A Critique of Optimization Modeling Environments for Complex Engineered Systems

Michael Bynum, William Hart, Jordan Jalving, Carl Laird, Bethany Nicholson, John Siirola

Sandia National Laboratories

wehart@sandia.gov
Elements of the solution of complex optimization applications

<table>
<thead>
<tr>
<th>Application Formulation</th>
<th>Optimization Equations</th>
<th>Modeling Environment</th>
<th>Optimization Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification of application goals and decision options</td>
<td>Mathematical representation of application goals</td>
<td>General-Purpose tools for expressing models</td>
<td>Optimization solvers identify best solutions</td>
</tr>
</tbody>
</table>

**Our focus today**

\[
\begin{align*}
\text{Attacker: Maximize Damage/Disruption} & \\
\text{Defender: Minimize Losses} & \\
\max_{x \in [0,1]} F(x, y) & \\
\text{s.t.} & \\
G(x, y) & \leq 0 \\
\min_{y \geq 0} F(x, y) & = f(x, y) \\
g(x, y) & \geq 0
\end{align*}
\]
Examples of optimization modeling environments

- Prescient
- EGRET
- PyTorch
- TensorFlow
- AMPL
- GAMS
- Julia
- MATLAB
- PYOMO
- IBM CPLEX
- DAE
- POEK
- COEK
- Gravity
- ExaGO
Impact of optimization modeling environments

1. **Simplify expression of complex applications**
   - Intuitive algebraic expressions
   - Compact mathematical notation
   - Domain-specific problem representations

2. **Automate grungy parts of the computational workflow**
   - Automatic differentiation
   - Application of model transformations

3. **Facilitate transformations between problem representations**
   - Transformations to simplify the problem formulation
   - Transformations to tailor problem to solver requirements
How do we effectively use general-purpose modeling environments on emerging computational platforms?
Some perspectives on future modeling environments

1. Performance optimization
2. Application-centric vs Solver-centric
3. SME- vs Data-driven models
Can we use the modeling environment to tailor calculations in the optimization workflow?

**Idea 1**: Use code generation or automatic differentiation to tailor sparse, unstructured derivative calculations to target hardware

**Explicit Code**
Derivatives are coded by hand.

Pro:
- Can directly tailor calculations to HW

Con:
- Hand coding can be error-prone

**Code Generation**
Code is generated automatically for derivatives.

Pro:
- Can tailor code generation to HW

Con:
- Complex SW code management

**Automatic Differentiation**
Derivatives computed numerically with AD

Pro:
- Flexible model for applications

Con:
- Complex mapping of AD to HW
Can we use the modeling environment to tailor calculations in the optimization workflow?

**Idea 2:** Rethink workflow to exploit model structure

- AMPL Model → NL File → Solver
- JuMP Model → Solver C-API → Solver
- Pyomo Model → JSON File → COEK Model → NMPC → Solver

**Static problem representation**

**Code generation for embedded applications (e.g. control)**
Can we use the modeling environment to tailor calculations in the optimization workflow?

**Idea 3:** Exploit model structure to parallelize optimization workflow

- **Coarse-Grain Decomposition**
- **Fine-Grain Mapping of Dense Kernels**
- **Sparse, Irregular Computations?**

Parapint: Parallel-in-time decomposition

Dense matrix-matrix multiplication on tensor cores

ARIAA: Mapping execution graphs onto data flow architectures
Can we use the modeling environment to tailor calculations in the optimization workflow?

**Idea 4:** Exploit model structure to *automatically* interface with distributed optimization solvers

Unstructured Coarse-Grain Decomposition

- Can we automatically partition model generation across processors?
- How do we robustly parallelize model transformations?
- How do we coordinate parallel model generation and setup of parallel solver?
How can we leverage modeling environments to both express problem structure and inform the optimization workflow?

**Challenge**: Where do we construct the model and apply model transformations in our optimization workflows?

Application interaction is naturally supported by the CPU, but we want to exploit solvers running on GPUs.

Do we represent models on CPUs or GPUs or both?

If “both”, then how do we manage multiple representations?
How can we leverage modeling environments to both express problem structure and inform the optimization workflow?

**Idea 1**: Model expansion on GPUs

- Continuous domains
- Ordinary differential equations
- Partial differential equations
- Systems of differential algebraic equations
- Higher order differential equations and mixed partial derivatives

**Pyomo DAE:**
1. Used to express model dynamics within Pyomo
2. Includes model transformations to discretize the model (e.g. Collocation)

**Key Idea:**
- Use transformation to expand model directly on GPU
- Can leverage HW-specific features to tailor model calculations
How can we leverage modeling environments to both express problem structure and inform the optimization workflow?

Idea 2: Facilitate user- and solver-specific representations

E.g. the PAO library supports multiple problem representations

It is easy to write solvers for models with matrix/vector data

Key Idea:
- Explicitly support CPU- and GPU-specific model representations
How can we leverage modeling environments to both express problem structure and inform the optimization workflow?

**Idea 3**: Use modeling environments that facilitate the application of model transformations on CPUs, GPUs and between them

Blocks can be transformed in a modular manner

E.g. locally transform complementarity conditions to big-M representation

**Pyomo**

**AMPL, GAMS, ...**
Can we develop effective strategies to integrate both SME and data-driven modeling strategies?

**Challenge**: Evolve our modeling capabilities to support a continuum of application needs

---

**SME-driven Models**

- AMPL
- PYOMO

**Data-driven Models**

- Julia
- PyTorch
- TensorFlow

---

Note: There have been few demonstrations of capabilities “in the middle”
Can we develop effective strategies to integrate both SME and data-driven modeling strategies?

**Idea 1:** Augment SME models with embedded data-driven models

E.g. SNL Redly project: use neural network to replace contingency constraints

![Diagram showing nominal operation (ACOPF) with contingencies 1, 2, and 3. Training neural network (NN) on security boundary.]

The diagram illustrates the process of training a neural network (NN) on a security boundary and applying it to replace contingency constraints. The goal is to develop optimization modeling environments that can effectively represent large ML models.

We need optimization modeling environments that can effectively represent large ML models.
Can we develop effective strategies to integrate both SME and data-driven modeling strategies?

Idea 2: Data-driven methods tailored for specific application domains

E.g. Partition of unity networks

Partition of unity networks mimic the structure of traditional finite elements

Data-driven exterior calculus discovers bilinear form that conserves mass/momentum/energy without knowing underlying physics

“A key challenge is optimization methods that can handle general nonlinear constraints.”

“Partition of unity networks: deep hp-approximation.”
Lee, et al. 2021
The End