Building Models to Support Synthesis in Early Stage Product Design

R. Bharat Rao
Learning Systems Department
Siemens Corporate Research, Inc.

Stephen C-Y. Lu
Knowledge-based Engg. Systems Res. Lab
University of Illinois at Urbana-Champaign

Abstract

Current computer-aided engineering paradigms for supporting synthesis activities in engineering design require the designer to use analysis simulators iteratively in an optimization loop. While optimization is necessary to achieve a good final design, it has a number of disadvantages during the *early* stages of design. In the *inverse engineering* methodology, machine learning techniques are used to learn a multidirectional model that provides vastly improved synthesis (and analysis) support to the designer. This methodology is demonstrated on the early design of a diesel engine combustion chamber for a truck.

Introduction

A design engineer's primary task is to develop designs that can achieve specified performances. For example, an engine designer may be required to design a "combustion chamber that delivers at least 600hp while minimizing the fuel consumption." The horsepower and fuel consumption are the performance parameters. In a parameterized domain, the designer sets the values of the decision/design parameters (e.g., engine rpm) so as to meet the performance. Under current computer-aided engineering (CAE) paradigms, design support is typically provided by computer simulators. These simulators are computerized *analysis* models that analyze a design and map decisions to performances. However, engineering design is largely a *synthesis* task that requires mapping the performance space, P, to the decision space, D. In the absence of models that provide synthesis support, the designer must use the simulator, F, in an iterative generate and test paradigm, namely, in an optimization loop. The designer begins with an initial design (a point in D), evaluates the design with F, and moves to a new design, based on the difference between the actual and required performances. A common way of moving within D is by response surface fitting, where the designer exercises F repeatedly in the neighborhood of the current design, fits a surface to the performances, and moves based on this surface.

While the above methodology is essential for the final stages of design, it has serious drawbacks during *early* design. First, a poor starting design can result in a large number of optimization steps, which can be very time-consuming. The designer would prefer to rapidly develop a good initial design to use as the starting point in the optimization. Second, the simulator is a point to point simulator. This means that the designer must assign values to every decision variable at the outset of the design. Ideally, the designer would specify or restrict only the variables in which he was interested, leaving the others to be automatically specified as the design progresses. Third, every new performance objective must be set up as a separate design problem. For example, instead of designing an engine that generates 600hp, perhaps there exists an engine that delivers 590hp but with markedly improved fuel consumption, or another that delivers 640hp with slightly less efficient fuel consumption. If the designer had a synthesis model, he could apply and retract conditions on some of the performance variables and quickly determine their effects on the other variables. Similarly he could constrain the decision variables to reflect cost concerns, inventory stocks, or simply as part of a "What-if" analysis. The ability to treat decision and performance variables more or less identically would prove extraordinarily valuable during early design. The synthesis model, once learned, could be used repeatedly for different designs.

The *inverse engineering* methodology [Rao, 1993] provides solutions to all the above problems by building an accurate *multidirectional* model of the problem domain. The designer uses the model directly to prototype a design quickly by successively refining the problem space, $D \cup P$ (as opposed to the traditional CAE paradigm where the designer works only in D). A single invocation of F at the end of the process is sufficient to check the design. The foundation of inverse engineering is KEDS, the Knowledge-based Equation Discovery System [Rao and Lu, 1993]. KEDS ability to learn accurate models in representations that can be converted into constraints (i.e., as piece-wise linear models) makes inverse engineering a viable proposi-

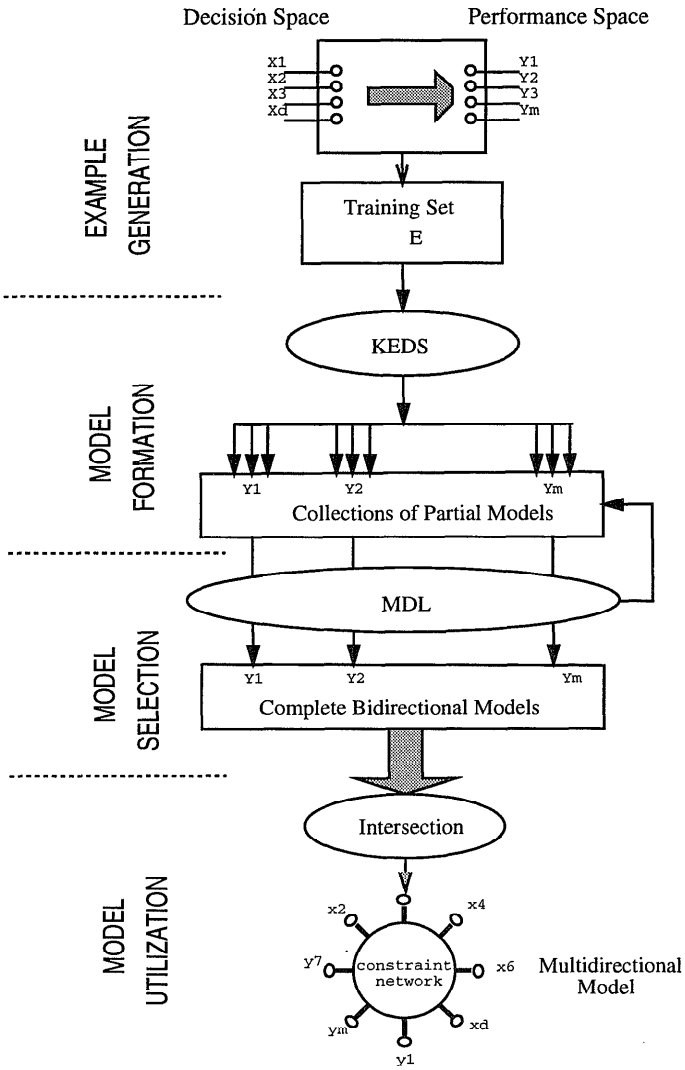


Figure 1: The Inverse Engineering Methodology

tion. This rest of this paper describes this methodology, and demonstrates (through an example) how these techniques provides improved support for early design.

The Inverse Engineering Methodology

The essential problem with current CAE paradigms for design is the lack of synthesis support. The barrier between analysis and synthesis activities is especially unbearable in a concurrent engineering framework, where speed and timely execution of tasks is paramount. As directly learning synthesis models is a very hard task [Rao, 1993], the inverse engineering approach is to *learn analysis models in representations that provide synthesis support.* For example, if the analysis model for $y_j \in P$ can be accurately represented as a linear function of some $x_i \in D$ (i.e., $y_j = \sum(a_i x_i) + b$), it can then be converted into a constraint that provides

Table 1: Decision and Performance variables in DESP

Variable		Min	Max
STBR	Stroke-to-Bore Ratio	0.8	1.2
TIM	Injection Timing (deg)	320.0	335.0
FMIN	Fuel Mass Injected (gm/cyc)	0.1	0.4
CR	Compression Ratio	13.0	17.0
ERPM	Engine Speed (rpm)	1000.	2400.
DVOL	Displacement Volume (l)	0.005	0.015
BSFC	Brake Specific Fuel Consumption		
ENBHP	Engine Brake Horsepower (HP)		

both analysis and synthesis support. The 4 phases of inverse engineering (see Figure 1) are described below.

Example Generation Phase The Diesel Engine Simulation Program, DESP [Assanis and Heywood, 1986], provides the data for KEDS. DESP solves mass and heat balance equations from Thermodynamics and uses finite difference techniques to provide data that is representative of real-world engines. The 6 decision and 2 performance (real-valued) variables for a 6-cylinder, diesel combustion engine are shown in Table 1. The decision variables are randomly varied to generate 145 events. This results in two data sets (one for each performance variable, BSFC and ENBHP in Table 1), such that each event is a two-tuple of a decision vector $X \in D$ and a corresponding performance variable, $y_j \in P$. These data sets are input to KEDS to learn multidirectional models.

Model Formation Phase KEDS is a model-driven empirical discovery system that learns models in forms restricted to $\mathcal{F}$, a user-defined class of parameterized model families (both linear and non-linear $\mathcal{F}$ are permitted). For the purposes of inverse engineering, $\mathcal{F}$ is restricted to the class of linear polynomials, $y = \sum(a_i x_i) + b$. However, it is unlikely that a simple linear representation will be sufficient to accurately model most real-world domains. KEDS can simultaneously be viewed as a conceptual clustering system, which partitions the data based upon the mathematical relationships that it discovers between the variables. Each call to KEDS results in a single *partial model* (R, f), that consists of a region (hyperrectangle), R $\subset D$, associated with an equation, $f \in \mathcal{F}$, that predicts y for all $X \in$ R . The KEDS algorithm (described in [Rao and Lu, 1993]) involves recursing through equation discovery (fitting) and partitioning (splitting) and combines aspects of fit-and-split [Langley *et al.*, 1987] and split-and-fit [Friedman, 1991; Quinlan, 1986] modeling systems as KEDS refines both the region and the equation. A sample partial model is shown below.

```
[321.0< TINJ] [.2482< FMIN <.3941] [13.16< CR <16.8]
[1074< RPM][.0103< VOL] [.813< STBR] ::>
::> BSFC = 1.3 FMIN -.003 TIM -.008 CR +1.5E-5 RPM -27. VOL +1.
```

Model Selection Phase KEDS is invoked repeatedly to generate a collection of overlapping partial models. KEDS-MDL [Rao and Lu, 1992] is a resource-bounded incremental algorithm that uses the minimum description length [Rissanen, 1986] principle to select partial models to build a piece-wise complete model. This is a collection of disjoint partial models that describes the entire decision space.

Model Utilization Phase Each partial model (region-equation pair) is equivalent to a linear constraint that maps a region in D to an interval in y_j. KEDS-MDL learns piece-wise linear models for ENBHP and BSFC. The constraints for ENBHP are intersected with the constraints for BSFC to produce a set of *intersections*. An intersection maps a region in D to a region in P, and also supports reasoning from P to D. No two intersections overlap within the decision space, but the regions in performance space do typically overlap (as several different designs can achieve the same performance). Unlike the traditional CAE paradigm, the designer works in the problem space, $D \cup P$. The designer can *refine* any intersection by refining a variable, i.e., by shrinking the interval associated with that variable. Refining a decision variable leads to *forward* propagation along a constraint and the new intervals for the other variables can be determined in a straightforward fashion. Refining a performance variable requires *inverse* propagation along constraints. One possibility is to solve the intersection to find the new feasible region in D (for example, by using Simplex). Instead, this is done by computing the projection of the feasible region onto the decision variables (i.e., the enclosing hyperrectangle) in a single step computation [Rao, 1993]. Inverse propagations can lead to forward propagations, and vice versa.

The DESP domain has 15-30 intersections, depending upon the model formation parameters used in KEDS. While it would be a great strain, it is remotely possible that a designer would be able to work individually with each intersection. However, other domains can give rise to many more intersections (a process planning application for a turning machine has 1000+ intersections). Instead of working with each individual intersection, the designer refines a single composite region that consists of the union of the intervals for all intersections. A truth maintenance system keeps track of the effects of refinements on each intersection, and the designer only sees the composite interval for each variable. This occasionally leads to *gaps* in the problem space, when two or more disjoint decision regions have similar performance.

A number of CAD/CAM [Finger and Dixon, 1989; Vanderplaats, 1984] and AI [Dixon, 1986] techniques have been developed to support engineering design. A complementary approach to inverse engineering for breaking the analysis-synthesis barrier for early design is to develop representations and theories for multi-directional models [Herman, 1989] that could replace existing analysis simulators. However, this fails to take advantage of past research efforts in developing computer simulators. Another approach is to speed up the iterative optimization process by replacing slow computer simulators with faster models [Yerramareddy and Lu, 1993]. For a detailed review of related machine learning and design research, see [Rao, 1993].

Product Design Demonstration

The inverse engineering interface is shown in Figure 2. There are 6 function windows. The *Control Panel* is used primarily to initialize the domain by loading models created offline by KEDS-MDL , and to simulate the final design. The original intervals of the multidirectional model are displayed in the *Original Model* Window. The *Messages Window* displays detailed domain information. The *Lisp Listener* is for development.

The *Decision Panel* is the window in which the designer does virtually all his work. Clicking on the "Refine" button brings up a pop-up menu of the variable names. Clicking on a variable (e.g., ENBHP) brings up an Emacs window titled "Ranges for parameter: ENBHP" that displays the current ranges for that variable. After refining the values with Emacs commands, hitting the return key causes the refinement to be quickly propagated through all of the intersections creating a new *world*. In Figure 2 the designer has just refined the ENBHP variable in the Decision Panel to demand that the engine deliver at least 600hp. The *Worlds Display Panel* (WDP) shows a *world view* reflecting the state of the world after the ENBHP refinement. The first three columns in the world view show the names of the variables and the current intervals. The last two columns, "Dmin" and "Dmax," in the world view represent the change (i.e., the delta) in the intervals relative to the previous world. Figure 2 shows that after the ENBHP refinement both boundaries of the compression ratio were moved inwards, the lower bound of the engine speed was increased, and there was no influence on the fuel consumption (see Table 1 for acronyms). Successive world views occlude previous views in the WDP. The Messages Window indicates that the designer cannot refine STBRAT to fall completely within the gap, $]0.969, 0.988[$. Clicking on the "Retract" button in the Decision Panel retracts the last refinement, and uncovers the previous world view. This interface was built on the interfaces for the HIDER [Yerramareddy and Lu, 1993] and IDEEA [Herman, 1989] systems.

Forming an Early Design

The designer's task is to design a combustion chamber for a 6-cylinder diesel engine for a truck. The engine should deliver at least 600hp, though this could be slightly relaxed based on the designer's judgment. In general, good designs have high ENBHP with a low BSFC. There are other cost concerns that may come

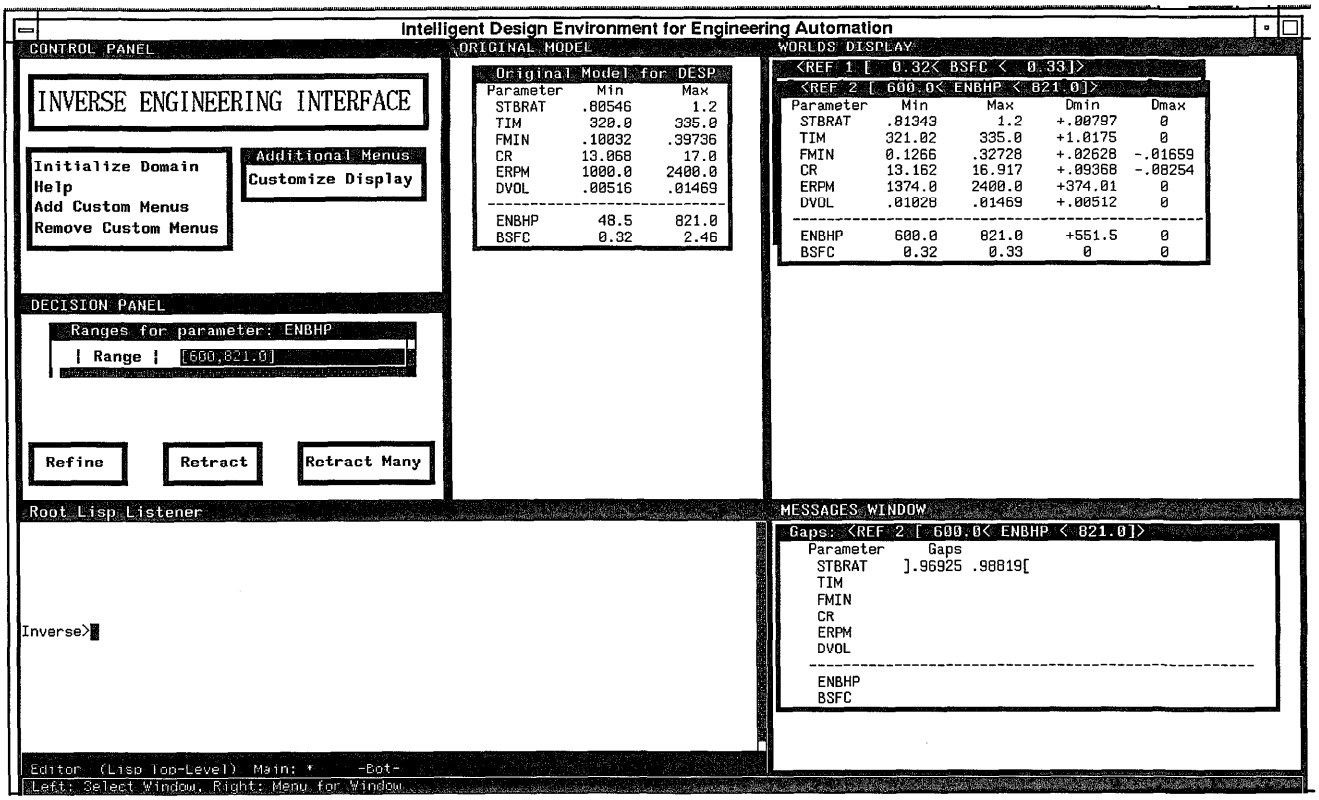


Figure 2: User Interface for Inverse Engineering Environment

into play as the designer applies his background knowledge. In this section we follow a designer step by step, as he uses the inverse engineering interface to come up with a complete early design. Each refinement step is indexed by a number indicating the level of refinement.

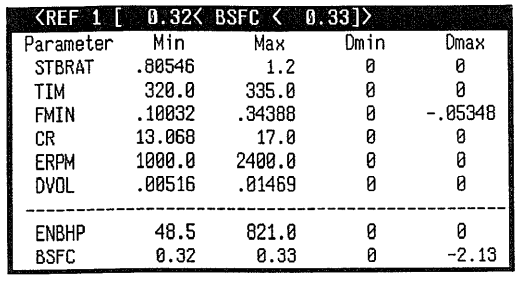
- (1) *Refine BSFC to a max of 0.33.* The designer exploits the synthesis support to set fuel consumption to a low value. The screen bitmap of the corresponding world view is shown in Figure 3(a).
- (2) *Refine ENBHP to a min of 600.* Figure 3(b) shows that this refinement influences many other variables (see "Dmax" and "Dmin" fields).

While the designer does not have to begin all designs by restricting P, the importance of being able to directly constrain the performance parameters is tremendous. From this point onwards the designer can make any changes in D, and is assured that the propagation mechanisms will constrain the remaining variables to meet the performance specifications.

- (3) *Refine ERPM to a max of 1400.* Engines that run at lower speeds have higher manufacturing tolerances and thus lower costs associated with them. Unfortunately, restricting the speed to a very low value adversely affects other decision variables as shown in Figure 3(c). In order to deliver 600hp with BSFC< .33, the CR must be a minimum of 16.3. Higher CR's requires thicker engine cylinder walls, increasing the cost of the engine.
- *Retract Refinement 3.* The system returns to the state shown in Figure 3(b).
- (3) *Refine CR to a value of 15.0.* See Figure 3(d).

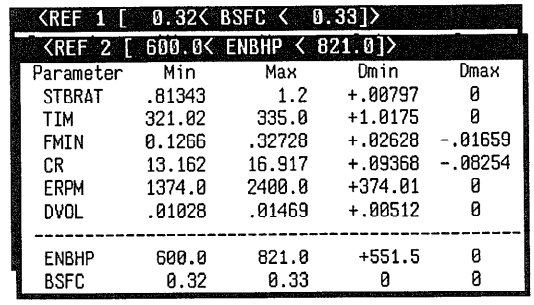
While the designer can restrict the CR to a range, the ability to set a variable to an exact value is very useful. Typically, the values of the variables are optimized by exploring the terrain in the problem space. Even though decision variables, such as STBRAT and CR, are continuous-valued, the engine is most easily manufactured if these variables are set to values that can be easily machined. These settings could also cut down on manufacturing costs and time by using existing inventory and machine setups, rather than retooling factories for every new design.

- (4) *Refine DVOL* $= 0.0145$ (cylinder displacement volume is 14.5 liters).
- (5) *Refine STBRAT* $= 1.0$. The designer notices a gap in the range $]0.969, 0.988[$. He chooses 1.0 as an easily machined value of STBRAT.
- (6) *Refine TIM* $= 334.5$. See Figure 3(e).
- (7) *Refine ERPM* $= 2060$. The designer conservatively picks central values that meet manufacturing requirements for the last two unspecified variables.



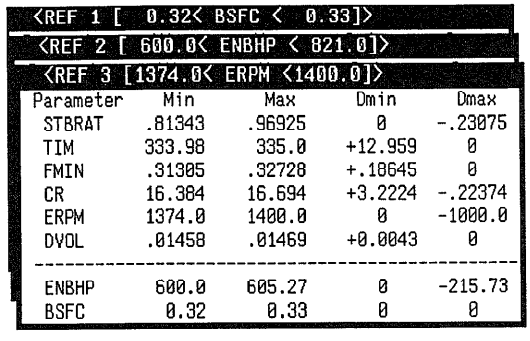
Parameter	Min	Max	Dmin	Dmax
STBRAT	.80546	1.2	0	0
TIM	320.0	335.0	0	0
FMIN	.10032	.34388	0	-.05348
CR	13.068	17.0	0	0
ERPM	1000.0	2400.0	0	0
DVOL	.00516	.01469	0	0
ENBHP	48.5	821.0	0	0
BSFC	0.32	0.33	0	-2.13

(a) Refining BSFC



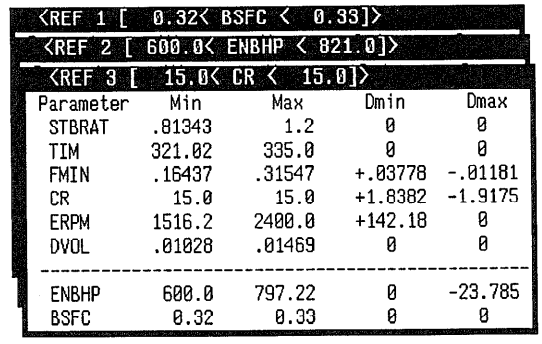
Parameter	Min	Max	Dmin	Dmax
STBRAT	.81343	1.2	+.00797	0
TIM	321.02	335.0	+1.0175	0
FMIN	0.1266	.32728	+.02628	-.01659
CR	13.162	16.917	+.09368	-.08254
ERPM	1374.0	2400.0	+374.01	0
DVOL	.01028	.01469	+.00512	0
ENBHP	600.0	821.0	+551.5	0
BSFC	0.32	0.33	0	0

(b) Refining ENBHP



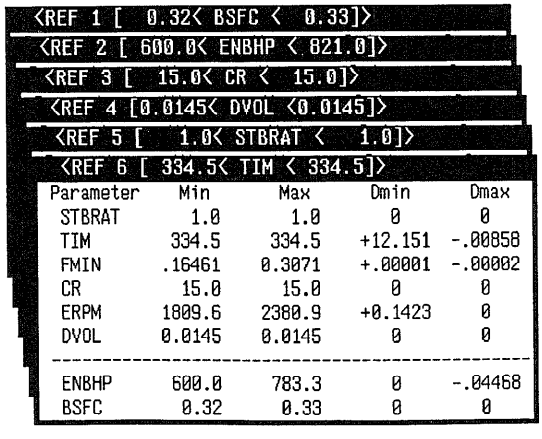
Parameter	Min	Max	Dmin	Dmax
STBRAT	.81343	.96925	0	-.23075
TIM	333.98	335.0	+12.959	0	
FMIN	.31305	.32728	+.18645	0
CR	16.384	16.694	+3.2224	-.22374
ERPM	1374.0	1400.0	0	-1000.0
DVOL	.01458	.01469	+0.0043	0
ENBHP	600.0	605.27	0	-215.73
BSFC	0.32	0.33	0	0

(c) Refining ERPM



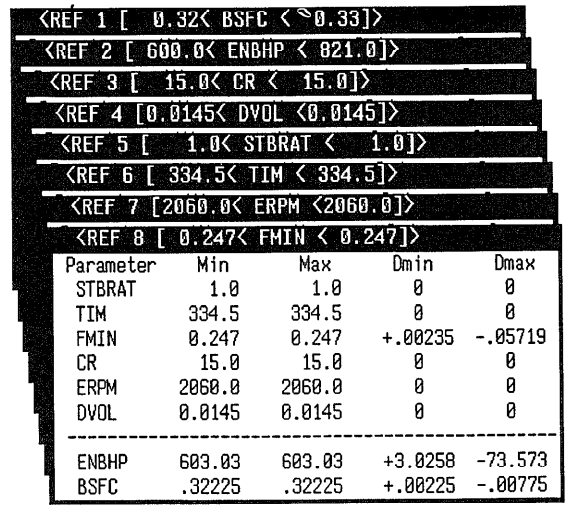
Parameter	Min	Max	Dmin	Dmax
STBRAT	.81343	1.2	0	0
TIM	321.02	335.0	0	0
FMIN	.16437	.31547	+.03778	-.01181
CR	15.0	15.0	+1.8382	-1.9175
ERPM	1516.2	2400.0	+142.18	0
DVOL	.01028	.01469	0	0
ENBHP	600.0	797.22	0	-23.785
BSFC	0.32	0.33	0	0

(d) Refining CR



Parameter	Min	Max	Dmin	Dmax
STBRAT	1.0	1.0	0	0
TIM	334.5	334.5	+12.151	-.00858
FMIN	.16461	0.3071	+.00001	-.00002
CR	15.0	15.0	0	0
ERPM	1809.6	2380.9	+0.1423	0
DVOL	0.0145	0.0145	0	0
ENBHP	600.0	783.3	0	-.04468
BSFC	0.32	0.33	0	0

(e) Refining TIM



Parameter	Min	Max	Dmin	Dmax
STBRAT	1.0	1.0	0	0
TIM	334.5	334.5	0	0
FMIN	0.247	0.247	+.00235	-.05719
CR	15.0	15.0	0	0
ERPM	2060.0	2060.0	0	0
DVOL	0.0145	0.0145	0	0
ENBHP	603.03	603.03	+3.0258	-73.573
BSFC	.32225	.32225	+.00225	-.00775

(f) Refining FMIN

Figure 3: Engine Design Example: World views from the Inverse Interface

- (8) *Refine FMIN = 0.247.* The initial design (henceforth, D1) is complete. The world-view in Figure 3(f) indicates that according to the model, D1 delivers 603hp at a fuel consumption of 32.3%.

The designer uses the "Simulate Design" option in the Control Panel to run DESP on the design. The performance of D1 is computed to be 612.55 hp at 32.9% fuel consumption, which meets the performance constraints of Refinements 1 and 2 above. Note that any optimization of D1 with DESP will almost certainly result in a superior design in the neighborhood of D1.

Exploring alternate designs

The designer chooses the "Retract Many" option to retract Refinement 1 limiting the BSFC to 0.33. The designer is willing to loosen up slightly on the BSFC requirement if improvements appear elsewhere, for instance in the form of increased horsepower. The designer now sets the minimum ENBHP to 650hp and proceeds in a similar fashion to that described in the previous section. The resulting engine, parameterized by D2=(1.0 334.5 0.247 13.5 2380.0 0.0145), delivers 681hp at 34.2% consumption. The designer had earlier (while designing D1) unsuccessfully tried to lower the engine speed so as to reduce manufacturing costs (see Figure 3(c)). In a further attempt to achieve this, the designer relaxes the ENBHP constraint (Refinement 2 above) while imposing low BSFC and RPM constraints. The resulting design, D3=(1.0 334.5 0.247 17.0 1800.0 0.0145), has 32.09% consumption but delivers only 555hp. Another design, D4=(0.85 334.5 0.247 15.0 2000.0 0.0145), is created when the designer constrains STBRAT=0.85, CR$\leq$15, and BSFC$\leq$0.33. This design delivers 592hp at 33.0% consumption.

Of the 4 designs, D1–4, D3 is discarded because the horsepower delivered by that engine is too low (555hp), and D4 is eliminated because its performance is worse than D1 for both horsepower (590hp versus 603hp) and fuel consumption (33.0% versus 32.9%). The designer can make a choice between D1 and D2 at this point; for example, he can eliminate D2 if he deems that the extra 69hp (=681-612) is not worth the 1.3% drop in fuel efficiency. Alternatively, he could choose to optimize both D1 and D2 using the traditional CAE paradigm and defer the decision. He could then decide to manufacture two lines of trucks or search for more designs with the user interface. Whichever option the designer chooses, his choice is likely to be more informed, than would have been the case had he worked with the traditional CAE paradigm.

Conclusions

This research demonstrates that machine learning techniques can be used to provide vastly improved design support in parameterized domains. The designer is able to refine both decision and performance variables and can reuse the model for new performance specifications. The inverse engineering methodology has also been applied to process design as a *model translator* to convert a point-to-point simulator into a region-to-region model in a process planner for a turning machine. In a few design scenarios the design task is precisely defined and can be automated. This is the approach we are applying to support "worst-case" design of analog MOS circuits. The inverse engineering methodology opens up unexplored paradigms in knowledge processing by harvesting existing analysis-based simulators to ease the knowledge acquisition bottleneck. This methodology shows tremendous promise for solving a wide variety of problems in engineering decision making.

Acknowledgments

This work was begun while R. Bharat Rao was at the University of Illinois at Urbana-Champaign (UIUC) and was partially supported by the Department of Electrical Engineering. We are grateful to Sudhakar Yerramareddy, Allen Herman, and Prof. Dennis Assanis, all from the Department of Mechanical Engineering, UIUC.

References

Assanis, D.N. and Heywood, J.B. 1986. *The Adiabatic Engine: Global Developments.* 95–120.

Dixon, J.R. 1986. Artificial intelligence and design: A mechanical engineers view. In *AAAI-86.* 872–877.

Finger, S. and Dixon, J.R. 1989. A review of research in mechanical engineering design. part i: Descriptive, prescriptive, and computer-based models of design processes. *Research in Engineering Design* 1(1):51–67.

Friedman, J.H. 1991. Multivariate adaptive regression splines. *Annals of Statistics.*

Herman, A. E. 1989. An artificial intelligence based modeling environment for engineering problem solving. Master's thesis, M&IE, University of Illinois, Urbana, IL.

Langley, P.; Simon, H.A.; Bradshaw, G.L.; and Zytkow, J.M. 1987. *Scientific Discovery: Computational Explorations of the Creative Processes.* MIT Press.

Quinlan, J.R. 1986. Induction of decision trees. *Machine Learning* 1(1):81–106.

Rao, R. B. 1993. *Inverse Engineering: A Machine Learning Approach to Support Engineering Synthesis.* Ph.D. Dissertation, ECE, University of Illinois, Urbana.

Rao, R. B. and Lu, S. C-Y. 1992. Learning engineering models with the minimum description length principle. In *AAAI-92.* 717–722.

Rao, R. B. and Lu, S. C-Y. 1993. KEDS: A Knowledge-based Equation Discovery System for learning in engineering domains. *IEEE Expert* (to appear).

Rissanen, J. 1986. Stochastic complexity and modeling. *Annals of Statistics* 14(3):1080–1100.

Vanderplaats, G. N. 1984. *Numerical Optimization Techniques for Engineering Design - With Applications.* McGraw-Hill.

Yerramareddy, S. and Lu, S.C-Y. 1993. Hierarchical and interactive decision refinement methodology for engineering design. *Research in Engineering Design* (to appear).