Learning To Select Prototypes and Reformulations for Design

Mark Schwabacher Thomas Ellman Haym Hirsh
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903
908-445-{4838, 4184, 4176}
{schwabac, ellman, hirsh}@cs.rutgers.edu

1 Introduction

We are working on using machine learning to make engineering design optimization faster and more reliable. We envision a system that learns from previous design sessions knowledge that enables it to assist the engineer in setting up and carrying out a new design optimization. We have performed initial experiments for two aspects of setting up an optimization: selecting a prototype to serve as a starting point for the optimization, and selecting a reformulation of the search space. Both choices can dramatically affect both the speed and the reliability of design optimization. Our technique applies a standard inductive-learning algorithm, such as C4.5, to a set of training data describing which particular prototype or reformulation would have been the best choice for each goal encountered in a previous design session. We have tested our technique in the domains of racing-yacht-hull design and supersonic transport aircraft design, comparing our inductively learned selection rules to several competing prototype- and reformulation-selection methods. Our experimental results show that using the prototypes and reformulations selected by these rules improves both the speed and reliability of design optimization, and that the inductive method of selecting prototypes and reformulations outperforms competing methods.

Our selection techniques have been developed as part of the "Design Associate" [Ellman *et al.* 1992, Ellman *et al.* 1993], a system for assisting human experts in the design of complex physical engineering structures. The Design Associate has been tested in the domain of

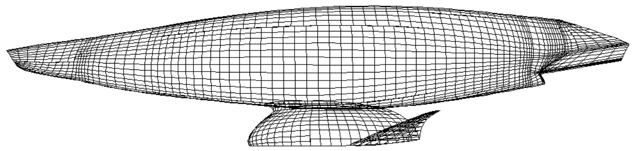

Figure 1: The Stars and Stripes '87.

12-meter racing yachts, which until recently was the class of yachts raced in America's Cup competitions. An example of a 12-meter yacht, the Stars and Stripes '87, is shown in figure 1.

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 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 [Quinlan 1993] using a set of 30 "training goals," and tested it using a set of 10 "testing goals." Table 1 compares the performance of C4.5 with that of several alternative methods, some of which are based on simple heuristics. ([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 Prototype Synthesis

A variation on prototype selection is *prototype synthesis.* Instead of selecting a prototype from a database of prototypes, prototype synthesis synthesizes a new prototype by mapping the design goal directly into the design parameters that define a prototype. We performed some experiments to test prototype synthesis in the domain of supersonic transport aircraft

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

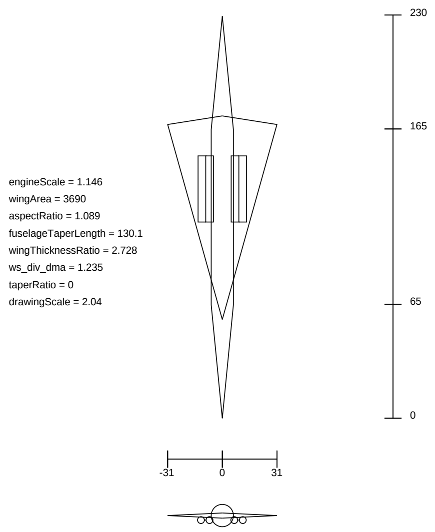

Figure 2: Supersonic transport aircraft designed by our system (dimensions in feet)

Table 2: Comparison of prototype-synthesis methods.

Method	Success	Cost
CART	13/16	7394
mean	14/16	11963
random	8/16	17915

design. Figure 2 shows an example of an airplane designed by our system [Gelsey *et al.* 1996].

In order to map the design goal into the numerical design parameters that define a prototype, we need to use *continuous-class induction* (which is also known as regression). We used CART (Classification And Regression Trees), which builds a "regression tree" that has a numerical constant at each leaf [Breiman 1984]. We trained CART on 100 randomly generated training goals, and used 25 randomly generated testing goals. Table 2 compares using the prototypes synthesized by CART with using randomly generated prototypes, or always using the prototype which is the mean of all the optimized prototypes in the training data. Of the 25 randomly generated test goals, 16 were achievable. The "success" column shows the number of optimizations that came within 1% of the point that we believe to be the global optimum. Some of the failures occurred because the learning method produced an "unevaluable" prototype that could not be simulated, and therefore could not be optimized. Other failures occurred because the optimizer, when started from the syntesized point, failed to get within 1% of the apparent global optimum. The "cost" column shows the total number of simulations used in the 16 optimizations. Using the mean prototype instead of a random prototype resulted in much greater success, at 33% lower cost. Using CART produced a success rate about the same as using the mean prototype, with an additional 38% cost reduction.

4 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 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

Table 3: 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%

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. 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 3 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.* 1996].

References

[Breiman 1984] L. Breiman. *Classification and regression trees.* Wadsworth International Group, Belmont, Calif., 1984.

[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. ftp://ftp.cs.rutgers.edu/pub/technical-reports/cap-tr-7.ps.Z.

[Ellman *et al.* 1993] T. Ellman, J. Keane, and M. Schwabacher. Intelligent model selection for hillclimbing search in computer-aided design. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, D.C., 1993.

[Gelsey *et al.* 1996] Andrew Gelsey, Mark Schwabacher, and Don Smith. Using modeling knowledge to guide design space search. In *Artificial Intelligence in Design Conference.* Kluwer Academic Publishers, June 1996. To appear.

[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.* 1996] M. Schwabacher, T. Ellman, H. Hirsh, and G. Richter. Learning to choose a reformulation for numerical optimization of engineering designs. In *Artificial Intelligence in Design Conference*, June 1996. To appear.