SAND2006-5746C

# Modern Machine Learning for Automatic Optimization Algorithm Selection

Patricia D. Hough, Pamela J. Williams
Computational Sciences and Mathematics Research Department, Sandia National Laboratories, PO Box 969 MS 9159, Livermore, California 94551-0969, USA, {pdhough@sandia.gov, pwillia@sandia.gov}

**Abstract:** Optimization software is commonly used to solve simulation-based problems such as optimal design and control, model parameter estimation, and best/worst-case scenario identification. While the value of such software is widely recognized, user feedback indicates that these tools are difficult for non-experts to use. In particular, users are unfamiliar with the details of the plethora of available optimization algorithms and thus do not know which algorithm is appropriate for the problem at hand. In this paper, we will present an approach based on modern machine learning techniques for automatically selecting an optimization algorithm to solve a given problem. We will discuss the feature sets used to characterize the optimization problems and algorithms, and we will review the metrics used to evaluate optimization algorithm performance. Finally, we will present the results from our initial studies with the CUTEr optimization test set and discuss the challenges of generalizing the methodology to simulation-based optimization problems.

## 1 Motivation

Optimization software is commonly used to solve simulation-based problems such as optimal design and control, model parameter estimation, and best/worst-case scenario identification. While the value of such software is widely recognized, user feedback indicates that these tools are difficult for non-experts to use. In particular, users are unfamiliar with the details of the plethora of available optimization algorithms and thus do not know which algorithm is appropriate for the problem at hand. While the user has a great deal of intuition regarding the underlying simulation, he or she usually must consult an optimization expert to determine the best algorithm or make a guess based on documentation written by optimization developers for optimization developers. Our goal is to address these concerns by developing technology to automatically choose an appropriate optimization algorithm to solve a given problem.

To state the problem more formally, given 1) a canonical representation of an optimization problem, 2) a set of optimization algorithms, and 3) a performance metric, determine which algorithm is best for solving the problem relative to the performance metric. This problem is a specific instance of the more general algorithm selection problem and as such, has no analytical solution [15]. Therefore, our approach is based on machine learning algorithms coupled with computational experiments.

The remainder of the paper is organized as follows. In Section 2, we will briefly mention related work in the area of automatic algorithm selection. Section 3 provides a summary of the machine learning algorithms was have considered to date, and we discuss the feature sets and metrics needed to make use of machine learning in Section 4. We will present the results from our initial studies with the CUTEr optimization test set in Section 5. Finally, in Section 6, we discuss the next steps and the challenges of generalizing the methodology to simulation-based optimization problems.

## 2 Related Work

Algorithm selection is a problem that is pervasive throughout the realm of computational mathematics; however, we are aware of very little work in the area of automatic algorithm selection. Nonetheless, there are a number of recent efforts that show promising results. Perhaps the most established of these efforts is PYTHIA [19] and its successors. In this work, the authors use Bayesian networks and neural networks to choose and appropriate numerical solver for elliptic partial differential equations. Bhowmick, *et al.* [2] have recently published work in which they use decision trees and boosting to select sparse linear solvers for systems arising in various physics applications.

With regard to choosing optimization algorithms, there are a couple of notable tools intended to aid the user in choosing an optimization algorithm. Those are the NEOS Server for Optimization (URL: http://www-neos.mcs.anl.gov) and Decision Tree for Optimization Software web page published by Mittelman (URL:

http://plato.asu.edu/guide.html). While they provide a nice categorization of optimization problems and approaches, they do not provide an automated selection capability. In addition, Ong studied a range of machine learning approaches for choosing and optimization algorithm in his Ph. D. thesis [12]. His work was focused on scenarios where the user is solving the same, or a very similar, problem over and over again as opposed to solving a diverse set of problems.

# 3 Summary of Selected Machine Learning Algorithms

While there is a wide range of machine learning algorithms to choose from, we focus our proof of concept on ensemble methods. Ensemble classifiers have become popular in modern machine learning. The basic approach consists of constructing a set of classifiers, training each on a subset of the training data. New data points are then classified by taking a weighted vote of the individual predictions of each classifier. One particularly nice featuer of ensemble classifiers is that they are often more accurate than a single classifier. A nice survey of ensemble methods, their features, and their performance can be found in [5].

In the following sections we describe the two types of base classifiers we consider in the context of ensemble methods.

## 3.1 Decision Trees

There are a number of examples of ensemble classifiers based on decision trees, including ([6], [10]). As a brief review, a decision tree algorithm recursively partitions the feature space. Each partition is determined in such a way as to achieve some level of purity in the resulting subsets while not making those subsets so small as to be meaningless for prediction purposes. Each partition is encoded as a branch in the decision tree. One example of a decision tree is depicted below.

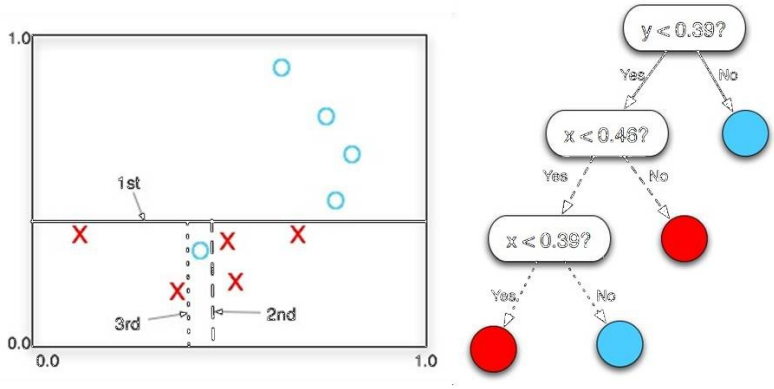


Figure 1: This figure depicts one example of a decision tree. Each partition in feature space is encoded as a branch in the tree. (Figure courtesy of Philip Kegelmeyer.)

## 3.2 Support Vector Machines

Support vector machines (SVMs) were developed by ([3], [4], [17]). Using support vector machines for pattern recognition is a two-stage process, which consists of training data and classifying testing data. Training support vector machines requires the solution of a convex quadratic programming problem that depends on the number of examples in the dataset. In general, SVMs operate by finding a discriminant function, which separates a multidimensional data set into two or more classes. SVMs have been applied successfully to a wide variety of problems such as handwritten digit recognition, vehicle identification, and face recognition [14].

# 4 Feature Set for Optimization

In order to effectively use machnie learning algorithms, and attribute set relevant to the problem space is required. In our work, optimization problem, algorithm, and runtime environment features comprise the attribute set. The list of features that is used by an optimization expert to describe problems and algorithms is extensive, some are not easily quantified or verified. Thus, we initially focus on a reduced set of features that are easily defined and readily accessible. Table 1 contains the list of features of interest in this work. For similar reasons, we focus on a reduced set of performance metrics and treat them as independent in our initial proof of concept. Those metrics are also shown in Table 1.

| Problem Features | Algorithm Features | Runtime Environment Features | Performance Metrics |
|---|---|---|---|
| • smoothness<br>• convexity<br>• availability of analytic derivatives<br>• degree of nonlinearity<br>• types, number of variables<br>• types, numbers of constraints | • means of globalization<br>• method of gradient approximation<br>• method of Hessian approximation<br>• constraint handling<br>• parallelization<br>• algorithmic parameters | • processor type<br>• operating system<br>• compilers (and optimizations)<br>• number of processors<br>• availability | • number of function evaluations<br>• optimal function values<br>• stopping criteria<br>• solution robustness |

Table 1: This table shows the list of features on which we initially focus. They are categorized according to whether they describe optimization problem, algorithm, or runtime environment.

# 5 Numerical Experiments

We conducted a range of numerical experiments to demonstrate proof of concept for our machine learning-based approach to automatic algorithm selection. The high-level concept is illustrated in Figure 5. Starting with a set of optimization problems and a set of optimization algorithms, we executed all of the algorithms on all of the problems to determine which performed best according to our metrics. The data we generated was used to train the machine learning algorithms. Given these trained classifiers, we then used them to predict the best algorithms for a new set of problem instances. More details on the various components follow.

The set of optimization test problems we chose are those in the Common Unconstrained Testing Environment, revisited (CUTEr) [8]. The CUTEr test problems are often used by optimization developers to stress test their algorithms due to the quantity and diversity of the available problems. Due to the diversity of this set of problems, we were able to ensure coverage of the problem feature set listed in Table 1. Our test set contains 776 CUTEr problems excluding problems with more than 1025 combined variables and/or constraints.

Optimization algorithms were taken from, OPT++ [11], a library of nonlinear optimization algorithms written in C++. The focus of the library is on robust and efficient algorithms for problems in which the function and constraint evaluations require the execution of an expensive computer simulation. OPT++ includes the classic Newton methods, conjugate gradient, generating set search, limited memory BFGS method, parallel direct search, a trust-region parallel direct search hybrid, a nonlinear interior-point method, and a wrapper to NPSOL [7]. In Table 2, we summarize the problem features of each class of algorithm can address. Note that for each feature, there are at least two algorithms from which to choose.

To generate experimental data for our machine learning algorithm, we created an OPT++ interface to the (CUTEr) test set and used PERL scripts to postprocess the data into a format that could be used as input to machine learning software. All experiments were conducted on a 750 MHz Pentium III processor

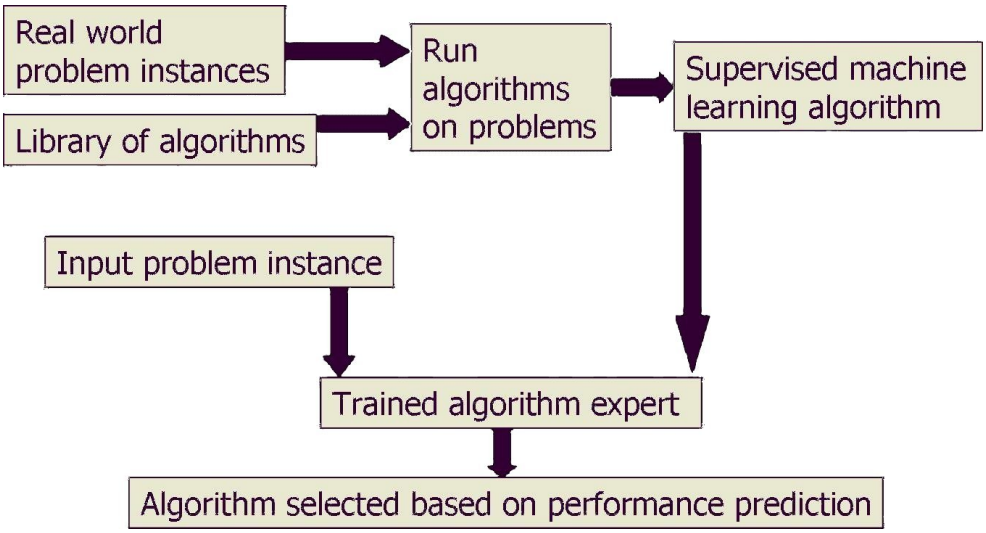


Figure 2: This figure illustrates the high-level concept behind our approach. Computational experiments in which we solve a set of optimization problems with a range of algorithms generate training data for machine learning algorithms. The trained classifiers are then used to choose an algorithm for a new problem instance.

| | No Derivatives | Derivatives | No Constraints | Bound Contraints | Linear Constraints | Nonlinear Constraints |
|---|---|---|---|---|---|---|
| CG | | X | X | | | |
| GSS/PDS | X | | X | X | | |
| Classic Newton | X | X | X | X | X | |
| Interior Point | X | X | X | X | X | X |
| NPSOL | X | X | X | X | X | X |

Table 2: This table depicts how the optimization algorithms in OPT++ cover the problems feature set.

IBM Thinkpad running the Red Hat Linux operating system. Results were computed using OPT++ version 2.2 with default algorithmic-specific parameter settings.

The two machine learning software packages we used to classify the generated data were OpenDT version 0.52 [1] and use SvmFu version 3.1 [16]. OpenDT is an open source ensemble classifier software package. In particular, it uses ensembles of decision trees and supports a variety of feature types and voting schemes. SvmFu is an open source support vector machine package that can handle multiple as well as binary class categorization.

With OpenDT, we tested ensembles of varying size using C4.5 and INFOGAIN style splitting for the decision trees. We also experimented with bagging, boosting, and random forests for ensemble construction. We used 75% of 835 examples to train the classifiers and the remaining data to test their predictive capability. A confusion matrix that exemplifies our results is shown in Table 3. In particular, it shows results for the number of function evaluations metric and illustrates that we achieved 81% accuracy. This result was insensitive to the size, splitting, and ensemble construction choices. The optimal function values metric demonstrated more sensitivity and ranged for 65-75% accuracy. In cases when the OpenDT classifiers did not choose the correct algorithm, the algorithm chosen was viable from the perspective of an optimization expert.

With SvmFu, we compared two multiple class categorization approaches, one-vs-all (or one-vs-rest) and one-vs-one. Assume that $k$ classes are represented in the dataset. As its name suggests, the one-vs-all method performs binary classification to construct a separating hyperplane between class $i$ and the remaining $k-1$ classes. The one-vs-all method yields $k$ classifiers and corresponding discriminant functions $f_i(x)$. The point

| True/Assigned | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | 2 | | | | | | | | |
| 2 | | 31 | | | 4 | | | | | | | | |
| 3 | | | | | | | | | | | | | 1 |
| 4 | | | | 26 | | | | | | | | | |
| 5 | | 2 | | | 4 | | | | | | | | |
| 6 | | | | | | 9 | | | | | | | 3 |
| 7 | | | | | | | 12 | | | | | | |
| 8 | | 4 | | | | | | 2 | | | | | |
| 9 | | | | | | | | | 1 | | | | 3 |
| 10 | | | | | | | 3 | | | 18 | | | |
| 11 | | 1 | | | | | | 1 | | | 24 | | |
| 12 | | | | | | | | | | | | | 8 |
| 13 | | | | | | 5 | 3 | | | | | | 42 |

Table 3: This table shows the confusion matrix generated by OpenDT for the number of function evaluations metric. The rows represent the "true" best algorithm, and the columns represent the algorithm assigned by the classifier. The desired outcome is large entries on the diagonal.

$x$ belongs to the class that maximizes $f_i(x)$. The one-vs-one method computes $\frac{k(k-1)}{2}$ models for pairwise binary classification. Without loss of generality, assume that we want to categorize examples into either class $i$ or $j$. Given $\frac{k(k-1)}{2}$ discriminant functions, membership in class $i$ is determined by a voting algorithm.

Our training set consists of 14 classes of 984 examples with 11 features. The first test set contains 646 examples and the second set 364 examples. In both cases, the prediction accuracy was 97.4%.

# 6 Future Work

Future work will focus on increasing the complexity of the problems that we can address. One aspect of this includes expanding the feature set to include algorithmic parameters such as finite difference approximation and globalization strategy as well as runtime features such as operating system, compiler, linker and number of processors. Moreover, we will introduce algorithm stopping criteria and robustness of the solution as additional performance metrics.

As we expand the list of features, we will begin to include features that are uncertain. Examples of uncertain features include smoothness and convexity of the optimization problem. Managing these uncertainties will become increasingly important as we move away from standard test sets and begin to work with simulation-based problems. We will explore the use of Bayesian networks to handle the uncertainties and develop the means to integrate them into decision tree-based ensemble classifiers.

To date, we have treated the performance metrics as independent. Furthermore, in our studies, we found that there are numerous examples with conflicting algorithm labels in the training data. Thus, there are open question regarding how to break ties and how to consider tradeoffs betweeen. We will investigate the use of heterogeneous data fusion techniques in the context of ensemble classifiers to address these questions. Our work will be based on that described in [9].

One final aspect that we will address is the computational expense associated with generating training data for simulation-based optimization problems. Unlike for the CUTEr test problems, we will be unable to conduct an extensive set of computational experiments to generate the training data. Instead, we will take an approach based on statistically sampling the simulation-based problems, constructing computationally inexpensive surrogates, and applying the methodology used for the CUTEr test problems to the surrogates. As part of this process, it will be necessary to characterize the surrogates according to our feature set. To accomplish this, we will investigate the use of Dr. AMPL [13] in conjunction with input from the user.

**Acknowledgements:** The authors would like to acknowledge Philip Kegelmeyer for generously sharing his expertise in machine learning, particularly in the area of ensemble classifiers and the OpenDT software.

# References


[1] Banfield, R. E. OpenDT. Available online at http://opendt.sourceforge.net.

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

[3] Boser, B, Guyon, I., and Vapnik, V. A training algorithm for optimal margin classifiers. *Fifth Annual Workshop on Computational Learning Theory*, pp. 144-152, 1992.

[4] Cortes, C. and Vapnik, V. The nature of statistical learning theory. *Machine Learning* 20:273-297, 1995.

[5] Dietterich, T. G. Ensemble Methods in Machine Learning. In Kittler, J. and Roli, F., editors, Proceedings of the First International Workshop on Multiple Classifier Systems, Cagliari, Italy, pp. 2 - 15, 2000.

[6] Dietterich, T. G. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting and Randomization. *Machine Learning*, 2000.

[7] Gill, P. E., Murray, W., Saunders, M. A., and Wright, M. H. User's Guide for NPSOL(Version 4.0): A Fortran Package for Nonlinear Programming. Technical Report SOL-86-2, System Optimization Laboratory, Stanford University, Stanford, CA, 1986.

[8] Gould, N. I. M., Orban, D., and Toint, P. L. CUTEr and SifDec: A constrained and unconstrained testing environment, revisited. *ACM Transactions on Mathematical Software (TOMS)* Vol. 29(4), pp.373 - 394, 2003.

[9] Gray, G., Williams, P., Sale, K. and Gay, D. Enhancing Information Extraction by Applying Ensemble Classification to Disparate Data Sets, Technical Report SAND2006-0866C, Sandia National Laboratories, Livermore, CA.

[10] Kwok, S. W. and Carter, C. Multiple Decision Trees. In Schachter, R. D, Levitt, T. S, Kannal, L. N., and Lemmer, J. F, editors, *Uncertainty in Artificial Intelligence 4*, pp. 327 - 335, Elsevier Science, Amsterdam.

[11] Meza, J. C., Oliva, R. A., Hough, P. D. and Williams, P. J. OPT++: An Object Oriented Toolkit for Nonlinear Optimization. *ACM Transactions on Mathematical Software (TOMS)*, to appear.

[12] Ong, Y. S. Artificial Intelligence Technologies in Complex Engineering Design. Ph. D. Thesis, University of Southampton, 2002.

[13] Orban, D. and Fourer, B. Dr. Ampl: A Meta Solver for Optimization. Available online at http://www.gerad.ca/ orban/drampl.

[14] Osuna, E., Freund, R., and Girosi, F. Training Support vector machines: an Application to Face Detections. Proceeding of CVPR'97, June 17-19, 1997, Puerto Rico.

[15] Rice, J. R. The Algorithm Selection Problem. In M. V. Zelkowitz, editor, *Advances in Computers*, Volume 15, pp. 65 - 118, 1976.

[16] Rifkin, R. SvmFu. http://five-percent-nation.mit.edu/SvmFu/ 2000.

[17] Vapnik, V. Estimation of dependencies based on empirical data. Nauka, Moscow, 1979.

[18] Wachter, A. and Biegler, L. T. On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming *Mathematical Programming*, Vol. 106 (1) pp. 25-57, 2006.

[19] Weerawarana, S., Houstis, E. N., Rice, J. R., Joshi, A., and Houstis, C. E. PYTHIA: A Knowledge-Based System to Select Scientific Algorithms. *ACM Transactions on Mathematical Software*, Volume 22 (4), pp. 447 - 468, 1996.