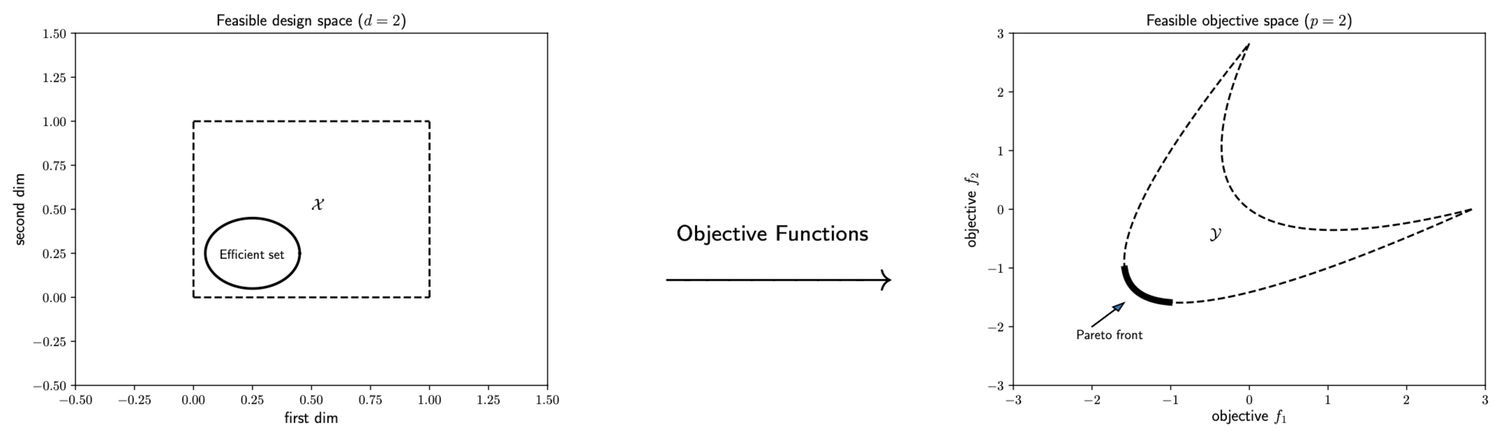
A framework for fully autonomous design of materials via multiobjective optimization and active learning

Tyler Chang[1], Jakob Elias[1], Stefan Wild[2], Santanu Chaudhuri[1,3] and Joseph Libera[1]

[1]Argonne National Laboratory [2]Lawrence Berkeley National Laboratory [3]University of Illinois Chicago

Multiobjective Optimization

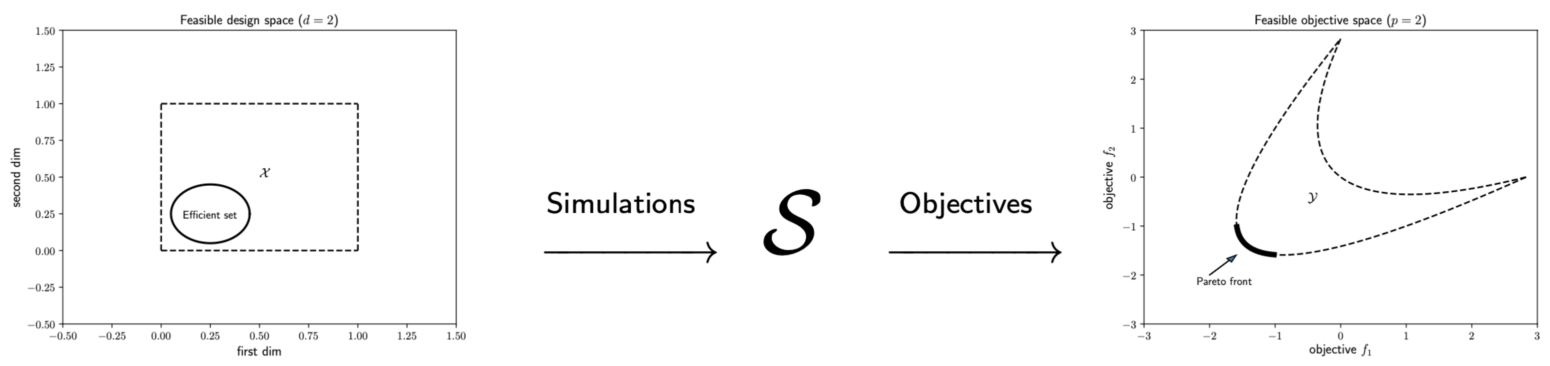
$$\begin{aligned} \underset{x \in \mathcal{X}}{\text{minimize}} \quad & F(x) \\ \text{subject to} \quad & G(x) \leq 0 \end{aligned} \qquad \begin{aligned} \mathbf{F(x)} &= (\mathbf{f_1(x)}, \mathbf{f_2(x)}, \ldots, \mathbf{f_o(x)}) \\ \mathbf{G(x)} &= (\mathbf{g_1(x)}, \mathbf{g_2(x)}, \ldots, \mathbf{g_p(x)}) \end{aligned}$$



Design space *Objective space*

Multiobjective *Simulation* Optimization

$$\begin{aligned} \underset{x \in \mathcal{X}}{\text{minimize}} \quad & F(x, \mathbf{S(x)}) \\ \text{subject to} \quad & G(x, \mathbf{S(x)}) \leq 0 \end{aligned} \qquad \mathbf{S(x)} = (\mathbf{s_1(x)}, \mathbf{s_2(x)}, \ldots, \mathbf{s_m(x)})$$



Design space *Objective space*

Common Simulation-based Structures

Sum-of-squared simulation outputs:

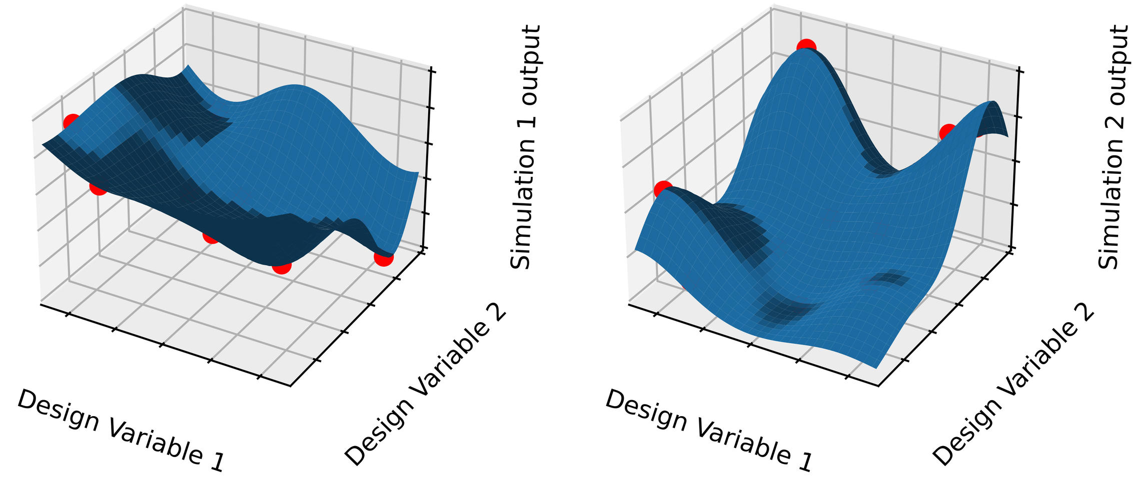
$$\min_{x \in \mathcal{X}} \left(\sum_{i=1}^{m_1} \mathbf{S_i(x)^2}, \quad \sum_{j=1}^{m_2} \mathbf{S_j(x)^2} \right)$$

One simulation, **one algebraic** objective:

$$\min_{x \in \mathcal{X}} \left(\mathbf{S(x)}, \quad \sum_{i=1}^{n} \mathbf{x^2} \right)$$

Response Surface Methodology

- **Search/sample** data for raw **simulations outputs**
- Use **surrogates** to model **simulations**, **not objectives**
- Separately define **objectives and constraints**
- Scalarize **objectives** using **acquisition functions**
- **Solve scalarized surrogate problems** and iterate



Design Principles

Mix-and-match

- Initial search (design-of-experiments)
- Surrogate models
- Acquisition/scalarization functions
- Scalar optimization solvers

Easy for users and developers

- Support for variety of design vars and simulations
- Support various scientific workflows
- Embed/extract problems from unit cube

Flexible problem definitions

- Add design vars, sims, objs, + constraints
- Add searches, surrogates, acquisitions, optimizer
- Solve serially or in parallel using `libEnsemble`

A Sample Script

```python
import numpy as np
from parmoo import MOOP, optimizers, searches, \
                   surrogates, acquisitions

# Create a MOOP object with LocalGPS solver
moop = MOOP(optimizers.LocalGPS)

# Add 2 design variables
moop.addDesign({"name": "x1", "des_type": "continuous",
                "lb": 0.0, "ub": 1.0})
moop.addDesign({"name": "x2", "des_type": "categorical",
                "levels": 3})

# Define and add 1 simulation with 2 outputs
def sim(x): return np.array([x["x1"]**2 + x["x2"],
                             (x["x1"] - 1)**2 + x["x2"]])
moop.addSimulation({"name": "MySim", "m": 2,
     "sim_func": sim, "search": searches.LatinHypercube,
     "surrogate": surrogates.GaussRBF})

# Add 2 objectives: minimize each sim output
def f1(x, s): return s["MySim"][0]
def f2(x, s): return s["MySim"][1]
moop.addObjective({"name": "f1", "obj_func": f1})
moop.addObjective({"name": "f2", "obj_func": f2})

# Add 1 constraint: x["x1"] <= 0.1
def g1(x, s): return 0.1 - x["x1"]
moop.addConstraint({"name": "g", "constraint": g1})

# Add 3 acquisition functions
for i in range(3):
   moop.addAcquisition({"acquisition":
                acquisitions.UniformWeights})

# Solve with 5 iterations and print the solution
moop.solve(5)
print(moop.getPF())
```

Download ParMOO

- `git clone https://github.com/parmoo/parmoo`
- `pip install parmoo`

parMOO pypi

Continuing Work

- Continue to add new solvers and techniques
- Support wider variety of problems & workflows