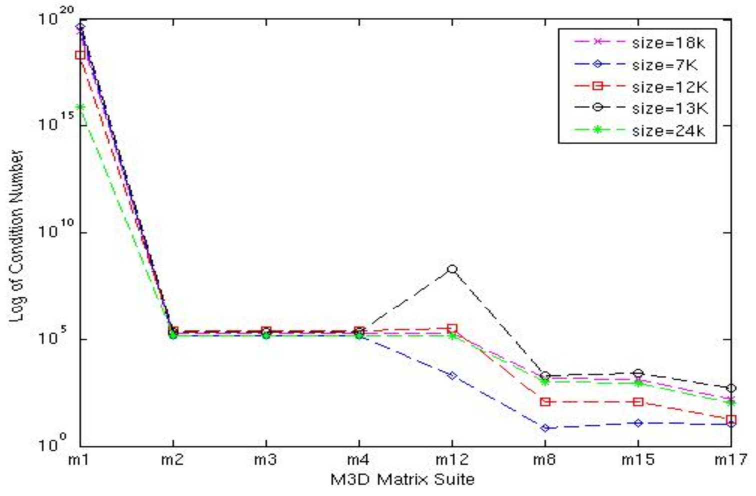
- Matrix Properties:
  - Number of Non-zeros (of the order 10): 7.6 12.6 13.1 18.5 24.2
  - Matrix groups m1,m2,m3,m4,m8,m12,m15,m17
- Solvers
  - Direct Methods: SuperLU, Spooles
  - Iterative Methods : bcgs, tfqmr, fgmres(5,30,60), gmres(5,30,60)
  - Preconditioners:
    - Domain Decomposition: asm0,asm1,asm2
    - Subdomain Preconditioner: ilu0,ilu1,ilu2, ilu3, icc0, icc1, icc2, icc3, lu,jacobi,sor,none
    - Algebraic Multigrid (Hypre): parasails,boomeramg,euclid,pilut
- Machine Learning Parameters
  - Classifier 1
    - Performance Metric : Convergence
  - Classifier 2
  - Performance Metric: linear solution time
    - Default Solver: gmres30 with asm0 and ilu0
    - Threshold: 1.5 times better than default

Solver Predictions

- M1:
  - Predicted Solver: gmres(60), fgmres(30,60) with BoomerAMG
  - 2/5 false predictions for both unseen matrices
- M2, M3, M4, M12
  - Predicted Solver: gmres(5), bcgs, tfqmr, with BoomerAMG
- M15, M17
  - Predicted solver gmres(30) with Jacobi
  - No solvers predicted for 24K matrices

Diverse database hampers predictions

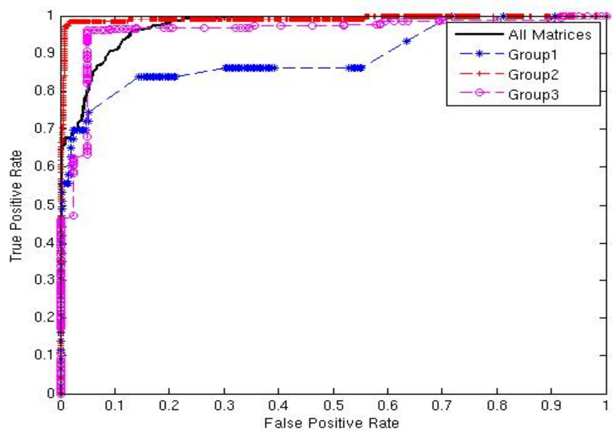
Results: Most Prominent Feature (Condition Number)



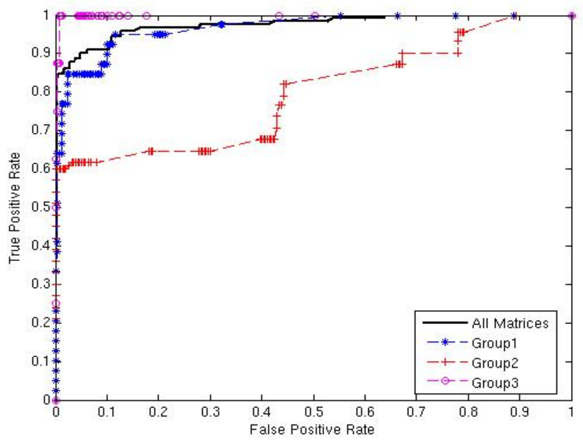
Grouping by Prominent Feature

- Matrices were grouped by most prominent feature
  - Group 1 (m1)
    - Seen Dataset: nnz 7.6,13.1,18.5
      - Training Set (524 entries)
      - Testing Set (298 entries)
    - Unseen Dataset: nnz 24.6 (matrix 1),12.6 (matrix 2) (700 entries)
  - Group 2 (m2, m3 , m4, m12)
    - Seen Dataset: nnz 7.6,13.1,18.5
      - Training Set (1644 entries)
      - Testing Set (870 entries)
    - Unseen Dataset: nnz 12.6, 24.6 (1788 entries)
  - Group 3 (m8, m15, m17)
    - Seen Dataset: nnz 7.6,18.5
      - Training Set (1144 entries)
      - Testing Set (596 entries)
    - Unseen Dataset: nnz 12.6, 24.6 (3342 entries)

Results: ROC Curves for Convergence

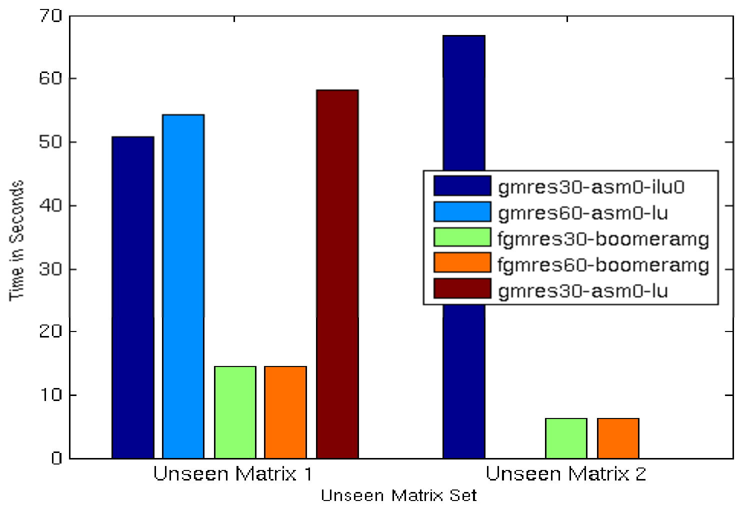


Results: ROC curves for time



Results: Performance

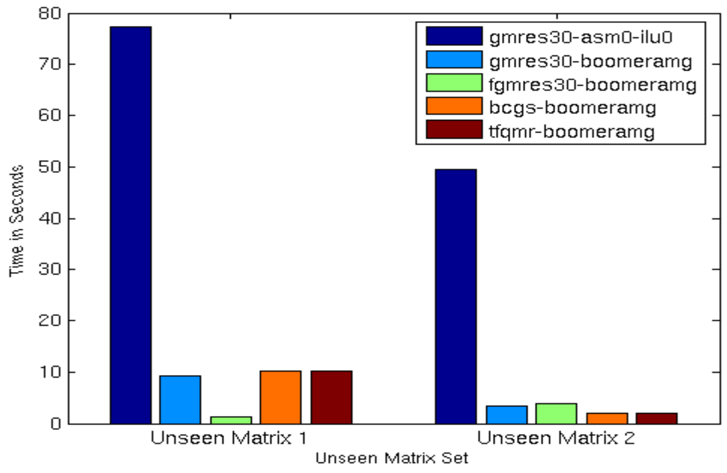
Group 1



False Positives: Unseen Matrix1 gmres30-boomeramg--diverged

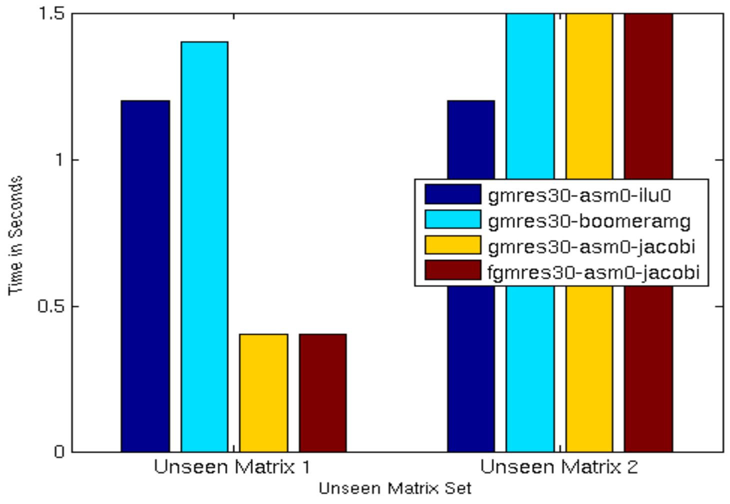
Results: Performance

Group 2



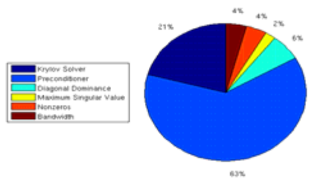
Results: Performance

Group 3



Feature Selection

- Computing linear system features is expensive
- Presence of corelated features
- Presence of zero variance features
- An inefficient set of features results in lower accuracy
- Identify important features using : weaker classifiers, genetic algorithms, principle component analysis, etc.
- Feature selection can highlight which matrix characteristics are necessary for solver selection
- An efficient set of features can nullify the choice of different machine learning methods



Summary

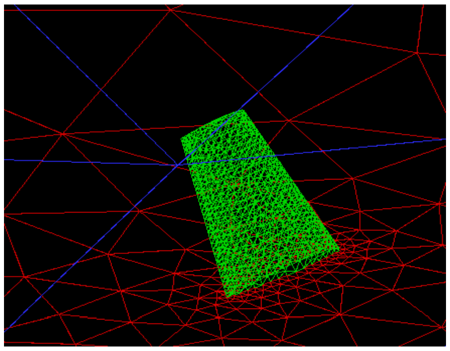
- We can use machine learning tools to build a software that detects a suitable set of linear solvers for a given system
- Once unseen data has been classified, the information can be fed into the seen data, creating a richer dataset
- Machine learning tools can be used to highlight important matrix characteristics that determine solver selection.
- Not confined to linear solvers but applicable to other multi method problems as well
- In the spirit of the multimethod approach techniques beyond machine learning can also be used.
- **WANTED---more applications!!!**

Acknowledgements

- Thanks to
  - Jin Chen (PPPL) for providing the linear systems from M3D code
  - Raphael Pelossof (Columbia University) for providing codes to draw ROC curves
  - NSF and NGS grants for "Self-Adapting Linear Solver Architecture"

PETSc-FUN3D

- 3D compressible Euler (used in this work; also supports incompressible Navier-Stokes)
- Fully implicit, steady-state
- Developed by D. Kaushik et al.
- Based on FUN3D (developed by W.K. Anderson, NASA Langley)
  - Tetrahedral, vertex-centered unstructured mesh
  - Discretization: 1st or 2nd order Roe for convection and Galerkin for diffusion
- Pseudo-transient continuation
  - backward Euler for nonlinear continuation toward steady-state solution
  - Switched Evolution/relaxation (SER) approach of Van Leer and Mulder



- Newton-Krylov nonlinear solver
  - Matrix-Free (2nd order FD)
  - Preconditioner (1st order analytical)
- Won Gordon Bell prize at SC99; ongoing enhancements and performance tuning