

Inductive Learning for Engineering Design Optimization

Mark Schwabacher Haym Hirsh Thomas Ellman

Department of Computer Science

Hill Center for the Mathematical Sciences

Busch Campus

Rutgers, The State University of New Jersey

Piscataway, New Jersey 08855

908-445-{3213, 4176, 4184}

FAX 908-445-5691

{schwabac, hirsh, ellman}@cs.rutgers.edu

May 1, 1995

Abstract

We applied inductive learning to a problem, engineering design optimization, for which the applicability of inductive learning is not immediately obvious. In this paper we describe how we were able to formulate two pieces of the optimization problem as inductive learning problems, and we describe some of the lessons that we learned in the process.

Keywords

engineering, design, decision tree induction, numerical optimization.

1 Introduction

The High-Performance Computing and Design (HPCD) project has attempted to apply various advanced computing technologies to the design of complex engineering artifacts[HPCD, 1995]. As part of this project, we are exploring the application of inductive learning to numerical optimization of complex engineering artifacts. It was not immediately obvious that inductive learning would be applicable to this problem. The problem with which we were faced – to produce a good design for a given goal – did not easily fit into the set of classification-type problems usually attacked using inductive learning. We believed that inductive learning might be appropriate, however, because of the existence of a library of past design that could be used as training data, and because of the existence of a simulator and optimizer that could be used to generate additional training data as needed.

We discovered that two pieces of the numerical optimization problem – prototype selection and reformulation selection – could be formulated as inductive learning problems, and that

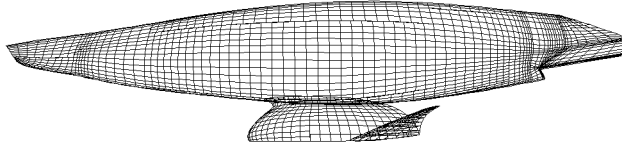


Figure 1: The Stars and Stripes '87.

using inductive learning for these pieces of the problem resulted in improved performance for the overall optimization problem.

In Section 2 we describe the domain of racing-yacht-hull design, in which we tested our prototype-selection and reformulation-selection methods. In Section 3, we describe how we formulated two pieces of the optimization problem as inductive learning problems, and the results that we obtained. In Section 4, we describe some of the lessons we learned from this process.

2 Yacht design

Our prototype-selection and reformulation-selection techniques have been developed as part of the “Design Associate,” a system for assisting human experts in the design of complex physical engineering structures [Ellman *et al.*, 1992]. One of the domains in which the Design Associate is currently being tested is the domain of 12-meter racing yachts, which until recently was the class of sailboats raced in America’s Cup competitions. An example of a 12-meter yacht, the Stars and Stripes '87, is shown in Figure 1. ¹

Racing yachts can be designed to meet a variety of objectives, such as course time or cost. In our work we have chosen to focus on a course-time goal, namely minimizing the time it takes for a yacht to traverse a given race course under given wind conditions. A particular course-time goal thus requires the specification of two environment parameters: (1) the race course, represented as a set of (*distance, heading*) pairs; and (2) the wind speed, represented as a scalar number, in knots. Our design system represents a yacht geometry by a set of design parameters, and evaluates course time using a “Velocity-Prediction Program” called “RUVPP,” [Schwabacher *et al.*, 1994] a somewhat simplified version of “AHVPP” from AeroHydro, Inc., which is a marketed product used in yacht design [Letcher, 1991]. Yacht designs are modified by operators that manipulate design parameters. A search space is thus specified by providing the parameters that define an initial prototype, and a set of operators for modifying that prototype.

To find a yacht for a given design goal our system uses one of two optimizers. The first is the Rutgers Hillclimber, which is a steepest-descent hill climber [Press *et al.*, 1986] with some enhancements that allow it do deal with noise. The second is CFSQP, a state-of-the-art implementation of the Sequential Quadratic Programming method [Craig *et al.*, 1994]. Sequential Quadratic Programming is a quasi-Newton method that solves a nonlinear constrained optimization problem by fitting a sequence of quadratic programs² to it, and then solving each of

¹This is the boat that won the 1987 America’s Cup competition, returning the trophy to the United States after an Australian win in 1983 (which represented the only non-US win in more than 100 years [Letcher *et al.*, 1987].)

²A quadratic program consists of a quadratic objective function to be optimized, and a set of linear constraints.

these problems using a quadratic programming method.

In the reformulation-selection experiments described in this paper, we ran CFSQP with *course-time* as the objective function, and with one explicit, nonlinear, “hard” constraint. This constraint specifies that the mass of the yacht, before adding any ballast, must be less than or equal to the mass of the water that it displaces. (In other words, the boat must not sink.)

Although the program we use to compute course time (RUVPP) is a state-of-the-art simulator, it nevertheless suffers from a number of deficiencies that make optimization difficult. For example, it will sometimes return a spurious root of the balance-of-force equations that it solves. It may also exhibit discontinuities, due to numerical round-off error, or due to truncation error in the numerical solver used to solve the balance-of-force equations. These deficiencies can produce “noise” in the evaluation function surface over which the optimization algorithm is moving. The algorithm can therefore easily get stuck at a point that appears to be a local optimum, but is nevertheless not locally optimal in terms of the true physics of the yacht design space. There is also noise in the search space caused by the constraints of the 12-Meter Rule, which is discussed further in Section 3.3.

3 Formulating the inductive learning problems

3.1 Mapping goals into attributes

The two pieces of the optimization problem to which we applied inductive learning are *prototype selection* and *reformulation selection*. In both cases, we used inductive learning to make decisions regarding how the numerical optimization is set up by mapping the *design goal* for a new design session into a selection for one of the optimization setup questions. Before we could apply inductive learning in this way, we had to define a way to map the *goal space* into a *feature space* suitable for use with standard inductive learning methods. In our initial experiments in the yacht domain, we restricted the goals to a set of goals that can be represented using two or three floating-point numbers, representing the wind speed and the race course for which course-time is to be minimized. The next two subsections describe how we applied inductive learning to map the design goal into the selection of a starting prototype for optimization, and into a reformulation of the search space within which we optimize.

3.2 Prototype selection

Many automated design systems begin by retrieving an initial prototype from a library of previous designs, using the given design goal as an index to guide the retrieval process. The retrieved prototype is then modified by a set of design modification operators to tailor the selected design to the given goals. In many cases the quality of competing designs can be assessed using domain-specific evaluation functions, and in such cases the design-modification process is often accomplished by an optimization method such as hill-climbing search.

In the context of such design systems, the choice of an initial prototype can affect both the quality of the final design and the computational cost of obtaining that design, for three reasons. First, prototype selection may impact quality when the prototypes lie in disjoint search spaces. In particular, if the system’s design modification operators cannot convert any prototype into any other prototype, the choice of initial prototype will restrict the set of possible designs that can be obtained by *any* search process. A poor choice of initial prototype may therefore lead

Table 1: Comparison of prototype-selection methods.

Method	Error Rate	Course-Time Increase (sec)
C4.5	30%	24
Most Frequent Class	70%	47
Random Guessing	75%	62
Best Init Eval	70%	64
Closest Goal	70%	78

to a suboptimal final design. Second, prototype selection may impact quality when the design process is guided by a nonlinear evaluation function with unknown global properties. Since there is no known method that is guaranteed to find the global optimum of an arbitrary nonlinear function, most design systems rely on iterative local search methods whose results are sensitive to the initial starting point. Finally, the choice of prototype may have an impact on the time needed to carry out the design modification process - two different starting points may yield the same final design but take very different amounts of time to get there. In design problems where evaluating even just a single design can take tremendous amounts of time, selecting an appropriate initial prototype can be the determining factor in the success or failure of the design process.

In our prototype-selection experiments, we used four prototypes which define four different search spaces. We trained C4.5, a standard inductive-learning algorithm[Quinlan, 1993], using a set of 30 “training goals,” and tested it using a set of 10 “testing goals.” The optimizations for each of these goals were performed using AHVPP and the Rutgers Hillclimber. Table 1 compares the performance of C4.5 with that of several alternative methods. ([Schwabacher *et al.*, 1994] provides more details of the experiments and explanations of the competing methods.) C4.5 outperformed the other methods both on error rate (the percentage of the testing goals for which it chose the right prototype) and course-time increase (the loss in design quality resulting from incorrect choices). We were surprised by how poorly the other methods performed. We did further experiments (also presented in [Schwabacher *et al.*, 1994]) which suggest that C4.5 was able to outperform the other methods because it is better able to deal with noise in the simulator.

3.3 Reformulation Selection

In a simulation-based automated engineering design system that uses numerical optimization, the decision on how to formulate the search space can dramatically affect the performance of the optimizer in two ways. First, using a lower-dimensional formulation of the search space makes optimization faster, since each gradient computation requires fewer runs of the simulator, and the distance in design space from the starting point to the optimum is smaller. Second, different formulations of the search space can result in different degrees of “smoothness” of the search space, which can impact not only the speed of the optimizer, but also the ability of the optimizer to get to the optimum, and therefore the quality of the resulting designs. We present a method of reformulation called “constraint incorporation,” which reduces the dimensionality

Table 2: Effect of using reformulations chosen by learner on optimization performance.

method	quality change	time change
omniscient	+0.085%	-36%
C4.5	+0.080%	-35%
Most Frequent Class	+0.029%	-32%
none	0	0
random	-0.276%	-40%
all	-0.599%	-74%

of the search space and increases its smoothness by incorporating constraints into the search space.

Constraints are often expressed as nonlinear inequalities of the form $f(x) \leq k$. The constraint is said to be *active* if $f(x) = k$. If it is known that the constraint will be active at the optimal design point, and the constraint function f is invertible, then the constraint can be incorporated into the search space. If there are n constraints that can be incorporated in this way, then there are 2^n possible reformulations that can be produced by incorporating different subsets of constraints.

To use inductive learning to form reformulation-selection rules, we take as training data a collection of design goals, each labeled with the set of constraints that are active at the optimal design point. We run the inductive learner once for each constraint, producing a separate set of rules for each constraint.

Yachts entered in the 1987 America’s Cup race had to satisfy what is know as the 12-Meter Rule, which contains several inequality constraints. We created operators that can independently incorporate each of three of these constraints into the search space. We thus defined a space of eight possible reformulations. We then used inductive learning to decide, based on the design goal, which reformulation to use. As training data, we used 99 previous optimizations, which had been done using CFSQP and RUVPP. For each previous optimization, we evaluated each 12-Meter Rule constraint function at the optimum, and determined if the constraint was active (within a tolerance). We ran the inductive learner once for each of the three constraints. Each time, the inductive learner was provided with a set of training data indicating for each goal, whether or not the constraint was active. We then performed optimizations for 25 new randomly generated goals using the reformulations suggested by C4.5 and each of several competing learning methods. Table 2 shows the results.

C4.5 produced a significant speedup in optimization, with no quality loss. In fact, it produced a small quality increase. It outperformed Most Frequent Class (by a small but statistically significant margin), and performed almost as well as the hypothetical omniscient learner, which means it performed almost as well as any learner could possibly do. Incorporating all of the constraints all of the time produced a very large speedup, with a small quality loss. Further details of these experiments can be found in [Schwabacher *et al.*, 1995].

Table 3: Cross-validated error rates for selecting whether to incorporate each constraint.

Constraint	C4.5 w/ pruning	C4.5 w/o pruning	C4.5rules	MFC	Random
Beam	11.1%	11.1%	11.1%	33.3%	66.7%
Displacement	15.1%	15.1%	15.1%	53.5%	66.7%
Winglet	7.0%	10.0%	10.0%	13.1%	66.7%

4 Lessons learned

During this work we have learned a number of lessons about the practical application of inductive learning to real-world problems. The next three subsections describe what we have learned about the importance of choosing the right evaluation criteria, the importance of getting good data, and the usefulness of visualization.

4.1 Evaluation criteria

We feel it is important to note the importance of choosing the correct evaluation criteria when applying inductive learning in a real-world domain such as complex engineering design, rather than on standard test problems such as those found in the UCI repository. Researchers in inductive learning have traditionally evaluated learning performance on the basis of *error rate*, the percentage of test examples incorrectly classified. We argue that it is more important to evaluate the learners on the basis of the impact that they have in the domain, using whatever evaluation criteria are normally used in the particular domain. For example, a learning system used in a business could be evaluated on the basis of its impact on profits. Since our inductive learning systems were used within an engineering design optimization system, we evaluated performance using the two measures of merit normally applied to design optimization systems: the quality of the resulting design, and the amount of CPU time needed to produce the design.

Further, we believe that using error rate instead of the real-world measure of merit can be misleading. For example, at first we used error rate to evaluate the inductive learners for the reformulation-selection problem. We obtained the results shown in Table 3. On the basis of this table, it looked like C4.5 performed substantially better than Most Frequent Class (MFC). When we later compared the learners on actual optimization performance, we obtained the results shown in Table 2, showing that C4.5 had only a small advantage over MFC.

4.2 The importance of getting good data

When the people setting up the learning system also have control over the generation of training data, as we did in our experiments, it is sometimes possible to improve learning performance by improving the training data. In both sets of experiments described in this paper, we at first obtained poor learning performance. We determined that the training data was “noisy” – that is, many of the “optimal” prototypes were not really optimal for the goals for which they were labeled as being optimal. We determined that the reason for this noise was that the optimizer often got “stuck” before reaching the true optimum. We were able to improve the quality of the training data by making several improvements to the optimizer and to the simulator that

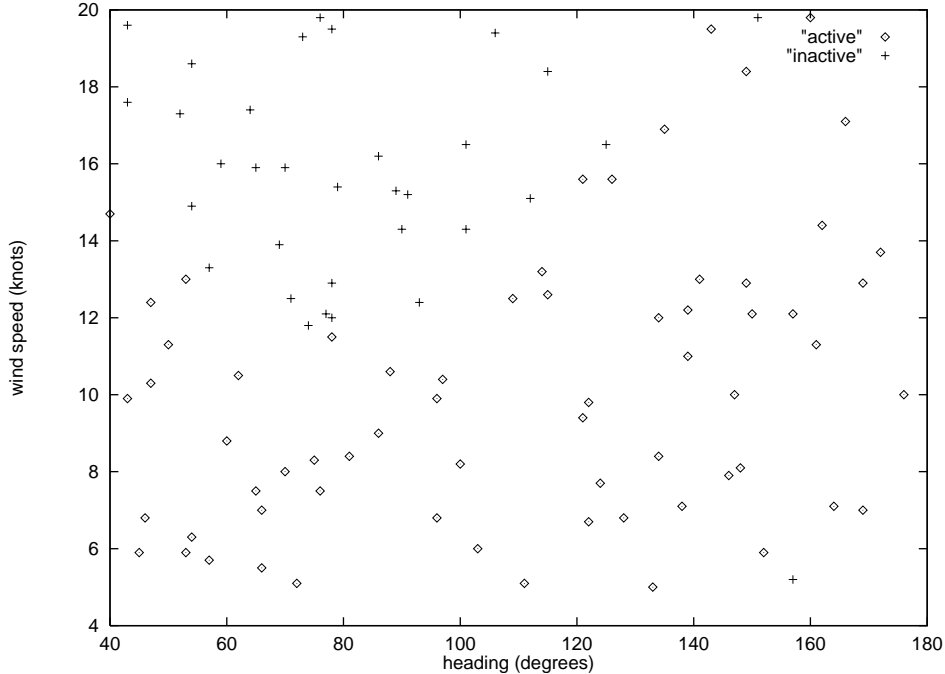


Figure 2: Activity of the beam constraint over the goal space.

was used within the optimizer. At first we were using a yacht simulator called AHVPP; we replaced this with a simulator called RUVPP for which the multidimensional surface defined by the search space and the simulator is smoother. At first we used a simple hill climber as our optimizer; we replaced the hill climber with CFSQP, and then later we further improved CFSQP’s performance by adjusting its parameters, and by making the constraint that the yacht not sink explicit.

4.3 The value of visualization

We have found that visualization can be very helpful in determining how noisy the training data is. For example, Figure 2 shows the training data from which we learned the activity of the beam constraint for reformulation selection. In this figure, there is one point labeled as “inactive” that is completely surrounded by points that are labeled as “active.” We believe that the activity of this constraint should be monotonic in the two goal parameters; we therefore believe that this point is noise. C4.5 has the ability to deal with limited amounts of noise, so this one noisy point did not cause a problem. However, in our earlier experiments we discovered through visualization that our training data was much noisier.

Visualization of the training data can also be used to choose an appropriate inductive learning algorithm. For example, C4.5, like most decision-tree learners, uses linear, axis-parallel cuts in its decision trees. However, Figure 2 shows that the space is clearly divided into two regions (except for the one point which we believe is noise). The border between these regions does not appear to be axis parallel, and appears to be nonlinear. This suggests that better performance might be achieved using an “oblique” decision tree learner, such as OC1 [Murthy *et al.*, 1993], or by attempting to learn nonlinear region boundaries.

5 Related work

Cerbone [Cerbone, 1992] has reported work which applied machine-learning techniques to a problem similar to our prototype-selection problem. His design space, in the domain of truss design, has an exponential number of disconnected search spaces. He uses inductive learning techniques to learn rules for selecting a subset of these search spaces for further exploration. Several investigators [Orelup *et al.*, 1988, Tong, 1988, Powell, 1990, Hoeltzel and Chieng, 1987] have developed alternative artificial-intelligence techniques for controlling iterative parameter-design optimization. [Choy and Agogino, 1986] describe a system that automates [Papalambros and Wilde, 1988]’s method of using monotonicity analysis to detect constraint activity. As far as we know, nobody has applied machine learning to predicting constraint activity, or to selection of a search space reformulation for design optimization.

6 Future work

This paper has described on-going work, and there are thus a number of directions for future work. For one, we plan to look for other inductive learning problems within the design optimization problem. For another, we plan to perform experiments to see how optimization performance is affected when two or more learning techniques, such as prototype selection and reformulation selection, are combined.

The results presented here apply to a constrained class of yacht-design goals, those comprised of either one or two legs. One question is how our approach can be applied to courses comprised of varying numbers of legs. Learning from race courses with variable numbers of legs would raise an interesting machine-learning question, since describing a multi-leg race course requires a variable number of attributes, and thus traditional learners such as C4.5 do not directly apply.

We also plan to apply our methods to more-difficult problems, such as those that involve a less-smooth search space, a higher-dimensional goal space, or a less reliable optimizer. Such problems may arise when we test our methods in other domains. In particular, we plan to test them in the domain of aircraft design.

As would be expected, even though our yacht-domain reformulation-selection results with C4.5 were nearly optimal for 100 examples, results degrade when given less training data. One would expect similar degradation with any inductive learning problem. Although it would be interesting to see if other learning methods would have better small-dataset performance, for any learner we would expect performance to be inferior for small enough datasets. One approach for improving results in such small-dataset cases — as well as in other cases where off-the-shelf learners such as C4.5 may not perform well even if given larger datasets — is to integrate background knowledge into the learning process. One form of background knowledge that is often available, such as in the yacht-design domain, is *modality constraints*. This is knowledge that expresses the modality of the learned class with respect to the attributes. For example, we believe that optimal *beam* is monotonically increasing in wind speed, and monotonically decreasing in heading. We also know that the activity of any constraint of the form $f(x_1, x_2, \dots, x_n) \leq k$ must be monotonic in k , so, for example, the activity of a cost constraint must be monotonic in the cost threshold. One open question is how such knowledge could be integrated into learning. One approach would be to use such modality constraints to remove from the training data points that violate the constraints (on the assumption that these points are noise). A second approach

is to modify the tree induction algorithm so that it will never construct a tree that violates the constraints.

Finally, even after our learning approach is applied, every additional future optimization can serve as an additional training point for the learning. Thus learning methods that can work in an incremental fashion might also prove useful for this task. In addition, it may prove useful to develop methods that select suitable data prior to learning. For example, when there are not enough existing optimizations to achieve adequate learning results, additional optimizations can be performed to generate further training data. Rather than performing these new optimizations for random goals or for a set of goals that span the goal space, one could allow the learner to choose the goals to be used in the new training data. Background knowledge — such as modality constraints — could prove particularly useful in selecting such goals.

Acknowledgments

This research has benefited from numerous discussions with members of the Rutgers HPCD project. In particular, we would like to thank Gerard Richter, Andrew Gelsey, and John Keane. This research is part of the Rutgers-based HPCD (Hypercomputing and Design) project supported by the Advanced Research Projects Agency of the Department of Defense through contract ARPA-DABT 63-93-C-0064.

References

- [Cerbone, 1992] G. Cerbone. Machine learning in engineering: Techniques to speed up numerical optimization. Technical Report 92-30-09, Oregon State University Department of Computer Science, 1992. Ph.D. Thesis.
- [Choy and Agogino, 1986] J. Choy and A. Agogino. Symon: Automated symbolic monotonicity analysis system for qualitative design optimization. In *Proceedings ASME International Computers in Engineering Conference*, 1986.
- [Craig *et al.*, 1994] L. Craig, J. Zhou, and A. Tits. User's guide for CFSQP version 2.1: A C code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints. Technical Report TR-94-16r1, Institute for Systems Research, University of Maryland, November 1994.
- [Ellman *et al.*, 1992] T. Ellman, J. Keane, and M. Schwabacher. The Rutgers CAP Project Design Associate. Technical Report CAP-TR-7, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1992.
- [Hoeltzel and Chieng, 1987] D. Hoeltzel and W. Chieng. Statistical machine learning for the cognitive selection of nonlinear programming algorithms in engineering design optimization. In *Advances in Design Automation*, Boston, MA, 1987.
- [HPCD, 1995] Hypercomputing and design. World-Wide Web Page <http://www.cs.rutgers.edu/hpcd/hpcd.html>, 1995.

- [Letcher *et al.*, 1987] J. Letcher, J. Marshall, J. Oliver, and N. Salvesen. Stars and Stripes. *Scientific American*, 257(2), August 1987.
- [Letcher, 1991] J. Letcher. *The Aero/Hydro VPP Manual*. Aero/Hydro, Inc., Southwest Harbor, ME, 1991.
- [Murthy *et al.*, 1993] S. Murthy, S. Kasif, S. Salzberg, and R. Beigel. OC1: Randomized induction of oblique decision trees. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, D.C., 1993.
- [Orelup *et al.*, 1988] M. F. Orelup, J. R. Dixon, P. R. Cohen, and M. K. Simmons. Dominic II: Meta-level control in iterative redesign. In *Proceedings of the National Conference on Artificial Intelligence*, pages 25–30, St. Paul, MN, 1988.
- [Papalambros and Wilde, 1988] P. Papalambros and J. Wilde. *Principles of Optimal Design*. Cambridge University Press, New York, NY, 1988.
- [Powell, 1990] D. Powell. Inter-GEN: A hybrid approach to engineering design optimization. Technical report, Rensselaer Polytechnic Institute Department of Computer Science, December 1990. Ph.D. Thesis.
- [Press *et al.*, 1986] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes*. Cambridge University Press, New York, NY, 1986.
- [Quinlan, 1993] John Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.
- [Schwabacher *et al.*, 1994] M. Schwabacher, H. Hirsh, and T. Ellman. Learning prototype-selection rules for case-based iterative design. In *Proceedings of the Tenth IEEE Conference on Artificial Intelligence for Applications*, San Antonio, Texas, 1994.
- [Schwabacher *et al.*, 1995] M. Schwabacher, T. Ellman, and H. Hirsh. Learning when reformulation is appropriate for iterative design. To appear at IJCAI-95 Workshop of Machine Learning in Engineering, 1995.
- [Tong, 1988] S. S. Tong. Coupling symbolic manipulation and numerical simulation for complex engineering designs. In *International Association of Mathematics and Computers in Simulation Conference on Expert Systems for Numerical Computing*, Purdue University, 1988.