ParMOO: A parallel framework for multiobjective simulation optimization problems

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 Berkley National Laboratory

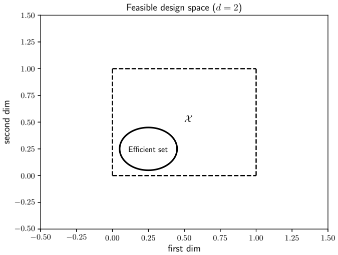
SIAM CSE 23

Outlines

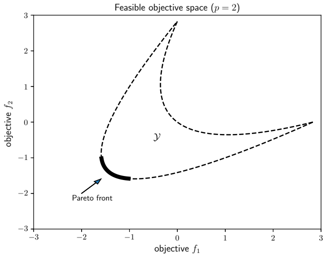
Multiobjective Optimization Problems

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

- $\mathcal{X} \subset \mathbb{R}^n$ is the feasible set
- $F(x) = (f_1(x), f_2(x), \ldots, f_o(x))$

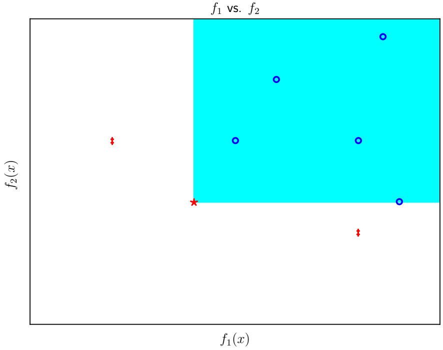


$$F : \mathcal{X} \to \mathcal{Y}$$

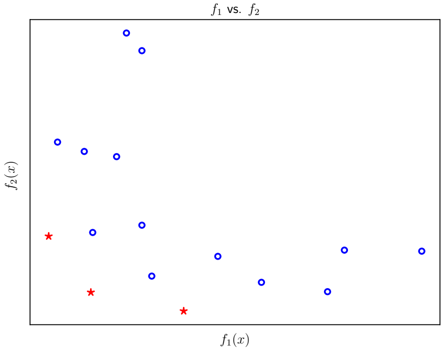


Dominance Relation

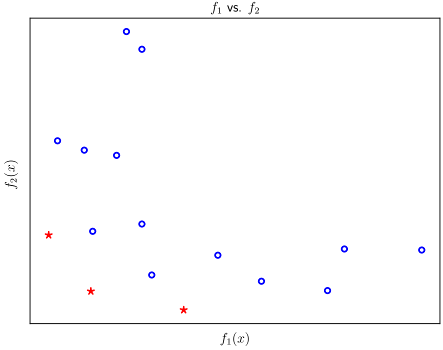
Dominance Relation

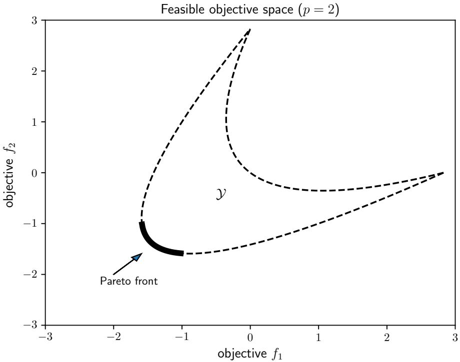


Dominance Relation



Dominance Relation



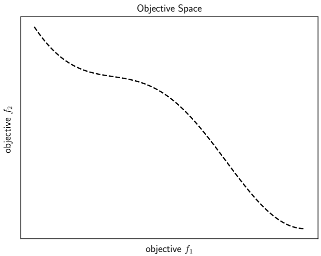


Scalarization

$$\min_{x \in \mathbb{R}^n} F(x) = (f_1(x), f_2(x), \ldots, f_o(x))$$

$$G : \mathbb{R}^o \to \mathbb{R}$$

$$\min_{x \in \mathbb{R}^n} (G \circ F)(x)$$

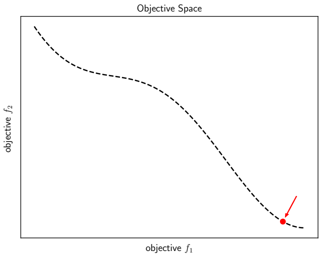


Scalarization

$$\min_{x \in \mathbb{R}^n} F(x) = (f_1(x), f_2(x), \ldots, f_o(x))$$

$$G : \mathbb{R}^o \to \mathbb{R}$$

$$\min_{x \in \mathbb{R}^n} (G \circ F)(x)$$

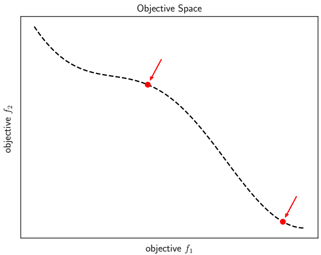


Scalarization

$$\min_{x \in \mathbb{R}^n} F(x) = (f_1(x), f_2(x), \ldots, f_o(x))$$

$$G : \mathbb{R}^o \to \mathbb{R}$$

$$\min_{x \in \mathbb{R}^n} (G \circ F)(x)$$

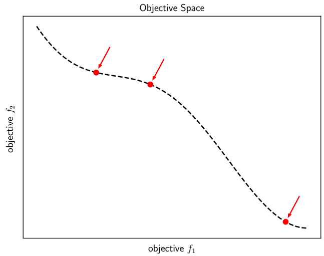


Scalarization

$$\min_{x \in \mathbb{R}^n} F(x) = (f_1(x), f_2(x), \ldots, f_o(x))$$

$$G : \mathbb{R}^o \to \mathbb{R}$$

$$\min_{x \in \mathbb{R}^n} (G \circ F)(x)$$



Summary of MOO Solvers

Scalarization + single-objective solver = multiobjective solver

Summary of MOO Solvers

Acquisition function

~~Scalarization~~ + single-objective solver = multiobjective solver

Summary of MOO Solvers

Acquisition function

~~Scalarization~~ + single-objective solver = multiobjective solver

Goal:

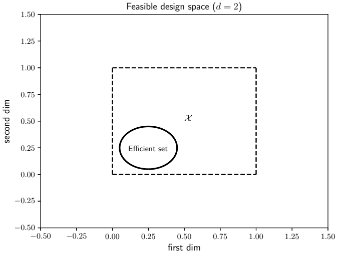
ParMOO a framework for developing, customizing, and deploying parallel multiobjective solvers for science/engineering applications

Multiobjective *Simulation* Optimization

Just “multiobjective solvers” is too broad!

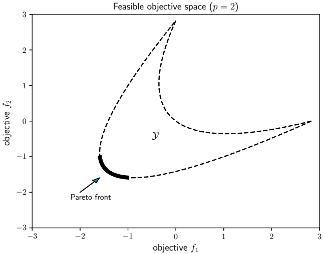
Multiobjective *Simulation* Optimization

Input variables



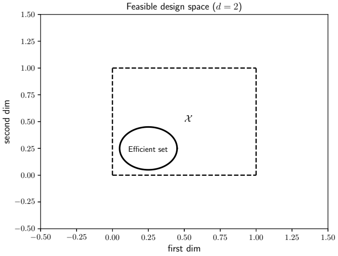
$$F : \mathcal{X} \to \mathcal{Y}$$

Objective space



Multiobjective *Simulation* Optimization

Input variables



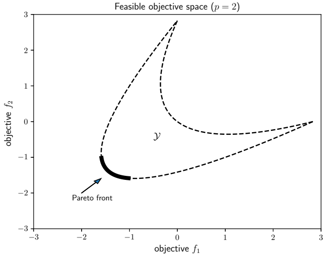
Blackbox process

Numerical simulation?
Real-world experiment?
Build a prototype?
Run a test?

$$\longrightarrow$$

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

Objective space



Existing Techniques

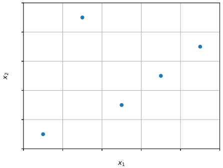
- Multiobjective Evolutionary/genetic algorithms
- Multidirectional search
- Multiobjective direct search
- Multiobjective Bayesian optimization

Existing Techniques

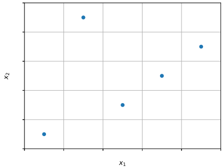
- Multiobjective Evolutionary/genetic algorithms
- Multidirectional search
- Multiobjective direct search
- Multiobjective Bayesian optimization
- **Multi response surface methodology (RSM)**

Multiobjective RSM

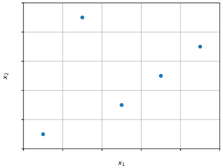
Multiobjective RSM

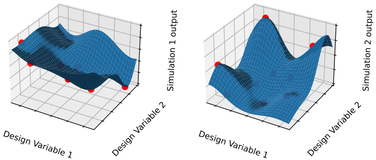


Multiobjective RSM

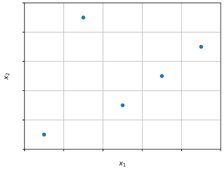


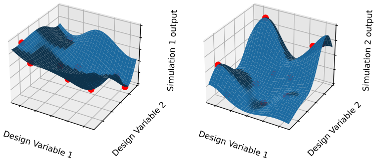
Multiobjective RSM

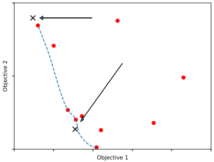




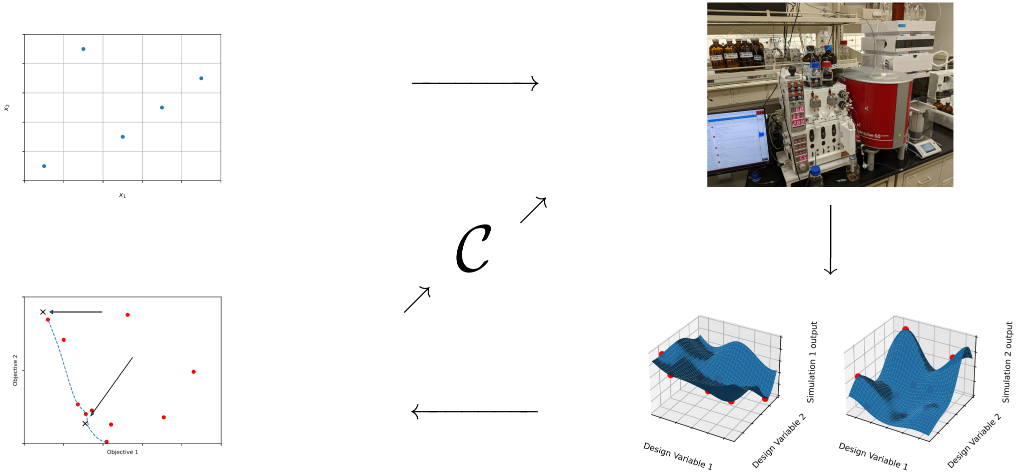
Multiobjective RSM







Multiobjective RSM



Existing Solvers, Libraries, and Frameworks

Name	Type	Language	Method	Consts	Var Types	Surrogates
BoostDFO	L	Matlab	MS	some	real	yes
BoTorch	F	Python	BO	yes	mixed	yes
Dragonfly	F/S	Python	BO	yes	mixed	yes
jMetal/jMetalPy	L/F	Java/Py	EA	yes	mixed	no
MODIR	S	Fortran	MS	no	real	no
BiMADS	S	C++	MS	yes	mixed	yes
ParEGO	S	C	EA/BO	no	real	yes
PlatEMO	L/F	Matlab	EA	some	mixed	some
Platypus	L	Python	EA	yes	mixed	no
pagmo/pygmo	F	C++/Py	EA	some	mixed	no
parmoo	F	Python	MS/BO	yes	mixed	yes
pymoo	L/F	Python	EA	some	mixed	no
PyMOSO	F	Python	MS	yes	int	no
SPEA2	S	C	EA	no	real	no
VTMOP	S	Fortran	MS	no	real	yes

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:

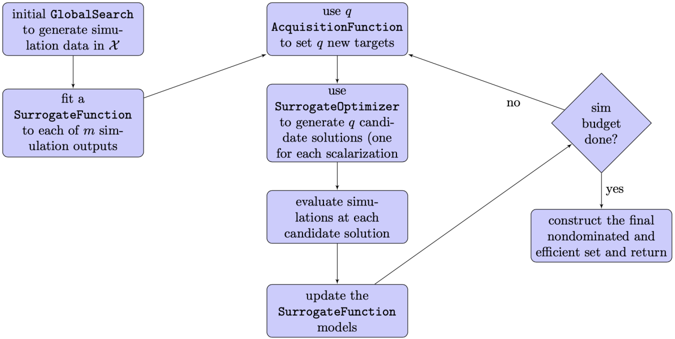
1. Highly customizable framework for multiobjective RSM
2. Exploit structure and domain knowledge simulation-based optimization problems
3. Flexible problem types (mixed-variables, constraints, etc.)

Design constraints:

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

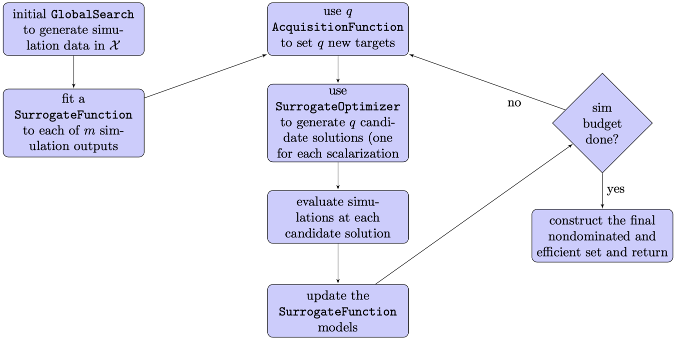
Customizability

ParMOO uses an object-oriented framework:



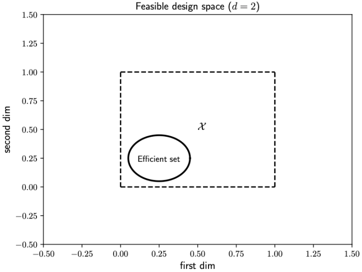
Customizability

ParMOO uses an object-oriented framework:



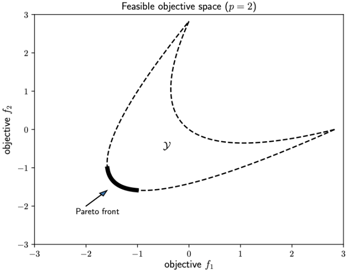
- Search/DOE
- Surrogate model
- Acquisition function
- Single-obj solver

Simulation Structure



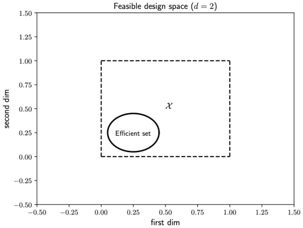
Design space

Objective Functions



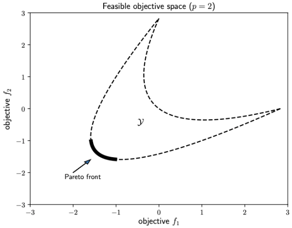
Objective space

Simulation Structure



Design space

$\xrightarrow{\text{Simulations}}$ $\mathcal{S}$ $\xrightarrow{\text{Objectives}}$



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**

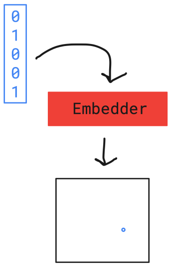
Flexible Problem Types

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

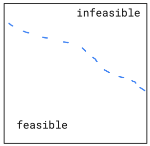
Flexible Problem Types

- 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

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. 2022. ParMOO: A Python library for parallel multiobjective simulation optimization. Under review with JOSS.

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:

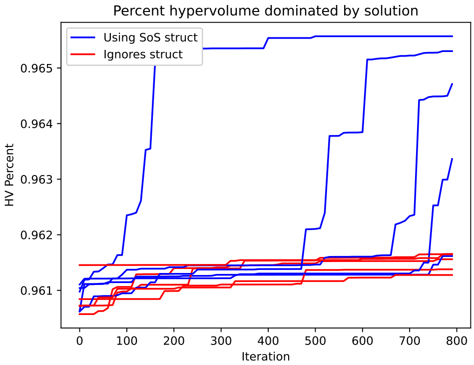
$$S_i(x) = M(\xi_i; x) - d_i, \qquad i = 1, \ldots, 198;$$

Min SOS across 3 observable classes $\qquad F_t = \sum_{i=1}^{m_t} \left(S_{t,i}(x)\right)^2$

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:

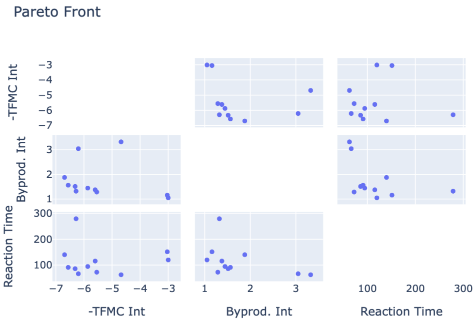
- Mixed variable types
- Heterogeneous objectives
- Must send experiments to run on CFR

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`

ParMOO is under review with JOSS

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.