Lighthouse : An Automated Solver Selection Tool

Pate Motter

Department of Computer Science,
University of Colorado, Boulder, CO
pate.motter@colorado.edu

Kanika Sood

Computer and Information Science,
University of Oregon, Eugene, OR
kanikas@cs.uoregon.edu

Elizabeth Jessup

Department of Computer Science,
University of Colorado, Boulder, CO
jessup@cs.colorado.edu

Boyana Norris

Computer and Information Science, University of Oregon, Eugene, OR
norris@cs.uoregon.edu

Abstract

Linear algebra provides the building blocks for a wide variety of scientific and engineering simulation codes. Users of these codes face a world of continuously changing algorithms and high-performance implementations. In this paper, we describe new capabilities of our Lighthouse framework, whose goal is to match specific problems in the area of high-performance numerical computing with the best available solutions. Lighthouse's innovative strategy eliminates intensive reading of documents and automates the process for developing linear algebra software. Lighthouse provides a searchable taxonomy of popular but difficult to use numerical software for dense and sparse linear algebra while providing the user with the best algorithms for a given problem based on machine learning methods. We introduce the design of Lighthouse and show examples of its interface. We also present algorithm classification results for the preconditioned iterative linear solvers in the Parallel Extensible Toolkit for Scientific Computation (PETSc) and the Trilinos library.

Categories and Subject Descriptors D.2.2 [Software Engineering]: Design Tools and Techniques

General Terms linear algebra, mathematical software, machine learning, taxonomy

1. Introduction

Solving large linear systems and computing eigenvalues are fundamental problems in high-performance scientific and engineering computing (HPC). In response to the need for high-performance algorithms, applied mathematics and computer science researchers have created a number of comprehensive numerical software packages that are widely used today. Because of the number and complexity of the algorithms available in these packages, finding the most suitable solution to a particular problem is a nontrivial task, even for experts in numerical methods. For example, for the relatively limited problem of solving a dense system of linear equations, the numerical library LAPACK [25] alone offers over 100 different functions.

Lighthouse [36] is a framework for creating, maintaining and using a taxonomy of available software for highly optimized matrix algebra computations. The taxonomy serves as a guide to HPC application developers seeking to learn what is available for their programming tasks, how to use it and how the various parts fit together. A number of taxonomies exist to aid developers in the translation of linear algebra algorithms to numerical software, e.g., [16, 24, 29]. However, these taxonomies do not provide accessible, comprehensive, and usable interfaces, nor do they supply tools for high quality code production.

Lighthouse is the first framework that offers an organized taxonomy of software components for linear algebra that enables functionality- and performance-based search and generates code templates and optimized low-level kernels. Lighthouse outperforms its alternatives in many ways. It offers functional classification of software as well as a variety of code generation and optimization capabilities. It also provides information about the code in the form of automatically extracted documentation. These characteristics make Lighthouse a multi-purpose tool that automates portions of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SEHPCCSE '15, November 20, 2015, Austin, Texas, United States.
Copyright 2015 ACM ISBN 978-1-4503-4012-0/15/11...\$15.00.
DOI: <http://dx.doi.org/10.1145/2830168.2830169>

the software design and implementation process for HPC applications while also helping to educate developers.

In this paper, we briefly introduce the features of the Lighthouse tool. We then focus on new Lighthouse capabilities in the area of parallel iterative solvers for sparse linear systems in the Portable Extensible Toolkit for Scientific Computation (PETSc) [3–5]. PETSc has been used for modeling in many areas, ranging from acoustics [39] to brain surgery [2] to ocean dynamics [38]. We then discuss our current efforts for measuring accuracy and performance for the Trilinos library. Trilinos, like PETSc is used in a variety of scientific applications, mainly focused around multiphysics simulations [1, 11].

While our initial focus is on numerical linear algebra, the Lighthouse approach is generalizable to any reusable software where multiple solution methods exist. The decades-old approach to reusing software by (1) *requiring users to read documentation* and (2) *use low-level code completion* is not sufficient for most effectively and efficiently using rapidly evolving third-party libraries. Creating and managing the software descriptions and performance attributes in a more systematic way allows for effective categorization of the development environments. Lighthouse is one possible solution to this need.

2. Design and Implementation

The main components of the Lighthouse framework are illustrated in Figure 1. Lighthouse defines its software taxonomy using Django [15] models, which correspond to MySQL databases that store information about the different packages. Currently Lighthouse contains routines from LAPACK, SLEPc [34], and PETSc.

Figure 1. Lighthouse overview

Lighthouse offers three search variants: guided search, advanced search and keyword search. In the guided search, users are asked detailed questions in order to describe the problem they wish to solve with Lighthouse. After answer-

ing all the questions, the user sees exactly one subroutine that corresponds to all the answers provided. The keyword search interface supports keyword-based search of the taxonomy information. In it, Lighthouse supports auto completion of words and spelling correction. Finally, the advanced search is geared toward users who are familiar with the packages.

Lighthouse generates two types of code. First, given a specific search result, Lighthouse provides users with the option of generating a complete program that uses those function(s) correctly, including declarations and initialization of all data structures. Second, Lighthouse provides a separate experimental interface that allows users to define their own high-level linear algebra computations for dense and some sparse linear systems and vectors. Using a MATLAB-like language as input, Lighthouse interfaces with the BTO compiler [7, 35] to generate highly optimized C implementations that can be downloaded and used in larger applications.

3. Taxonomy Search Interfaces

We briefly illustrate the user interfaces of Lighthouse with two examples: one that leads to functionality implemented by LAPACK and another that involves solving a large sparse linear system using PETSc. For each numerical library supported by Lighthouse, we have created decision trees with the questions used in the guided search interfaces. These trees are stored in databases from which the actual interactive Web forms are generated dynamically.

Figure 2 shows an example of a guided search interactive session, in which the user wishes to compute the condition number of a matrix. The interfaces for other LAPACK functionality are structured similarly. After answering a few questions, the user is given a single function in LAPACK (which has dozens of functions that relate to computing condition numbers). The user then can have Lighthouse generate a code template in C or Fortran90, which is a complete program that can be downloaded, compiled, executed, and modified as needed for integration into larger applications.

Figure 3 shows the guided search dialog in the Lighthouse PETSc interface which has a series of questions for the user. It is an interactive system that enables users to generate as well as download PETSc programs for solving sparse linear systems. To begin, the user has the option to upload a coefficient matrix and make Lighthouse compute the matrix's properties or *features*. If the user chooses this option, Lighthouse computes the features and uses them in a machine learning classification process (described in Section 4) to predict a good performing solver for the given system with that coefficient matrix. On the other hand, a user who does not opt to upload the matrix is offered the following options by Lighthouse: a PETSc program to download and use for computing matrix properties or a general PETSc program for solving a linear system.

Figure 2. Computing the condition number of a matrix with LAPACK (guided search)

The code generated by Lighthouse as a result of the PETSc guided search is shown in Figure 4. Lighthouse currently generates code templates in C or Fortran90 for the functions returned in the search. The template is a complete program, including all variable declarations and correct invocation of library functions, many of which take very large numbers of parameters. The template program is easy to modify as it is divided into multiple subprograms.

Figure 3. PETSc guided search interface

Figure 4. PETSc code template resulting from guided search

4. Sparse Linear Solver Classification

Large sparse linear system solution is a fundamental computational step in many scientific and engineering applications. Users generally choose one of many iterative methods for approximating the solution to a system (subsequently, we refer to these methods as *solvers*). Individual solver convergence and performance is highly dependent on the specific input problem characteristics and can be difficult to predict.

Lighthouse automates the performance-based solver selection process by employing machine learning techniques to classify solvers based on a training set generated with

over 1,000 linear systems from the University of Florida matrix collection [31]. We applied several supervised machine learning techniques to help make this decision based on relatively few, easily computed properties of the input system. Supervised learning involves determining a classification based on a set of already classified data. These data are split into training and testing sets. The training set is used to build the classifier and the testing set is used to verify the accuracy of the classifier. For our experiments, this process was repeated 10 times (10-fold cross validation), each time with a different subset as the testing set. Our ultimate goal is to provide a scalable and automated approach for choosing an efficient and accurate solver for a given linear system by classifying linear solvers using several machine learning methods. The success of previous research into performance-based multi-solver methods [8, 17, 20] motivates the addition of solver classification to Lighthouse, which we describe next.

Linear solver classification in Lighthouse is performed as a sequence of steps. Note that this process is generalizable to other problems beyond sparse linear system solution. The features used in building the solver classifiers include matrix properties such as simple (norm-like) quantities, variance (heuristics estimating how different matrix elements are), normality (estimates of the departure from normality), structure (nonzero structure properties), and spectral properties. Because computing so many matrix properties (in the case of PETSc 68) while an application is executing is too costly, we also performed feature set reduction, removing the features that do not contribute significantly to the process of deciding the best solver for a given linear system. Our primary software for this work was Weka [19], which is a collection of machine learning algorithms for data mining tasks. We used Weka’s RemoveUseless filter and attribute evaluators to generate the reduced features for PETSc and Trilinos. The evaluator determines a method to assign a worth to each subset of features and rank the features, allowing us to discard those that do not contribute much to the classification. Feature set reduction brings down the overall cost of the process for building *and* using the classifiers. The next step involves building classifiers by using several machine learning methods in Weka: BayesNet [9], k-nearest neighbor [13], Alternate Decision Trees [18], Random Forests [10], J48 [32], Support Vector Machines (SVM) [12], Decision Stump [22] and LADtree [21] methods. We compared the performance of these methods to select the one that produces the best (most accurate) solver classification.

4.1 Classification Results

We calculate the results based on two parameters : the performance accuracy of the classifiers and the time taken to construct each classifier, which can vary by orders of magnitude between different methods. The classification results were evaluated using 10-fold cross validation. The performance accuracy for PETSc and Trilinos is shown as radar charts in Figure 5, 6 and 7. An accuracy of 100% indi-

cates a classifier that predicts the good classifiers correctly each time. An accuracy of 0% indicates a classifier that fails to predict good classifiers in each case. In these charts, we show the performance accuracy for all features as well as both reduced feature sets for comparison.

The reduced feature sets shown in Tables 1 and 3 for PETSc and Trilinos, respectively, are different because our experiments indicate that using same sets for both PETSc and Trilinos drops the accuracy for one of them. Figure 5 and 6 include results from features computed using Anamod. Figure 7 have results for features extracted from Trilinos. These figures demonstrate that better performance is achieved with Trilinos-generated results than with Anamod-generated ones.

4.1.1 PETSc

To generate the input dataset used by the machine learning methods to classify solvers, we measured the performance of several solver and preconditioner combinations in PETSc version 3.5.3 on a Blue Gene/Q supercomputer with a total of 154 solver-preconditioner configurations.

For each input, we computed the features of the matrix, then solved the system using a specific solver configuration and measured the execution time, resulting in 4,648 data points. The threshold used for labeling this dataset, both for PETSc and Trilinos was 35%, which means the solvers with times within 35% of that of the best solver time were labeled as “good”. The threshold value was chosen by generating the classification results with threshold values varying from 1% to 45% and choosing the value that gave the best performance accuracy. Next, these data were used to build solver classifiers by using different machine learning methods. In Lighthouse, we focus only on the true positive rate (TPR) which is the probability that the classifier predicts a good entry as good. It is computed using the usual true positive rate formula: $TPR = TP / P = TP / (TP + FN)$, where P is the actual number of positive instances, i.e., solvers labeled as good.

We performed experiments with all features and with reduced features sets. Our experiments suggest that size-based features have minimal impact on the accuracy of the best classification. For PETSc, Reduced Set 1 ($RS1_P$) has 8 features which are shown in Table 1. Reduced Set 2 ($RS2_P$), also shown in Table 1, is generated by removing size-dependent features from $RS1_P$. Among all machine learning methods we used, BayesNet produced the best TPR solver accuracy of 87.6% with 68 input features. Using only eight computationally inexpensive features, the BayesNet-based classifier predicted good (well-performing) solvers correctly 86.9% of the time. Figure 5 shows the true positive prediction accuracy of several of the machine learning methods we tested for the full and reduced feature sets.

In addition to the accuracy of the classifier, we also considered the time it took to build the classifier which can be seen in the Table 2. The BayesNet method, which has the

Table 1. Reduced feature sets for PETSc

Feature name	Reduced Feature Set 1 ($RS1_P$)	Reduced Feature Set 2 ($RS2_P$)
avg-diag-dist	X	X
nnz	X	
norm1	X	X
col-variability	X	X
min-nnz-per-row	X	X
row-variability	X	X
n-nonzero-diags	X	
kappa	X	X

Figure 5. Machine learning method accuracy comparison for PETSc using Anamod-calculated features ($RS1_P$, $RS2_P$).**Table 2.** Time taken to construct the classifier for PETSc-calculated properties (in secs).

ML Method	T_{All}	T_{RS1_P}	T_{RS2_P}
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

highest accuracy, took a fraction of a second to build the PETSc solver classifier on an Intel Core i5 MacBook Pro. The slowest method was LibSVM, taking over a minute to build the classifier.

4.1.2 Trilinos

Generating the input dataset using Trilinos is similar in approach to the methods used for PETSc. Trilinos consists of packages that each perform a specific linear algebra task.

This structure allows for several approaches to solving linear algebra problems depending on which packages are chosen. From Trilinos version 11.12, we are using Tpetra [37] for data objects like matrices, Ifpack2 [23] for preconditioners, and Belos [6] for iterative solvers. Data from Trilinos, both the properties and solver-preconditioner timings, were recorded from single-node (12-core) executions on the University of Colorado’s Janus supercomputer [30].

To gather the timing data for each matrix, we performed solves using 66 solver-preconditioner pairs from the Belos and Ifpack2 packages. To determine the features of each matrix, we used two similar but unique approaches. The first feature set was calculated using Anamod, while the second was calculated using Trilinos itself. The main difference between them is that Anamod runs serially while Trilinos supports parallel execution. The Trilinos feature-set we calculate is currently a subset of the features available in Anamod.

We first used the Anamod-calculated features to perform classification on the same matrices used in the PETSc classification discussed in 4.1.1 to generate 14,348 data points. For the Anamod-calculated Reduced Set 1 ($RS1_A$) we have 8 features which are shown in Table 3. Reduced Set 2 ($RS2_A$), is generated by removing size-dependent features from $RS1_A$.

The best TPR solver accuracy with all 68 of the Anamod-calculated features was 71.8%. With $RS1_A$ the best TPR was 75.0% using Random Forest (100 trees). With $RS2_A$, the best accuracy of 73.1% was given by Random Forest (100 trees) as well. Table 4 shows the timing results to build the classifiers. LibSVM takes many times longer to build than Random Forest, and it is the worst performing model for predictions.

We performed the same steps for classification using the features calculated natively with Trilinos rather than Anamod. This feature set is a subset of those found in Anamod and currently contains 38 of its 68 features. The total number of data points available for this training set was 39,388 using the various solver-preconditioner pairs available within Trilinos. The best TPR for all 38 features was 93.0% via VFI [14]. For the Trilinos-calculated Reduced Set 1 ($RS1_T$) we have 8 features which are shown in Table 5. Reduced Set 2 ($RS2_T$), is generated by removing size-dependent features from $RS1_T$.

Using the reduced feature sets found in Table 5 we achieved a TPR of 92.0% and 92.5% for the Trilinos-calculated feature sets, $RS1_T$ and $RS2_T$, respectively, via VFI. Table 6 shows the timing results to build the classifiers for the Trilinos-based feature set. VFI takes the least amount of time to build, and is the best performing model for predictions. Figures 6 and 7 show the true positive prediction accuracy of the machine learning methods tested for the Anamod and Trilinos based feature sets respectively.

Our work of identifying solver-preconditioner pairs with PETSc and Trilinos is not currently available to users of

Lighthouse, but results with a high accuracy model for PETSc packages and Trilinos’ Belos iterative solver package and the Ifpack2 preconditioner package show promising results for Lighthouse. We will increase the number of features being calculated to potentially improve selection accuracy. Currently for Trilinos, we only calculate 38 features whereas Anamod calculates 68. After the larger feature set has been used for classification, the resulting model can be added to Lighthouse allowing users to find solvers and preconditioners for PETSc and Trilinos more effectively. Further work will need to be done to add additional Trilinos packages beyond Belos and Ifpack2.

Table 3. Reduced feature set for Trilinos using Anamod-based matrix properties.

Feature name	Reduced Feature Set 1 ($RS1_A$)	Reduced Feature Set 2 ($RS2_A$)
avgdistfromdiag	X	X
blocksize	X	X
left-bandwidth	X	X
col-variability	X	X
diagonal-dominance	X	X
row-variability	X	X
nnzeros	X

Figure 6. Machine learning method accuracy comparison for Trilinos using Anamod-calculated features ($RS1_A$, $RS2_A$).

4.2 Using the Classification Results to Select Solvers

The process discussed in the previous section helps us to answer the question of which classifier should be chosen for a given linear system. Once we have that classifier, we perform experiments with it to choose the solver configurations that are likely to perform well. The interface of Lighthouse to PETSc and Trilinos is presently in the development phase.

Table 4. Time taken to construct the classifier for Trilinos with Anamod-calculated properties (in secs).

ML Method	T_{All}	T_{RS1_A}	T_{RS2_A}
LibSVM	62.88	23.13	23.17
RF	7.90	3.16	2.63
BayesNet	0.22	0.07	0.05
knn	0.001	0.001	0.001
ADTree	4.31	0.43	0.32
J48	0.64	0.09	0.09

Table 5. Reduced feature set for Trilinos using Trilinos-computed matrix properties.

Feature name	Reduced Feature Set 1 ($RS1_T$)	Reduced Feature Set 2 ($RS2_T$)
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

Table 6. Time taken to construct the classifier for Trilinos with Trilinos-calculated properties (in secs).

ML Method	T_{All}	T_{RS1_T}	T_{RS2_T}
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

Once complete, Lighthouse will compute a reduced feature set for a system to upload to its interface by a user and will return a list of solver configurations that are likely to perform well.

Lighthouse is a tool that targets users from various backgrounds, including researchers and students. There might be cases where a user is interested in not just solving a problem but also in understanding the behavior of various solvers on different kinds of linear systems. In such cases, it will be helpful to not just deliver the best solver to the user but also to provide details about other good solvers. Keeping this in mind, we plan also to give the information about the top 10 solvers that were labeled as “good” for PETSc and Trilinos. The user will be able to easily view or download these solvers from the interface easily.

5. Evaluation

As Lighthouse continues to grow, we plan to apply formal methods of user testing from human computer interaction (e.g., [26–28, 33]) to evaluate its usability. For example, we will employ techniques for interface developers, such as cognitive walkthrough [33] and heuristic evaluation [27, 28],

Figure 7. Machine learning method accuracy comparison for Trilinos using Trilinos-calculated features ($RS1_T, RS2_T$).

and we will continue to use those directed at users, such as thinking aloud [26].

To date, we have carried out thinking aloud evaluations of Lighthouse in graduate courses at the University of Colorado Boulder. The clients had education and expertise in fields of engineering (aerospace, electrical, civil), mathematics, computer science, and computational science and varying expertise in linear algebra and programming. These clients discovered a number of usability issues and also made very good suggestions for improvement of Lighthouse. These preliminary evaluations aptly demonstrate the value of user feedback.

6. Conclusions and Future Work

Guidelines for creating a reusable component include automation, ease of use, and good documentation. However, reducing the dependence on documentation makes it easier for the user to use that software and saves a lot of time. With Lighthouse, we illustrate this idea and make a first effort to improve the conventional way of building numerical software. Because there are thousands of valid linear solver configurations, we have added to Lighthouse a machine learning-based solver classification. It can substantially improve developer productivity because there is no need to experiment with various solvers in an ad hoc fashion, and it can improve application performance. At present, Lighthouse for PETSc provides an efficient navigation system that allows users to generate, download and extend PETSc applications for solving large sparse linear systems. The addition of Trilinos into Lighthouse is underway and already shows significant results in terms of accuracy and speed.

Future work includes extending the present Lighthouse interface to automate more fully the solver selection process. More automation will enable the regular regeneration of the classifiers as more data are collected from different applica-

tions, both user-provided and our own. Our current analysis considers only sequential runs; we aim to add runs in parallel as well. More work needs to be conducted in order to incorporate our Trilinos results into Lighthouse’s user interface. Ifpack2 and Belos are only two of Trilinos’ many packages and only represent a small portion of problem types. In order to increase functionality for more problems, more Trilinos packages must be considered and measured. Other future goals involve expanding the number of iterative solvers packages covered by the framework and improving the performance accuracy. Currently Lighthouse does not factor in the scale of computational resources available. Scalability needs to be incorporated so that recommendations can be made across various machine sizes. Solver selection can also be integrated directly into the PETSc and Trilinos libraries, which provide extensible interfaces that enable this. As Lighthouse continues to support more options and users the maintainability will become more difficult and will therefore need to be adjusted as we expand.

Acknowledgments

This work is supported by the U.S. Department of Energy Office of Science (Contract No. DE-SC0013869) and by the National Science Foundation (NSF) awards CCF-1219089, CCF-155063 and CCF-1550202. This work utilized the Janus supercomputer, which is supported by the National Science Foundation (award number CNS-0821794) and the University of Colorado Boulder. The Janus supercomputer is a joint effort of the University of Colorado Boulder, the University of Colorado Denver and the National Center for Atmospheric Research.

References

- [1] Albany. <https://github.com/gahansen/albany/>. <https://github.com/gahansen/Albany/>, 2015.
- [2] Y. Ataseven, Z. Akalın-Acar, C. Acar, and N. G. Gençer. Parallel implementation of the accelerated BEM approach for EMSI of the human brain. *Medical & Biological Engineering & Computing*, 46(7):671–679, 2008.
- [3] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.
- [4] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, S. Zampini, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.6, Argonne National Laboratory, 2015. URL <http://www.mcs.anl.gov/petsc>.
- [5] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F.

- Smith, S. Zampini, and H. Zhang. PETSc Web page, 2015. URL <http://www.mcs.anl.gov/petsc>.
- [6] E. Bavier, M. Hoemmen, S. Rajamanickam, and H. Thornquist. Amesos2 and Belos: Direct and iterative solvers for large sparse linear systems. *Scientific ...*, pages 1–25, 2012. URL http://www.sandia.gov/~srajama/publications/Amesos2_Belos_2012.pdf<http://iospress.metapress.com/index/P055118195457462.pdf>.
- [7] G. Belter, E. R. Jessup, I. Karlin, and J. G. Siek. Automating the generation of composed linear algebra kernels. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, 2009. ACM. ISBN 978-1-60558-744-8. .
- [8] S. Bhowmick, V. Eijkhout, Y. Freund, E. Fuentes, and D. Keyes. Application of alternating decision trees in selecting sparse linear solvers. In *Software Automatic Tuning*, pages 153–173. Springer, 2010.
- [9] C. Bielza and P. Larrañaga. Discrete Bayesian network classifiers: A survey. *ACM Comput. Surv.*, 47(1):5:1–5:43, July 2014. ISSN 0360-0300. . URL <http://doi.acm.org/10.1145/2576868>.
- [10] L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, Oct. 2001. ISSN 0885-6125. . URL <http://dx.doi.org/10.1023/A:1010933404324>.
- [11] CoolFluid. <http://coolfluid.github.io/>. <http://coolfluid.github.io/>, 2015.
- [12] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- [13] P. Cunningham and S. J. Delany. k-nearest neighbour classifiers. *Multiple Classifier Systems*, pages 1–17, 2007.
- [14] G. Demiröz and H. A. Güvenir. Classification by voting feature intervals. In *Proceedings of the 9th European Conference on Machine Learning*, ECML '97, pages 85–92, London, UK, 1997. Springer-Verlag. ISBN 3-540-62858-4. URL <http://dl.acm.org/citation.cfm?id=645325.649678>.
- [15] Django. <https://www.djangoproject.com/>, 2015.
- [16] J. Dongarra. Freely available software for linear algebra on the web. <http://www.netlib.org/utk/people/JackDongarra/la-sw.html>, 2015.
- [17] P. R. Eller, J.-R. C. Cheng, and R. S. Maier. Dynamic linear solver selection for transient simulations using machine learning on distributed systems. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1915–1924. IEEE, 2012.
- [18] Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 124–133, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2. URL <http://dl.acm.org/citation.cfm?id=645528.657623>.
- [19] 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.
- [20] A. Holloway and T.-Y. Chen. Neural networks for predicting the behavior of preconditioned iterative solvers. In *Computational Science-ICCS 2007*, pages 302–309. Springer, 2007.
- [21] G. Holmes, B. Pfahringer, R. Kirkby, E. Frank, and M. Hall. Multiclass alternating decision trees. In *Machine learning: ECML 2002*, pages 161–172. Springer, 2002.
- [22] W. Iba and P. Langley. Induction of one-level decision trees. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 233–240, 1992.
- [23] Ifpack2. <https://trilinos.org/packages/ifpack2/>. <https://trilinos.org/packages/ifpack2/>, 2015.
- [24] E. Jessup, B. Bolton, B. Enosh, F. Ma, and T. Nguyen. LAPACK Internet Interface and Search Engine. <http://www.cs.colorado.edu/lapack>, 2008.
- [25] LAPACK - Linear Algebra PACKage. <http://www.netlib.org/lapack/>, 2015.
- [26] C. Lewis. *Using the "thinking-aloud" method in cognitive interface design*. IBM TJ Watson Research Center, 1982.
- [27] J. Nielsen. How to conduct a heuristic evaluation. http://www.useit.com/papers/heuristic/heuristic_evaluation.html, 2007.
- [28] J. Nielsen. Ten usability heuristics. http://www.useit.com/papers/heuristic/heuristic_list.html, 2007.
- [29] NIST. Guide to Available Mathematical Software. <http://gams.nist.gov>, 2015.
- [30] U. of Colorado. Compute resources - janus. <https://www.rc.colorado.edu/support/user-guide/compute-resources.html#janus>, 2015.
- [31] U. of Florida. The University of Florida Sparse Matrix Collection. <http://www.cise.ufl.edu/research/sparse/matrices/>, 2015.
- [32] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.
- [33] J. Rieman, M. Franzke, and D. Redmiles. Usability evaluation with the cognitive walkthrough. http://www.acm.org/sigs/sigchi/chi95/Electronic/documents/tutors/jr_bdy.htm, 2004.
- [34] Scalable Library for Eigenvalue Problem Computations (SLEPc). <http://www.grycap.upv.es/slep/>, 2015.
- [35] J. G. Siek, I. Karlin, and E. R. Jessup. Build to order linear algebra kernels. In *Workshop on Performance Optimization for High-Level Languages and Libraries (PO HLL 2008)*, pages 1–8, April 2008.
- [36] The Lighthouse Project. <http://lighthousehpc.github.io/lighthouse/>, 2015.
- [37] Tpetra. <https://trilinos.org/packages/tpetra/>. <https://trilinos.org/packages/tpetra/>, 2015.
- [38] A. D. T. van Scheltinga, P. G. Myers, and J. D. Pietrzak. A finite element sea ice model of the Canadian Arctic Archipelago. *Ocean dynamics*, 60(6):1539–1558, 2010.

- [39] J. Wang, Y. Wang, W.-k. Hu, and J.-k. Du. The high performance of finite element analysis and applications of surface acoustic waves in finite elastic solids. In *Piezoelectricity, Acoustic Waves, and Device Applications, 2008. SPAWDA 2008. Symposium on*, pages 66–71. IEEE, 2008.