Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.685 Electric Machines

Class Notes 11: Design Synthesis and Optimization
©2003 James L. Kirtley Jr.

1 Introduction

The notion of "Optimization" is, at first blush, fairly straightforward: it is finding a set of machine dimensions, materials, methods of assembly and so forth that constitute the "best" machine. In principle, it should be possible to figure out which machine is best and perhaps even to teach a computer program to seek out the optimal machine.

There are a few problems with this, however. The first and most difficult problem is that there is usually no way of knowing what the "best" machine is, *a priori*. All physical systems have a variety of *attributes*, and usually tradeoffs need to be made between them. Even if an appropriate set of tradeoffs has been made so that there is a scalar *objective function*, the design space is complicated by the existence of integer and integer-like variables and complex limits. In this chapter of the notes we will look at a small set of ways of handling this situation.
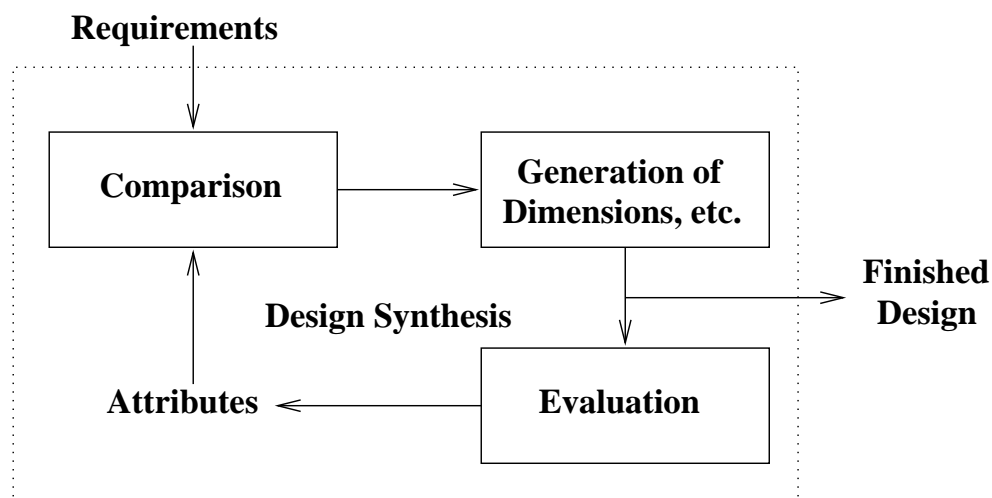

Figure 1: Design Synthesis

1.1 Design Synthesis

Illustrated in Figure 1 is a schematic representation of the design process. Here is some nomenclature:

Requirements are specifications which must be met by a design. They would include things such as rated power, limitations on tip speed and temperature, perhaps lower limits on efficiency

1

and power factor, etc.

Attributes are things which, all other things being equal, are to be either mimimized or maximized. These would include cost (to be minimized), efficiency (to be maximized), etc.

Feasibility is a binary ("go/no-go") variable which determines if a machine will work at all.

Design Variables are machine dimensions and material properties which fully specify a machine design.

Note that the distinction between requirements and attributes is not really clean: a requirement can be a limit on an attribute. However, in order to perform "optimization", it is necessary to have at least one attribute which can be maximized.

Now, the design process consists of two steps. One is generation of a set of design variables that constitutes a design, the other is evaluation of that design to establish feasibility and attributes. A machine which is *feasible* meets the requirements, and of course it makes sense to discuss the *attributes* only of feasible machines.

One way of setting up the process illustrated by Figure 1 is to use a design evaluator such as the MATLAB scripts that accompany these notes, with an expert machine designer serving to perform the synthesis step. So, for example, if a candidate machine has a power output that is too small, the designer might set the rotor length to be a bit larger and then run the evaluator again. This process works fairly well, particularly as the expertise of the designer improves.

In this set of notes, however, we will attempt to automate both the design evaluation *and* synthesis steps, so that the computer can produce finished (or nearly finished) designs. Our methodology here, which we do not claim to be the only way of accomplishing this task, involves random search (often called "Monte-Carlo") and multi-attribute evaluation. We have distilled these into a design methodology we call the "Novice Design Assistant".

2 The Pareto Surface and Dominance

There will usually *not* be a single attribute to be maximized, so it is necessary to carry out tradeoffs. Consider the situation illustrated in Figure 2. Here we have two attributes, labelled in the figure as "costs", to reinforce the notion that the "best" machine would be at the origin. Suppose "cost 1" is manufacturing cost and "cost 2" is the inverse of efficiency and so would represent the cost of operating losses.

There will be a feasibility frontier, shown as the solid curve in the figure. No machines can be made closer to the origin than this curve. We should note that the design process can be distilled into finding that curve and the machines whose attribute set lies on or very close to it. Now consider some specific cases. The circles labeled "A" and "B" are on that feasibility frontier. Neither of these machines is clearly better than the other, and the choice between them must be made by tradeoff of the attributes. Machine "A" would be best for a machine that does not run very much, while machine "B" would probably be best for a machine that operates a lot, due to energy costs.

Machines "C" and "D" are not *on* the design frontier, and are actually better examples of the sorts of machines that would be generated by a synthesis program. Note that we cannot, *a priori* say that these machines are inferior to either "A" or "B". However, we can see that machine "E" is inferior to machine "B" (it is both less efficient and costs more). We say that machine "E" is
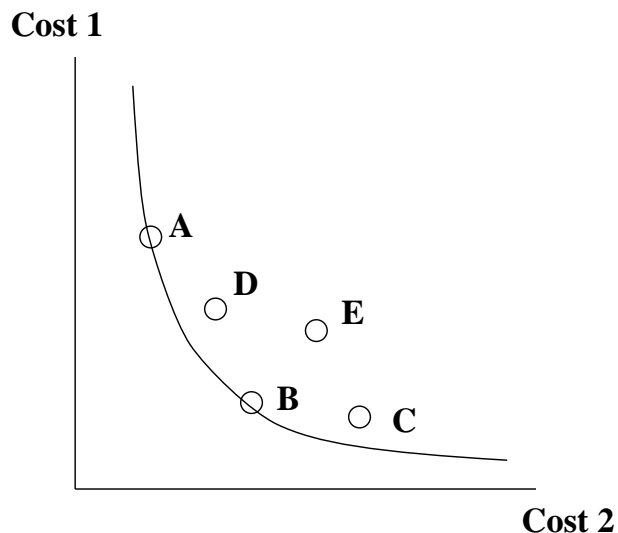
Cost 1

Figure 2: Energy Conversion Process

dominated by machine "B". Note that a good synthesis process would eventually find machines that are close to the design frontier than "C" and "D", so that those machines might eventually become *dominated*.

3   Monte Carlo Based Synthesis

There are many schemes for doing optimal design and we will not attempt to fully categorize them. However, it is probably useful to consider the difference between *deterministic* and *random* search processes. Deterministic search methods include grid search and hill climbing techniques. Random search methods are variations on "rolling the dice" and are therefore called "Monte Carlo".

One way of searching the design space is to evaluate all possible designs and then pick, somehow, the best one. A variation on this would be to use a "grid" search, evaluating all of the designs that are at intersections of a coarse grid in the design space. Then the grid could be re-oriented to be located around one or a few of the most promising nodes and to have shorter intervals between points. This process can be carried out until the grid spacing becomes small enough that the designer is confident that the "optimal" machine has been reached.

A second way of doing a deterministic search is to use a "hill climbing" routine, which starts at some point in design space and evaluates the gradient at that point. The design evaluator figures out which way is "up", and moves in that direction, usually with a fairly large step. If the next point is indeed better than the first, the gradient is evaluated and the process is repeated. If it is not better, usually the prior point is used with a reduced step size.

Grid search routines become computationally infeasible in design spaces with large dimensionality. Hill climbing routines don't work well in situations in which some variables are integer in nature (such as the number of turns or the type of steel to be used). Further, hill climbing techniques very often cannot handle situations in which there are *local optima*, since if a hill climber starts up the wrong hill, it has no way of knowing that it has found an inferior optimal solution. Because of this, random search routines are often use for optimization of complex designs. Random search is, as its

3

name implies, simply generating a random number or set of random numbers and using them for the design. In this note we will consider two distinct uses of Monte-Carlo techniques. One of these is the "Novice Design Assistant", the other is a scalar optimization routine.

It should be noted that Monte-Carlo search techniques work well in both of the situations that befuddle hill climbers: they can handle mixed continuous/integer variable situations and they are not (usually) troubled by local optima.

4   The Novice Design Assistant

One design synthesis method discussed above uses a real person in the design loop. Presumably an expert, this person would use knowledge of how machines work to perturb a trial design to make it conform to the requirements. There is probably no barrier to teaching a computer to do the same thing, and a set of "rules" for how to perturb the design could be developed. This would result in an "expert system" for designing electric motors.

The problem here is that, by distilling the role of the machine designer into an expert system, we would be freezing the state of technology to what was known and understood by that designer. We hope, in the machine design synthesis process, to be able to do better than that. An ideal process would have the ability to adapt to new material properties (we don't expect Maxwell's Equations to change with time) and requirements, and this is a major consideration in our rejection of an expert system to do the synthesis.

The design paradigm we call the "Novice Design Assistant" (NDA) is deliberately *not* an expert system. It employs a Monte Carlo scheme for generating candidate designs and multi-attribute evaluation of those designs. It also employs the notion of *dominance* to eliminate all designs found to be inferior to others. Also, many designs will be *infeasible*, and those designs are discarded.

Key to the NDA is a function script which in this implementation must be called `attribut.m`. This script returns two values: a *scalar, binary* number C, which is 1 if the design is feasible and 0 if the design is infeasible, and a *vector* of *attributes*. This function script does an evaluation of each design. It is generally good for efficiency reasons to do feasibility checks early and return a zero value of C for unfeasible designs without wasting a lot of cycles doing a complete evaluation.

For all of the continuous variables in the design space, the NDA required four vectors: lower and upper limits, a mean value and a standard deviation. This gives the program quite a lot of flexibility in operation: a small standard deviation concentrates all searches in a narrow range, while a large deviation searches over the whole space. Integer variables are given equal probabilities over their whole range of values. In the example that follows this chapter of the notes, those variables are set up by the script `msetup.m`.

The NDA establishes and keeps a list of feasible, non-dominated designs. It generates a set of random numbers and invokes `attribut.m`. When an evaluation is made, the returned design is compared with each of the designs in the list. If it is dominated it is discarded. If it dominates one of the designs on the list, it replaces that design. If it dominates more than one design on the list, it replaces the first one and all dominated designs are discarded.

4

5   Scalar Objective Function Optimization

Finally, under some circumstances a scalar *objective function* is possible, permitting a real optimization. The objective here is to obtain a design with the largest (or smallest) possible value of that objective function. There are, of course, many ways of generating this design. For example, in the MATLAB Optimization Toolbox is `constr.m`, a function which generates a constrained optimization of a scalar function.

Attached to these notes is an optimizer based on Monte-Carlo techniques. This is quite similar in structure to the NDA. What it does is regard the first element of the returned vector A from attribut.m as a variable to be minimized. In the example we use an evaluation of material cost to be that value. What `ropt.m` does is generate a random step in the design space, call `attribut.m` to evaluate the new design, compare that one with the old one and keep the best one. It does this a fixed number of times, reduces the standard deviation used in generating the random numbers and repeats the process. This is done a number of times determined when the program is first called. For details, see the code and try running it!

6   The Example

The code contained in the attachment to these notes is for a permanent magnet machine evaluated as a combination starter-motor/generator for an automobile. We do not make any representations with regard to the viability of the machine so designed, but it serves as a reasonable example. We suggest that you may want to adapt some other design program to running with the NDA, using this as an example.

6.685 Electric Machines
Fall 2013