Managing Computationally Expensive Blackbox Multiobjective Optimization Problems with libEnsemble

Tyler Chang[1], Jeffrey Larson[2], Layne Watson[1], & Thomas Lux[1]

[1] Dept. of Computer Science, Virginia Polytechnic Institute & State Univ.
[2] Mathematics and Computer Science Division, Argonne National Laboratory

May 19, 2020

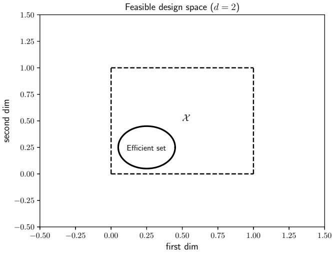
Table of Contents

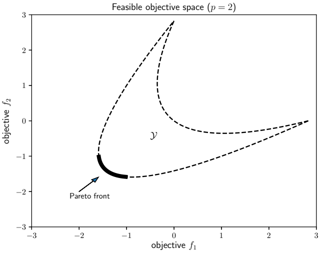
What is a MOP?

- The Multiobjective Optimization Problem (MOP) generalizes the Single Objective (Scalar) Optimization Problem (SOP);
- The MOP attempts to balance the tradeoff between multiple conflicting objectives;
- Whereas the SOP generally has a unique solution, the solution to a MOP is a *set* of *Pareto optimal* solutions;

The Objective Space and Pareto Front

- The solution to a MOP is a set of *nondominated* or *Pareto optimal* solutions;
- x^* is Pareto optimal if for all $x \in \mathcal{X}$, $F(x) \not\leq F(x^*)$;





Solving a MOP

Find a discrete set of approximately nondominated objective points that describes the Pareto front, and the corresponding efficient designs

Types of MOPs

functions are "cheap" to evaluate derivative info is available	functions are "cheap" to evaluate no derivative info is available
functions are costly to evaluate derivative info is available	functions are costly to evaluate no derivative info is available

Focus on bottom right: **expensive blackbox MOPs**!

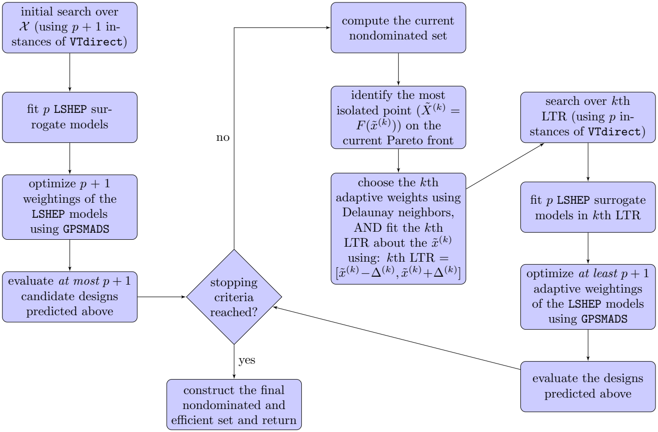
Algorithm Implemented

Provide a parallel implementation of the multiobjective optimization algorithm (MOA) proposed by

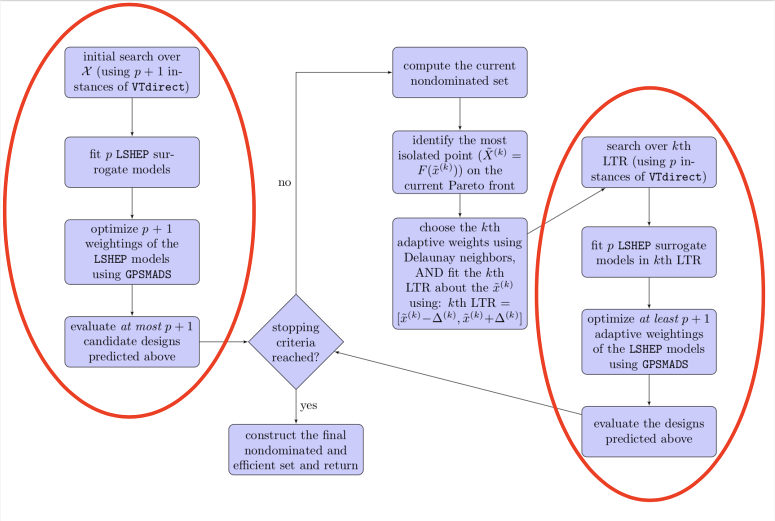
Shubhangi Deshpande, et al. "Multiobjective optimization using an adaptive weighting scheme." Optimization Methods and Software 31.1 (2016): 110-133.

Combines adaptive weighting scheme, response surface modeling, and trust region methods

The Algorithm Outline

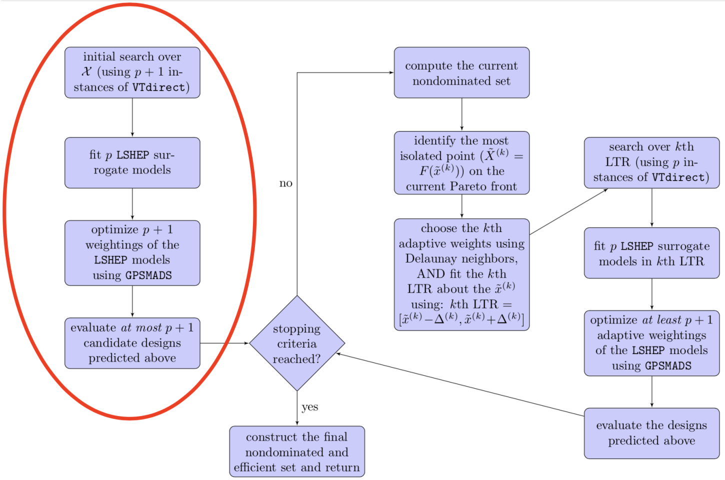


RSM phases

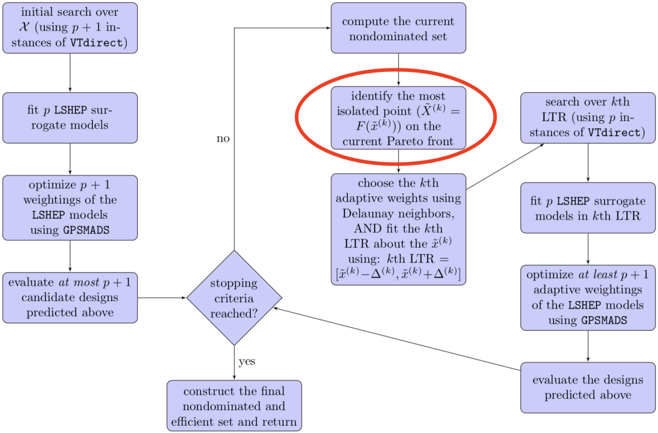


initial search over $\mathcal{X}$ (using $p+1$ instances of VTdirect)
fit p LSHEP surrogate models
optimize $p+1$ weightings of the LSHEP models using GPSMADS
evaluate *at most* $p+1$ candidate designs predicted above
stopping criteria reached?
no
yes
construct the final nondominated and efficient set and return
compute the current nondominated set
identify the most isolated point $(\tilde{X}^{(k)} = F(\tilde{x}^{(k)}))$ on the current Pareto front
choose the kth adaptive weights using Delaunay neighbors, AND fit the kth LTR about the $\tilde{x}^{(k)}$ using: kth LTR $= [\tilde{x}^{(k)} - \Delta^{(k)}, \tilde{x}^{(k)} + \Delta^{(k)}]$
search over kth LTR (using p instances of VTdirect)
fit p LSHEP surrogate models in kth LTR
optimize *at least* $p+1$ adaptive weightings of the LSHEP models using GPSMADS
evaluate the designs predicted above

0th iteration



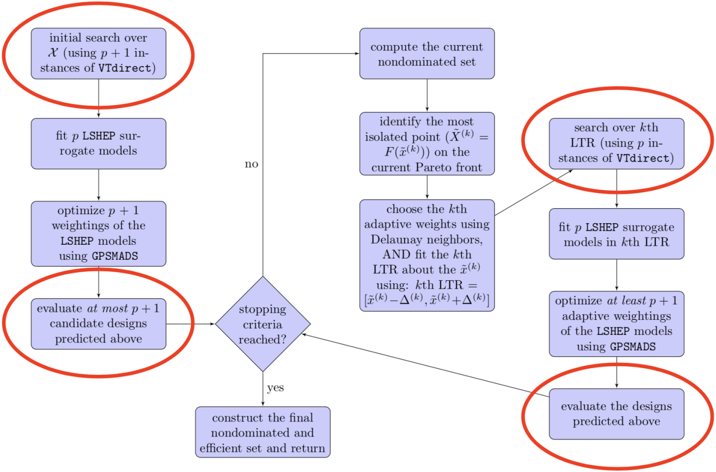
Key component



Sources of Parallelism

1. The function F (left to the user)
2. Iteration complexity (assumed to not offer much improvement)
3. Function evaluations $\leftarrow$ **most important**

Function evaluations



Parallelizing the original algorithm

- Recall that F is being distributed by user
- Use OpenMP *shared memory parallelism*, essentially for achieving asynchronous behavior
- Puts burden of distribution on user
- Better flexibility for real-world HPC systems

Parallelizing the original algorithm

- Recall that F is being distributed by user
- Use OpenMP *shared memory parallelism*, essentially for achieving asynchronous behavior
- Puts burden of distribution on user
- Better flexibility for real-world HPC systems

To parallelize function evaluations:

- Batch of candidates can be trivially evaluated in parallel
- Instead of using `pVTdirect` from VTDIRECT95 (which uses MPI) made a modification to the serial code called `bVTdirect` that evaluates batches of points in parallel using OpenMP
- For maximal parallelism, replace `bVTdirect` with a Latin hypercube design, which can be evaluated in parallel

libEnsemble

The `libEnsemble` library is part of the Exascale Computing Project at Argonne to harness increased levels of concurrency when distributing large simulations over extreme scale resources

- Generator function (generates simulations to run)
- Simulator function (runs simulations, possibly in parallel)
- Allocator function (decides whether to generate or simulate)

libEnsemble

The `libEnsemble` library is part of the Exascale Computing Project at Argonne to harness increased levels of concurrency when distributing large simulations over extreme scale resources

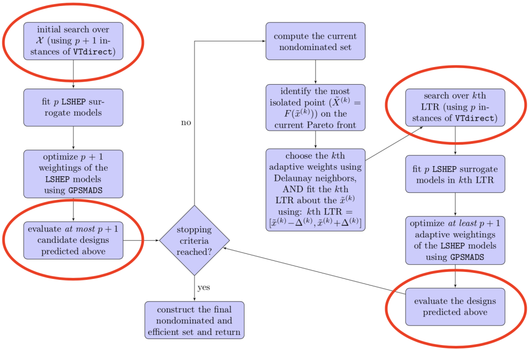
- Generator function (generates simulations to run)
- Simulator function (runs simulations, possibly in parallel)
- Allocator function (decides whether to generate or simulate)

`VTMOP` is the *generator* for `libEnsemble`

- Each call to the *generator* runs a half-iteration and requests a design exploration (using Latin hypercube) or a batch of candidate evaluations
- The *simulator* evaluates all the requested designs
- The *allocator* waits until all designs are evaluated and then swithes back to the *generator*

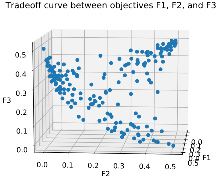
Integrating with libEnsemble

- Want nice big batches that match available resources
- Use a Latin hypercube search during the search phase
- Pad out batches of candidates using additional weights

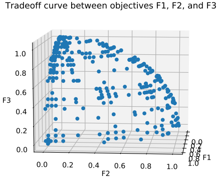


Test functions

- $F^{(c)}(x) = (\|x - 0.5e^{(1)}\|_2^2, \ldots, \|x - 0.5e^{(p)}\|_2^2)$
- Convex Pareto front $\Rightarrow$ "easier" problem



- `DTLZ2` from Deb et al.
- Concave Pareto front $\Rightarrow$ "harder" problem



Evaluating a multiobjective optimization solution is a multiobjective problem...

We want:

1. Many points on the Pareto front (measured by *cardinality* of the solution set)
2. Good convergence of the solution points to the true Pareto front (measured by RMSE)
3. Even spacing/good coverage of the solution set (measured by Delaunay discrepancy, computed with `ScipPy`)

Average (5 runs) number of solutions, RMSE, and Delaunay discrepancy for $F^{(c)}$ and DTLZ2 with $d = 5$ for VTMOP using `bVTdirect` (DIR), Latin hypercube search (LH), and `libEnsemble` (LIBE)

Prob/Meth	$p = 2$	$p = 3$	$p = 4$
F^c / bVTdir	73, .00100, .207	173, .0505, .579	288, .101, NA
F^c / libE1	38, .0115, .180	93, .0665, .512	171, .117, .689
F^c / libE2	78, .0127, .158	189, .0560, .429	283, .104, .551
DTLZ2 / bVTdir	139, .00713, .109	354, .0401, .230	658, .0443, NA
DTLZ2 / libE1	80, .0993, .139	264, .167, .458	510, .210, .757
DTLZ2 / libE2	66, .103, .201	258, .175, .691	548, .201, .793

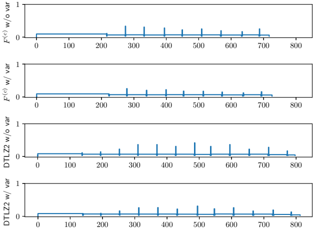
NA = Missing values due to `SciPy`'s Delaunay triangulation tool

Runtime performance (2000 evaluations)

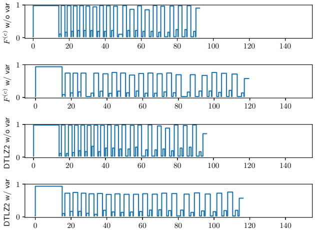
CPU time / wall time in seconds for DIR, LH, and LIBE versions with 36 cores & 1 sec (+ var)/1 core per eval

p	Meth	F_c no var	F_c + var	DTLZ2 no var	DTLZ2 + var
	`bVTdir1`	2008 / 1037	2007 / 1039	2007 / 1093	2004 / 1082
	`bVTdir2`	NA / 170	NA / 239	NA / 175	NA / 240
2	`libE1`	2015 / 89	2017 / 107	2028 / 89	2011 / 109
	`libE2`	2051 / 112	2070 / 142	2060 / 111	2064 / 143
	`bVTdir1`	2012 / 717	2012 / 719	2021 / 797	2018 / 797
	`bVTdir2`	NA / 137	NA / 207	NA / 165	NA / 237
3	`libE1`	2023 / 94	2034 / 116	2040 / 95	2023 / 117
	`libE2`	2077 / 133	2066 / 144	2054 / 99	2057 / 126
	`bVTdir1`	2026 / 582	2029 / 586	2177 / 807	2149 / 782
	`bVTdir2`	NA / 134	NA / 208	NA / 280	NA / 348
4	`libE1`	2042 / 108	2045 / 127	2176 / 200	2201 / 280
	`libE2`	2134 / 190	2124 / 186	2182 / 227	2185 / 257

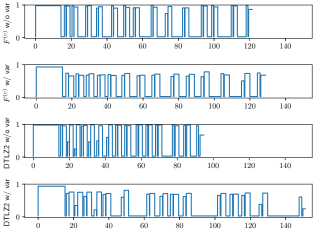
CPU Usage (36 cores)



bVTdir1 (not suitable for 36 cores)



libE1



libE2

Questions?

Background in MOPs

The Algorithm

Parallel implementations

Results