ParMOO: A Python library for parallel multiobjective simulation optimization

Tyler Chang[a] and Stefan Wild[a,b]

[a]Mathematics and Computer Science Division,
Argonne National Laboratory

[b]Applied Mathematics and Computational Research Division,
Lawrence Berkeley National Laboratory

SIAM CSE 23

Outlines

Introduction to MOSO + my experience

3 challenges + solutions

ParMOO software design + release

Example Problems

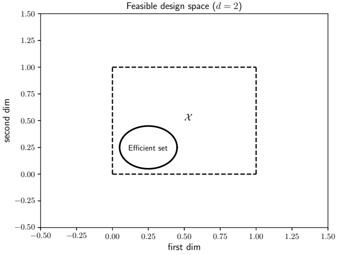
Conclusions + some closing thoughts

Multiobjective Optimization Problems

$$\min_{x \in \mathcal{X}} F(x)$$

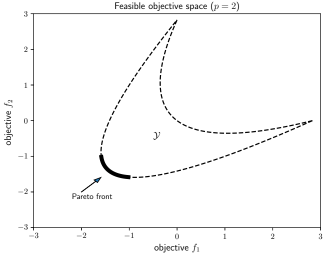
Multiobjective Optimization Problems

$$\min_{x \in \mathcal{X}} F(x)$$



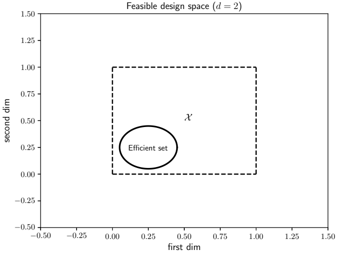
$$\longrightarrow$$

$$F : \mathcal{X} \to \mathcal{Y}$$



Multiobjective Optimization Problems

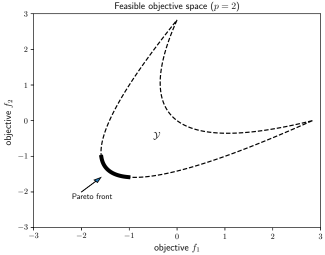
$$\min_{x \in \mathcal{X}} F(x)$$



$$\longrightarrow$$

$$F : \mathcal{X} \to \mathcal{Y}$$

expensive
blackbox process

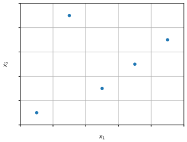


Multiobjective Response Surface Methodology

or Model-Based Optimization or Active Learning

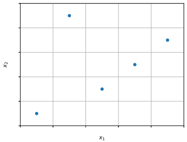
Multiobjective Response Surface Methodology

or Model-Based Optimization or Active Learning



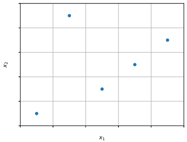
Multiobjective Response Surface Methodology

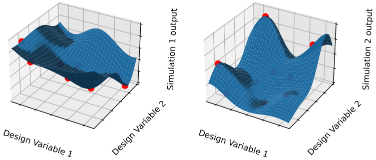
or Model-Based Optimization or Active Learning



Multiobjective Response Surface Methodology

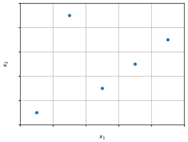
or Model-Based Optimization or Active Learning

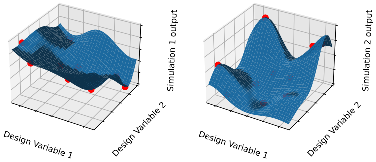


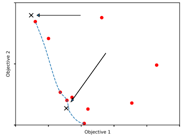


Multiobjective Response Surface Methodology

or Model-Based Optimization or Active Learning

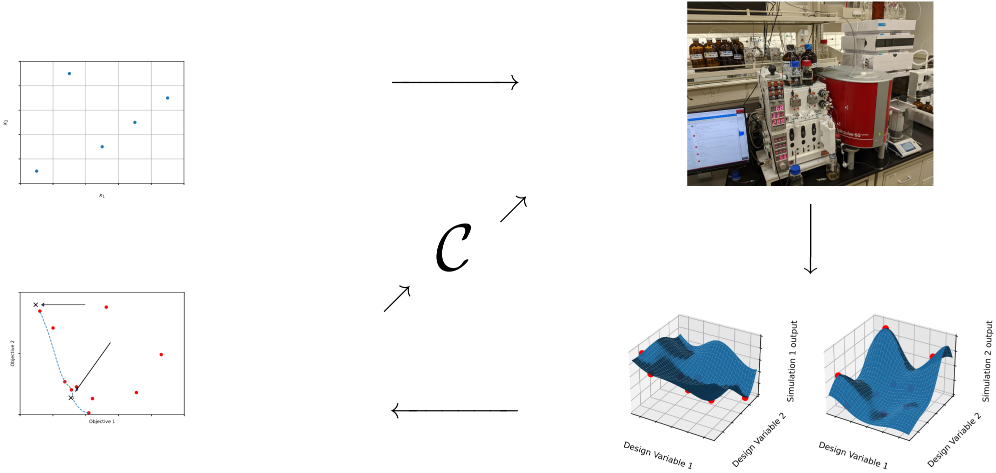




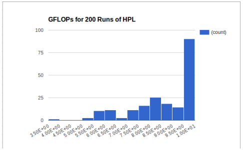


Multiobjective Response Surface Methodology

or Model-Based Optimization or Active Learning



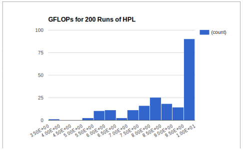
Example: HPC Performance Tuning



VT VarSys Project – 40 runs of HPL

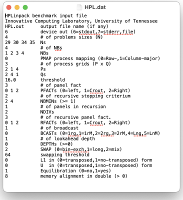
ANL – LCRC Computing Resources: Bebop

Example: HPC Performance Tuning



VT VarSys Project – 40 runs of HPL

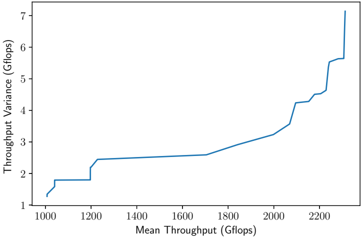
ANL – LCRC Computing Resources: Bebop




```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
4            # of problems sizes (N)
29 30 34 35  Ns
4            # of NBs
1 2 3 4      NBs
0            PMAP process mapping (0=Row-,1=Column-major)
3            # of process grids (P x Q)
2 1 4        Ps
2 4 1        Qs
16.0         threshold
3            # of panel fact
0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
2            # of recursive stopping criterium
2 4          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
3            # of recursive panel fact.
0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
0            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

VTMOP solver

$\longrightarrow$



[1] Chang, Larson, and Watson. Multiobjective optimization of the variability of the high-performance LINPACK solver. In Proc. WSC 2020.

Challenge 1:
mixed vars + problem types

Example: Particle Accelerator Design

Example: Parallel Runs

Challenge 2:

parallel evals + computing environments

ParMOO Design Criteria

Design goals:

1. Highly customizable framework for multiobjective RSM
2. Exploit structure and domain knowledge simulation-based optimization problems
3. Flexible problem types (mixed-variables, constraints, etc.)

ParMOO Design Criteria

Design goals:

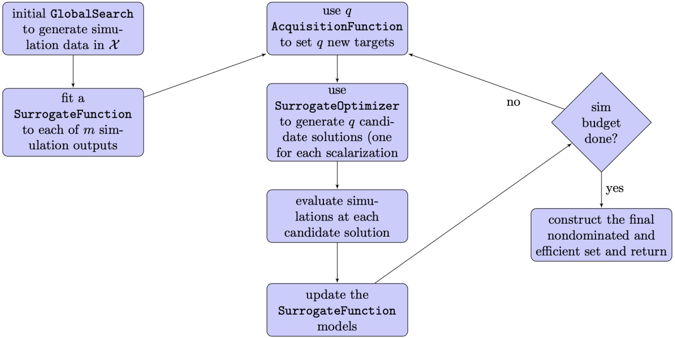
1. Highly customizable framework for multiobjective RSM
2. Exploit structure and domain knowledge simulation-based optimization problems
3. Flexible problem types (mixed-variables, constraints, etc.)

Design constraints:

1. Easy to deploy (parallelism, checkpointing, logging, flexibility)
2. Easy to maintain and extend
3. Easy to use (clean interfaces)

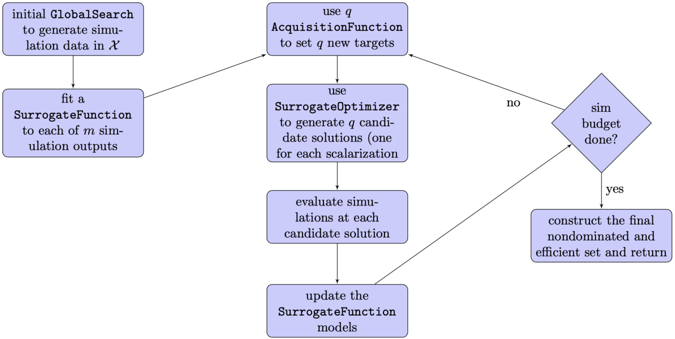
Customizability

ParMOO uses an object-oriented framework:



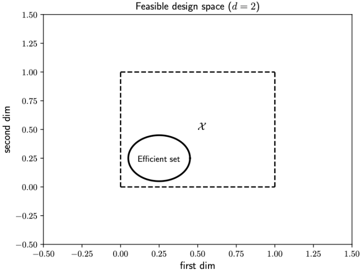
Customizability

ParMOO uses an object-oriented framework:



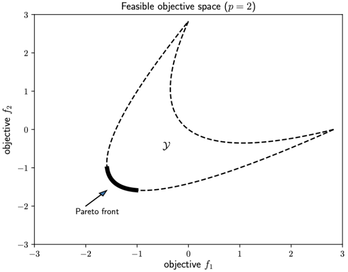
- Search/DOE
- Surrogate model
- Acquisition function
- Single-obj solver

Simulation Structure



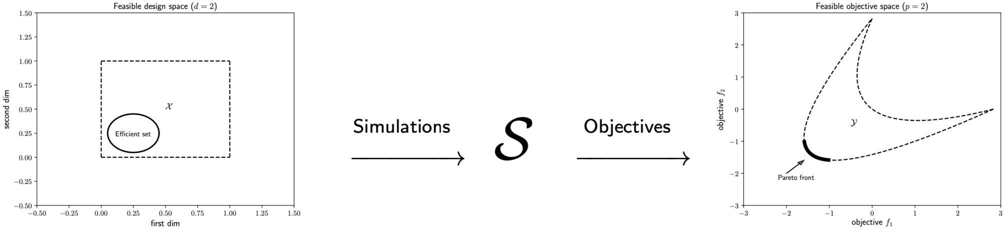
Design space

Objective Functions



Objective space

Simulation Structure



Design space *Objective space*

Simulation Structure

Simulation Structure

$$f_i(x) = h_i(x, S(x)) \qquad i = 1, \ldots, o$$

Simulation Structure

$$f_i(x) = h_i(x, S(x)) \qquad i = 1, \ldots, o$$

Sum-of-squares structure:

$$h_i(x, S(x)) = \sum_{j \in N_i} \left(S_j(x)\right)^2$$

where each $N_1, \ldots, N_o$ is an index set.

Increases order of approximation $\Rightarrow$
increases order of convergence

Simulation Structure

$$f_i(x) = h_i(x, S(x)) \qquad i = 1, \ldots, o$$

Sum-of-squares structure:

$$h_i(x, S(x)) = \sum_{j \in N_i} \left(S_j(x)\right)^2$$

where each $N_1, \ldots, N_o$ is an index set.

Increases order of approximation $\Rightarrow$ increases order of convergence

Heterogeneous MOOPs:

$$h_1(x, S(x)) = S_1(x)$$
$$h_2(x, S(x)) = \|x\|^2$$

Use expensive surrogate models for h_1 (i.e., S_1) but not for h_2

Flexible Problem Types

Flexible Problem Types

- Mixed variable-types

Flexible Problem Types

- Mixed variable-types **Ex: categorical variables**

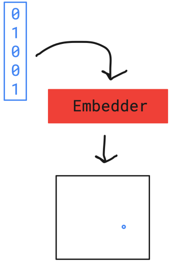
Flexible Problem Types

- Mixed variable-types **Ex: categorical variables**
  - Focus on *embedding* into continuous space

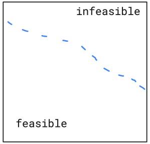
Flexible Problem Types

- Mixed variable-types **Ex: categorical variables**
  - Focus on *embedding* into continuous space
- (Nonlinear) constraints

Flexible Problem Types



- Mixed variable-types **Ex: categorical variables**
  - Focus on *embedding* into continuous space



- (Nonlinear) constraints
  - Focus on *augmented Lagrangian* penalties (relax to augmented unconstrained problem)

Design constraints

Design constraints

Easy to use:

- OOP gives a clean interface for adding attributes to a MOOP instance
- Total modularity (free to mix-and-match)

Design constraints

Easy to use:

- OOP gives a clean interface for adding attributes to a MOOP instance
- Total modularity (free to mix-and-match)

Easy to deploy:

- Simulation/experiment evaluations are only called in the `MOOP.solve()` method
- Extend `MOOP` class and overwrite `solve()` to deploy in different workflows
- **Ex:** Deploy parallel solvers on HPC systems using `libEnsemble`

Design constraints

Easy to use:

- OOP gives a clean interface for adding attributes to a MOOP instance
- Total modularity (free to mix-and-match)

Easy to deploy:

- Simulation/experiment evaluations are only called in the `MOOP.solve()` method
- Extend `MOOP` class and overwrite `solve()` to deploy in different workflows
- **Ex:** Deploy parallel solvers on HPC systems using `libEnsemble`

Easy to maintain and extend:

- OOP + total modularity makes adding new features easy
- Agile development with continuous integration
- Well-documented interface, contributing, and release process

ParMOO Release

parMOO

Written in `Python` (available on `pip` and `GitHub`)

`https://parmoo.readthedocs.io/en/latest/quickstart.html`

Combine with `libEnsemble` to use parallel solvers

Chang and Wild. ParMOO: A Python library for parallel multiobjective simulation optimization. *JOSS 8(82):4468 (2023).*

Example 1: Fayans EDF Model Calibration

Find params $x \in [0,1]^{13}$ to fit the Fayans model to data d_i:

$$M(\xi_i; x) \approx d_i \qquad i = 1, \ldots, 198$$

ParMOO simulation:

$$S_i(x) = M(\xi_i; x) - d_i, \qquad i = 1, \ldots, 198;$$

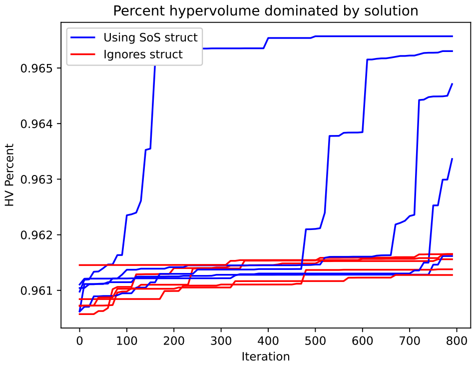
Min SOS across 3 observable classes

$$F_t = \sum_{i=1}^{m_t} \left(S_{t,i}(x)\right)^2$$

Bollapragada et al. Journal of Physics G: Nuclear and Particle Physics 48(2), 2020.

Fayans Solution with ParMOO

- Approximated Fayans model using inv dist weighting on existing dataset
- Implemented parallel solver in ParMOO using libEnsemble
- Just **14-25 lines of Python code**
- Ran for **10K** sim evals
- Compared against same solver w/o exploiting SOS structure



Example 2: Material Manufacturing with ParMOO

Choose optimal settings for material manufacturing in a continuous flow reactor (CFR)

We know how to make a desired material, need to produce at scale:

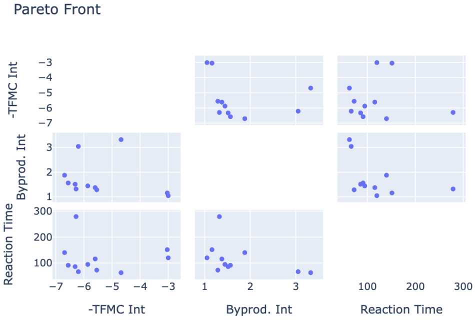
1. **Maximize the product** (battery electrolyte: TFML)
2. Can increase temperature to **reduce reaction time**
3. Too much heat activates a side reaction; need to **minimize unwanted byproduct**

Challenges:

- Mixed variable types
- Heterogeneous objectives
- Must send experiments to run on CFR

CFR Optimization with ParMOO

- Extend `MOOP` class to send/receive experiment data using `MDML` library (Apache Kafka)
- Used categorical variable embeddings
- Modeled Product/Byproduct as simulations and reaction time using algebraic equation of input



Next Release

Coming in v. 0.2

- Interactive post-run visualization tools
- Support for customized embeddings and passing raw (unscaled) inputs
- MDML (Apache Kafka) interface for distributing simulation evalutations
- (Maybe) advanced techniques for design-of-experiments

Resources

E-mail: tchang@anl.gov
E-mail: parmoo@mcs.anl.gov

JOSS 8(82):4468 (2023)

GitHub: github.com/parmoo/parmoo
Docs: parmoo.readthedocs.io
PyPI: pip install parmoo

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, SciDAC program under contract number DE-AC02-06CH11357.