Design Optimization of Large and Complex Systems

Optimal Design Laboratory
University of Michigan
Outline

- Optimal Systems Design
  - Challenges
    - Large-scale
    - Complex
    - Multiobjective
Optimal Design and Nonlinear Programming

\[ \min_x \ f(x) \]

s.t. \[ g(x) \leq 0 \]

\[ h(x) = 0 \]

where \( x \in \mathbb{R}^n, f : \mathbb{R}^n \rightarrow \mathbb{R}^m, \)
\[ g : \mathbb{R}^n \rightarrow \mathbb{R}^k, h : \mathbb{R}^n \rightarrow \mathbb{R}^l, \]

and \( f, g, \) and/or \( h \) are nonlinear
Design vs. Analysis Models

- **Analysis Models**
  - Take design variables and parameters as inputs
  - Return responses for current design
  - Can be explicit functions, response surface models, simulations, or spreadsheets

- **Design Models**
  - Call analysis models to make design decisions
  - Are composed of design objectives and constraints
Challenges: Large-scale

- n, k, and/or l are big
- Prohibitively expensive computations
- Results from (nonlinear) optimization algorithms become unreliable as the size of optimization model grows
- Engineering tradeoffs implied by the computer-generated numbers may not be adequately interpretable
Solution for Large-scale: Decomposition

- Breaks up a large-scale problem into a number of loosely linked smaller, more manageable subproblems.
- Decreased problem size enhances robustness and speed of numerical solution algorithms.
- The subproblems can be solved in parallel, and different optimization techniques most suitable for the specifics of the subproblems can be utilized.
- May reflect the typical multidisciplinary nature of system design problems.
Decomposition Example
Model-based Decomposition

POWERTRAIN ANALYSIS

- HEAT TRANSFER
- THERMODYNAMICS & COMBUSTION
- MULTIBODY DYNAMICS

- FATIGUE
- CONTROL
- ACOUSTICS
- STRUCTURAL
- FLUIDS
- CHEMICAL
Optimal Model-based Decomposition Approach

- Represent design model with a hypergraph
- Use graph partitioning techniques to identify an optimal partition of the design model
The University of Michigan

FDT-based methodology

\[ \text{min } f = f_1 + f_2 + f_3 + f_4 \]
\[ x \in [0.1, 10]^8 \]

subject to
\[ f_1 = 0.4x_1^{0.67} x_7^{-0.67} \]
\[ f_2 = 0.4x_2^{0.67} x_8^{-0.67} \]
\[ f_3 = 10 - x_1 \]
\[ f_4 = -x_2 \]
\[ g_1 = 0.1x_1 + 0.06x_5 x_7 - 1 \leq 0 \]
\[ g_2 = 0.1x_1 + 0.1x_2 + 0.06x_6 x_8 - 1 \leq 0 \]
\[ g_3 = 4x_3 x_5^{-1} + 2x_3^{-0.71} x_5^{-1} + 0.06x_3^{-1.3} x_7 - 1 \leq 0 \]
\[ g_4 = 4x_4 x_6^{-1} + 2x_4^{-0.71} x_6^{-1} + 0.06x_4^{-1.3} x_8 - 1 \leq 0 \]
Hypergraph representation
Hypergraph-based Partitioning

- Minimize interconnection among subproblems
- Balance size of partitions

$x_1, x_2, x_3$ are linking variables
OPTIMAL MODEL- BASED DECOMPOSITION OF DESIGN PROBLEMS

Hypergraph Partitioning Approach

Input Data

Browser or specify the path of the file containing the Functional Independence Table (FIT). The format of this file is available here:

- Path of file containing Functional Independence Table (FIT): Browse or
- Choose Functional Independence Table here.

Browser or specify the path of the file containing the design relation weights and the design variable weights. Choose one of the default choices. A description of the file format is available here:

- Path of file containing design relation weights: Browse or
- Design relation weights equal to one: or
- Design relation weights equal to the number of variables in relation: or
- Design relation weights equal to the number of variables in relation: or
- Path of file containing design variable weights: Browse or
- Design variable weights equal to one: or

Enter the number of subproblems you want the design problem to be divided into:

- Number of subproblems: or
- Number of parts: or

Browser or specify the path of the file containing the subproblem sizes. A description of the file format is available here. Also enter the allowed imbalance between subproblem sizes, as a percentage of the average size:

- Path of file containing subproblem sizes (if any): Browse or
- Allowable deviation of subproblem sizes (in %): or

problem specs
variable weights
relation weights
number of parts
partition sizes
SP1: Wheel model, powertrain and vehicle geometry relations; acceleration, starting gradeability, and cruising velocity criteria

SP2: Engine relations

SP3: Torque converter, transmission, and powertrain geometry relations

SP4: Engine relations; anti-lug constraint; emissions and fuel consumption criteria

- 22 linking variables
Coordination

Non-Hierarchically Partitioned Problem

Hierarchically Partitioned Problem

Subproblem local variables

linking var’s

Subproblem local variables

Subproblem local variables

Master Problem linking var’s

linking variables

Subproblem local var’s

Subproblem local var’s

Subproblem local var’s
Hierarchical Coordination: Sequential Decomposition

- NLP algorithms are modified to accommodate hierarchical structure while still retaining theoretical properties of the original algorithm (global & local convergence).

- Add a step to the existing NLP during which the linking variables are held constant and the subproblems are solved.

- Subproblems are used not only to improve their respective objectives but also to give better estimates of other quantities (e.g., Hessian estimates, penalty parameters, trust region radii).
Sequential Decomposed Quadratic Programming

1. Start at some initial point
2. If current iterate is not near the solution, treat linking variables as parameters and solve each subproblem:
   - Compute search direction
   - Do line search
   - Check convergence
   - Generate Hessian estimates
3. Compute search direction using entire model
4. Do line search
5. Check convergence
6. Generate Hessian estimates
Hierarchical Overlapping Coordination (HOC)

- Uses two or more model decompositions, which should have specific characteristics
- Each decomposition “coordinates” the others, i.e., there is no Master Problem
- Convergence for convex programs with linear constraints depends on model decomposition, for example:
  - Reduced number of linking variables
  - Disjoint set of linking variables
$\alpha$-Decomposition
\[ \min f(x) \quad \text{s.t.} \quad A_E x = c_E, A_I x \leq c_I \quad \text{and} \quad H_\alpha x = y_\alpha \]

$\beta$-Decomposition
\[ \min f(x) \quad \text{s.t.} \quad A_E x = c_E, A_I x \leq c_I \quad \text{and} \quad H_\beta x = y_\beta \]

Decomposition of Design Vector
\[ x = x_I \]
\[ y_\alpha \Leftrightarrow x_A \]
\[ y_\beta \Leftrightarrow x_B \]
Challenges for Complexity

- Some of the f, g, and/or h are multidisciplinary expensive computations or simulations
- Simulations can be coded in many different programming languages
- Expense of numerical techniques can be very large
- Models tend to contain excessive noise from the numerical techniques, making derivatives often inaccurate
- Simulation-based models also can contain discontinuities due to discrete decision variables
- Simulations tend to be reliable to only a few significant digits
Issues related to Simulation-based Design

- Integration of different languages/simulations
  - Wrapping
  - Object-Oriented Optimization

- Computational expense
  - Surrogate Models
  - Distributed Computing (across network)

- Numerical Noise & Discontinuities
  - Scaling, Finite-Difference Steps & Auto-differentiation
  - Derivative-Free Algorithms
  - Response Surface Methods
Address Integration Issues

- Object-Oriented Optimization
  - Facilitates Distributed Computing
  - Creates a Standard for Client/Server Integration (Plug & Play)

```
Eval(x,[f,g])
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```

```
Run(x,y)
```
Address Computation Issues

- Surrogate Models
  - Black-box Simulation
  - Data Sample & Fit (Kriging, ANNs, etc)

- Distributed Computing (across network)
  - Client
  - Server: Machine A
  - Server: Machine B
  - Server: Machine C
  - Server: Machine D
  - Results
Metamodelling

- Design of Experiments (e.g., orthogonal arrays, Latin Hypercube)
- Polynomials
- Artificial Neural Networks
- Kriging
- Response Surfaces
Exploit model inventory

Create identifiers:
- Document Type Description (DTD)
- XML-based

Develop “glueing” methods to address:
- Time dependency
- Causality
Numerical Noise and Discontinuity Issues

- Numerical noise
- Unimodal function
- Multimodal function
- Disconnected feasible region
Discontinuity Issues: Traditional Remedy

- Scaling & Finite-Difference Step
Derivative-Free Algorithms
- Search Algorithms (DIRECT, PDS)
- Genetic Algorithms (GA)
- Simulated Annealing (SA)

Algorithms for Mixed Variable Programming
- Categorical variables (e.g., which material to use)
Discontinuity Issues: Novel Techniques (cont.)

- Response Surface Methods
  - SMO (Sequential Metammodel Optimization)
  - EGO (Efficient Global Optimization)
1. Fit smooth metamodel to data sample
2. Use optimization algorithm to find optimum of metamodel
3. Fit new metamodel smaller area around optimum found in (2)
4. Go back to (2) until convergence (area around optimum is sufficiently small)
1. Fit Kriging metamodel to data sample

2. Use optimization algorithm to find optimum of Expected Improvement function

3. Add point from (2) to data sample

4. Go back to (1) until convergence (Expected Improvement is sufficiently small)
Multiobjective optimization: $f$ is a vector

Define scalar function, e.g.,

$$\min_x \sum_{i=1}^{m} w_i f_i(x)$$

Challenge: How to select the weights
Minimize:
\[ f = (-) w_1 * f_{\text{HEV}}(x_{\text{HEV}}) + w_2 * f_{\text{CVT}}(x_{\text{CVT}}) \]

Subject to:
\[ g_{1-10-\text{HEV}} \]
\[ g_{1-8-\text{CVT}} \]
case 1:
\[ h_1 = (\text{engine size})_{\text{HEV}} = (\text{engine size})_{\text{CVT}} \]
case 2:
\[ h_1 = (\text{final drive})_{\text{HEV}} = (\text{final drive})_{\text{CVT}} \]
The University of Michigan

Pareto Sets

Pareto Set Defining Design Trade-Offs

- Graph showing the trade-off between 0-60 mph time (Conventional-CVT) and mpg - combined (Parallel-HEV)
- Three platforms: Common Engine, Common Final Drive, Null Platform
- The graph illustrates how different design choices affect performance metrics.