Diversity-Guided Multi-Objective Bayesian Optimization With Batch Evaluations -Supplementary-

Mina Konaković Luković[*]
MIT CSAIL
minakl@mit.edu

Yunsheng Tian[*]
MIT CSAIL
yunsheng@csail.mit.edu

Wojciech Matusik
MIT CSAIL
wojciech@csail.mit.edu

A Gaussian process derivatives

To better explore the Pareto front based on the surrogate model, in our case Gaussian process (GP), our Pareto front approximation algorithm (see Section 4.2 of the paper) makes use of the Jacobian and Hessian of the GP prediction $\mu(\mathbf{x}), \Sigma(\mathbf{x})$ w.r.t. the input $\mathbf{x}$:

$$\frac{\partial \mu}{\partial \mathbf{x}} = \frac{\partial m}{\partial \mathbf{x}} + \frac{\partial \boldsymbol{k}}{\partial \mathbf{x}} \boldsymbol{K}^{-1} Y$$
$$\frac{\partial \mu}{\partial^2 \mathbf{x}} = \frac{\partial m}{\partial^2 \mathbf{x}} + \frac{\partial \boldsymbol{k}}{\partial^2 \mathbf{x}} \boldsymbol{K}^{-1} Y$$
$$\frac{\partial \Sigma}{\partial \mathbf{x}} = -\frac{\partial \boldsymbol{k}}{\partial \mathbf{x}} \boldsymbol{K}^{-1} \boldsymbol{k}^T - \boldsymbol{k} \boldsymbol{K}^{-1} \frac{\partial \boldsymbol{k}^T}{\partial \mathbf{x}}$$
$$\frac{\partial \Sigma}{\partial^2 \mathbf{x}} = -\frac{\partial \boldsymbol{k}}{\partial^2 \mathbf{x}} \boldsymbol{K}^{-1} \boldsymbol{k}^T - 2\frac{\partial \boldsymbol{k}}{\partial \mathbf{x}} \boldsymbol{K}^{-1} \frac{\partial \boldsymbol{k}^T}{\partial \mathbf{x}} - \boldsymbol{k} \boldsymbol{K}^{-1} \frac{\partial \boldsymbol{k}^T}{\partial^2 \mathbf{x}}$$

where $\boldsymbol{k} = k(\mathbf{x}, X)$, $\boldsymbol{K} = k(X, X)$, $\frac{\partial m}{\partial \mathbf{x}} = 0$ and $\frac{\partial m}{\partial^2 \mathbf{x}} = 0$ since we use $m(\mathbf{x}) = 0$.

In this paper we use Matern 5/2 kernel [16] as default, defined as:

$$\boldsymbol{k} = \sigma_f^2 (1 + \sqrt{5}d + \frac{5}{3}d^2) \exp(-\sqrt{5}d) + \sigma_n^2$$

where $\sigma_f \in \mathbb{R}$ is a scaling parameter and $\sigma_n \in \mathbb{R}$ is a constant parameter, d denotes Euclidean distance $d(\mathbf{x}/l, X/l)$, and $l \in \mathbb{R}^d$ is the length-scale parameter of the kernel. The derivation of $\frac{\partial \boldsymbol{k}}{\partial \mathbf{x}}$ and $\frac{\partial \boldsymbol{k}}{\partial^2 \mathbf{x}}$ on this kernel is as follows:

$$\frac{\partial \boldsymbol{k}}{\partial \mathbf{x}} = -\frac{5}{3}\sigma_f^2 \exp(-\sqrt{5}d)(1 + \sqrt{5}d) d \frac{\partial d}{\partial \mathbf{x}}$$
$$\frac{\partial \boldsymbol{k}}{\partial^2 \mathbf{x}} = -\frac{5}{3}\sigma_f^2 \exp(-\sqrt{5}d)(-5d^2 (\frac{\partial d}{\partial \mathbf{x}})^2 + (1 + \sqrt{5}d)((\frac{\partial d}{\partial \mathbf{x}})^2 + d\frac{\partial d}{\partial^2 \mathbf{x}}))$$

We omit the derivation for derivatives of Euclidean distance d since it is well known. Extending the derivation to other kernels such as Matern 1/2, Matern 3/2 and RBF [16] kernels is straightforward.

B Experimental setup details

Our algorithm does not rely on any specific computing infrastructure. For hardware, a modern CPU will suffice for running the experiments. See more details about the recommended software setup in the README of our code.

[*]Equal contribution.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

B.1 Problem description

In this section, we briefly introduce the properties of each problem, including the dimensions of the design space $\mathcal{X} \subset \mathbb{R}^d$ and performance space $\mathbf{f}(\mathcal{X}) \subset \mathbb{R}^m$, and the reference points we use for calculating the hypervolume indicator. The problem descriptions for 13 synthetic functions and 7 real-world problems are shown in Table 1 and Table 2 respectively. For all functions that can work with arbitrary dimensions, we use 6 variables and 2 objectives for consistency in testing. We perform 10 independent test runs with 10 different random seeds for each problem on each algorithm. For each test run of one problem, we use the same initial set of samples for every algorithm, which is generated by Latin hypercube sampling [13] using a same random seed. To have a fair comparison, we simply set the reference point $\mathbf{r} \in \mathbb{R}^m$ as a vector containing the maximum value of each objective over the initial set of samples $\{\mathbf{x}_1, ..., \mathbf{x}_k\}$:

$$\mathbf{r} = (\max_{1 \leq i \leq k} f_1(\mathbf{x}_i), ..., \max_{1 \leq i \leq k} f_m(\mathbf{x}_i)).$$

Table 1: Description of synthetic functions.

Name	d	m	$\mathbf{r}$
ZDT1	6	2	(0.9902, 6.3936)
ZDT2	6	2	(0.9902, 7.7158)
ZDT3	6	2	(0.9902, 6.5464)
DTLZ1	6	2	(360.7570, 343.4563)
DTLZ2	6	2	(1.7435, 1,6819)
DTLZ3	6	2	(706.5260, 746.2411)
DTLZ4	6	2	(1.8111, 0.7776)
DTLZ5	6	2	(1.7435, 1.6819)
DTLZ6	6	2	(5.7482, 5.6523)
OKA1	2	2	(7.4051, 4.3608)
OKA2	3	2	(3.1315, 4.6327)
VLMOP2	6	2	(1.0, 1.0)
VLMOP3	2	3	(8.1956, 53.2348, 0.1963)

Table 2: Description of real-world problems.

Name	Description	d	m	$\mathbf{r}$
RE1	Four bar truss design [3]	4	2	(2967.0243, 0.0383)
RE2	Reinforced concrete beam design [1]	3	2	(703.6860, 899.2291)
RE3	Hatch cover design [1]	2	2	(5885.4870, 5.5063)
RE4	Welded beam design [17]	4	3	(202.8569, 42.0653, 2111643.6209)
RE5	Disc brake design [17]	4	3	(6.1356, 6.3421, 12.9737)
RE6	Gear train design [8]	4	3	(6.6764, 59.0, 0.4633)
RE7	Rocket injector design [9]	4	3	(0.8136, 0.8889, 0.9799)

B.2 Hyperparameters

For fair comparison among different algorithms, we try to use the same set of common hyperparameters as much as possible. The common hyperparameters are presented in Appendix B.2.1 and the algorithm-specific hyperparameters are presented in Appendix B.2.2. Here we list the important hyperparameters used in the implementation, and more details could be found in the code.

B.2.1 General hyperparameters

Surrogate model We use the same Gaussian process model as a surrogate for all experiments. We use zero mean function and anisotropic Matern 5/2 kernel. The corresponding hyperparameters are specified in Table 3, which are suggested by TSEMO.

Table 3: GP hyperparameters.

parameter name	value
initial l	$(1, ..., 1) \in \mathbb{R}^d$
l range	$(\sqrt{10^{-3}}, \sqrt{10^3})$
initial σ_f	1
σ_f range	$(\sqrt{10^{-3}}, \sqrt{10^3})$
initial σ_n	10^{-2}
σ_n range	$(e^{-6}, 1)$

Multi-objective evolutionary algorithm MOEA/D-EGO, TSEMO, USeMO-EI share the same NSGA-II solver using simulated binary crossover [5] and polynomial mutation [6] for finding the Pareto front of acquisition functions. The initial population is obtained from the best current samples determined by non-dominated sort [7]. The other hyperparameters are specified in Table 4.

Table 4: NSGA-II hyperparameters.

parameter name	value
population size	100
number of generations	200
crossover η_c	15
mutation η_m	20

We also use the baseline algorithm NSGA-II to compare with other MOBO algorithms. For that, we use the same hyperparameters for crossover η_c and mutation η_m as stated in Table 4, but the population size is set to the batch size and the number of generations is equivalent to the number of algorithm iterations.

B.2.2 Algorithm-specific hyperparameters

ParEGO Since ParEGO is a single-point MOBO method, we extend it to the batch setting by using b random scalarization weights in each iteration, where b is the batch size. We use Chebyshev scalarization [14] and CMA-ES algorithm [10] for solving the scalarized single-objective problem with $\sigma = 0.5$ as initial standard deviation.

MOEA/D-EGO We use a similar implementation as the original MOEA/D-EGO, except that we remove the FuzzyCM. Nowadays it is already computationally efficient to train the Gaussian process model and directly use it for prediction, rather than relying on relatively faster but less accurate approximation methods. Hence, the performance of our implementation could potentially be relatively higher than the original implementation thanks to the accuracy gain of removing the FuzzyCM. We use simulated binary crossover and polynomial mutation for MOEA/D, and the other hyperparameters used are described in Table 5.

Table 5: MOEA/D hyperparameters.

parameter name	value
number of reference directions	100
number of generations	200
number of neighbors	20
neighbor mating probability	0.9
crossover η_c	20
mutation η_m	20

TSEMO For TSEMO, we use mostly the same set of hyperparameters used in its original implementation. For spectral sampling we use 100 points.

USeMO-EI Since USeMO-EI is a single-point MOBO method, we extend it to the batch setting by selecting top-b points with maximal uncertainty, where b is the batch size.

DGEMO Most of the distinct hyperparameters of DGEMO come from the multi-objective optimization algorithm [18], as presented in Table 6. In this MOO algorithm, the initial population is also obtained from the best current samples determined by non-dominated sort. Generally DGEMO is robust to the hyperparameter choices and in practice mildly changing the values of these hyperparameters wouldn't affect the final performance of DGEMO too much. For potentially slightly better performance, the number of buffer cells and the number of grid samples on local manifold could be increased for obtaining more candidate solutions to select, at the cost of spending more computation time. The label cost for graph cut is a key hyperparameter that controls the number of diversity regions, which could be adjusted case by case. Higher cost leads to less diversity regions. Even though we use the same label cost across all the benchmark problems, which is obviously sub-optimal, DGEMO still outperforms other baseline algorithms. More suggestions on hyperparameter choices could be found in the original paper of this algorithm[18].

Table 6: DGEMO hyperparameters.

parameter name	value
number of buffer cells	100 for 2-dim, 1000 for higher-dim
max number of samples in each cell	10
buffer origin	$(0, ..., 0) \in \mathbb{R}^m$
δ_b	0.2
δ_p	10
δ_s	0.3
label cost for graph cut	10
number of grid samples on local manifold	100

C Computational complexity

C.1 DEGMO algorithm complexity

Since our implementation is based on several advanced open-source Python packages doing the mathematical calculation (such as NumPy [15] and SciPy [20]) and highly memory-efficient, there is very a small memory occupation when running DGEMO algorithm. Here we mainly analyze the time complexity of our algorithm from three stages.

Surrogate model fitting It is well known that Gaussian process fitting is of $O(N^3)$ time complexity, where N is the size of the current dataset. However, in our case, N is usually not a big number (up to several hundred) since we are focusing on a problem setting with relatively few data available, and that is the common scenario in applications of Bayesian optimization.

Multi-objective optimization In the algorithm of [18], at each generation, for each individual in the population, the computationally expensive operations are the following four components: applying L-BFGS [12] for local optimization, SLSQP [11] for solving KKT dual variables, computing exploration directions by taking the null space of a matrix (see Equation 3 of [18]), and GP prediction on neighboring grid samples. Specifically, L-BFGS operates at $O(Md)$ time complexity, where M is a relatively small number of algorithm updates and d is the dimension of design variables; SLSQP has $O(d^3)$ time complexity; computing the null space is roughly within $O(d^3)$ time; for GP prediction our algorithm only cares about the mean value without the standard deviation, hence it takes $O(Nn)$ time to complete, where n is the number of grid samples. After all generations are calculated, the graph-cuts [2] is performed to reach a continuous Pareto front representation. Representing the number of generation and population sizes as N_{gen} and N_{pop} respectively, the graph-cuts algorithm has $O(N_{gen}N_{pop})$ time complexity if we consider the number of algorithm iterations as a small constant, where $N_{gen}N_{pop}$ equals the maximum number of labels in the graph. As a result, the overall time complexity in this stage is roughly $O(N_{gen}N_{pop}(d^3 + Nn))$.

Table 7: Algorithm runtime comparison (seconds per iteration).

Problem	NSGA-II	ParEGO	MOEA/D-EGO	TSEMO	USeMO-EI	DGEMO (Ours)
ZDT1	0.01	16.39	25.39	1.90	1.82	5.51
ZDT2	0.01	13.34	25.09	1.89	1.80	5.24
ZDT3	0.01	15.40	25.24	1.73	1.84	5.45
DTLZ1	0.01	11.94	25.39	1.82	2.26	6.53
DTLZ2	0.01	14.05	25.14	1.50	1.78	4.76
DTLZ3	0.01	11.85	25.34	1.48	1.88	6.84
DTLZ4	0.01	13.76	25.46	1.65	1.93	7.33
DTLZ5	0.01	13.79	25.37	1.51	1.78	4.72
DTLZ6	0.01	10.57	24.75	1.62	1.56	5.69
OKA1	0.01	6.10	24.35	1.52	1.50	4.97
OKA2	0.01	9.28	24.59	1.52	1.60	5.78
VLMOP2	0.01	9.08	25.07	1.47	1.78	5.11
VLMOP3	0.01	7.10	29.24	2.18	1.90	6.38
RE1	0.01	14.37	24.85	1.99	1.66	4.60
RE2	0.01	7.03	24.66	1.61	1.65	4.44
RE3	0.01	7.90	24.56	1.65	1.65	4.77
RE4	0.01	12.60	28.79	2.02	2.25	8.12
RE5	0.01	17.92	28.91	2.89	1.99	7.00
RE6	0.01	6.74	28.89	1.81	1.95	7.03
RE7	0.01	17.91	28.86	4.82	1.98	6.66

Batch selection algorithm Given the batch size b, the maximum *performance buffer* capacity D (a number of cells times a maximum number of samples in each cell), the number of points on the current Pareto front N_{pf}, and the number of objectives m, our batch selection algorithm has time complexity $O(bDN_{pf})$ for $m = 2$, $O(bDN_{pf} \log N_{pf})$ for $m = 3$, and $O(bDN_{pf}^{\lfloor (m-1)/2 \rfloor + 1})$ for $m > 3$ due to the algorithm complexity difference for hypervolume calculation w.r.t. m.

C.2 Algorithm runtime comparison

To empirically investigate the efficiency of different algorithms, we record the runtime statistics in seconds as shown in Table 7. This comparison is done with batch size as 10 and 20 algorithm iterations, and the statistics are averaged across 6 different random seeds and all the iterations. Note that this comparison is highly implementation-dependent and is done by using our codebase. We run this comparison on an Intel Xeon(R) W-2195 CPU @ 2.30GHz * 36 processor. For ParEGO and DGEMO they can be easily parallelized, hence we record the runtime of experiments using 36 parallel processes. For NSGA-II, MOEA/D-EGO, TSEMO, and USeMO-EI, the serial runtime is recorded. From this table, we can see that NSGA-II is the most time-efficient algorithm since it operates simply without Bayesian optimization. For all the problems we test they have an analytical form of evaluation function thus the evaluation could be done extremely fast. However, in practice, many real-world problems require physical experiments for evaluation which could often take hours to days to finish. Hence for these real-world problems, the runtime cost of the algorithm would be negligible (commonly less than the 30s per iteration).

D Other comparison metrics

In this section, we compare the performance of DGEMO and other baseline algorithms using *generational distance (GD)* [19] (Figure 1) and *inverted generational distance (IGD)* [4] (Figure 2) as metrics, which are also very popular metrics in MOO literature besides hypervolume. These metrics measure the Euclidean distance between the Pareto front approximation found by the algorithm and the true Pareto front in different ways, thus the lower the better. But since most our benchmark problems do not have an analytical solution of the true Pareto front, in practice we approximate the true Pareto front by merging all the Pareto front approximations found by all algorithms using all random seeds. The results show that DGEMO performs the best across almost all the benchmark problems under either comparison metric.

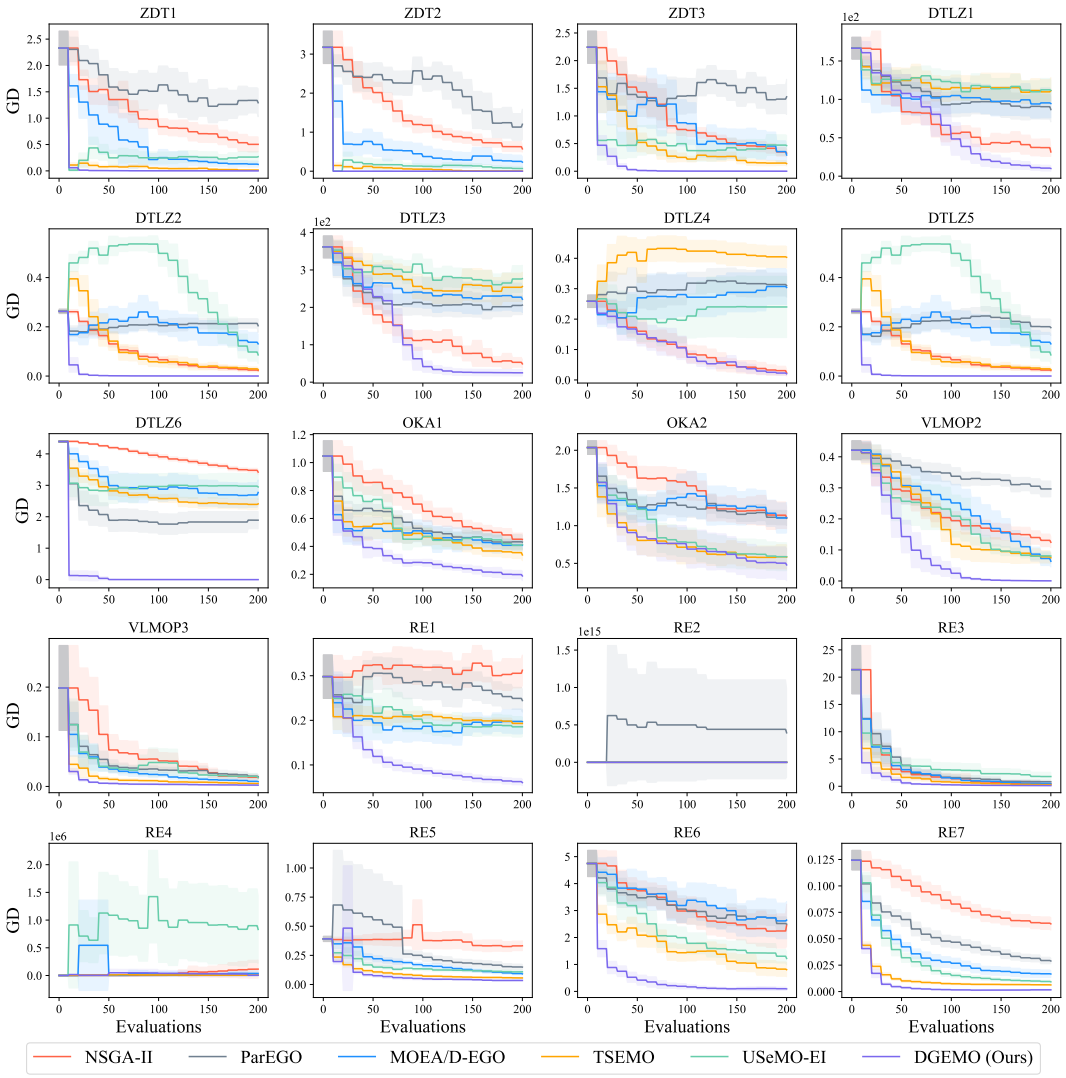

Figure 1: Generational distance of different algorithms on all problems with batch size as 10, shown with respect to the different number of function evaluations.

E Ablation studies

E.1 Batch size

We conduct experiments using different batch sizes, including 1 (Figure 3), 2 (Figure 4), 4 (Figure 5), 5 (Figure 6), and 20 (Figure 7). The result shows our DGEMO still consistently outperforms other algorithms and has a robust performance over a wide range of batch size.

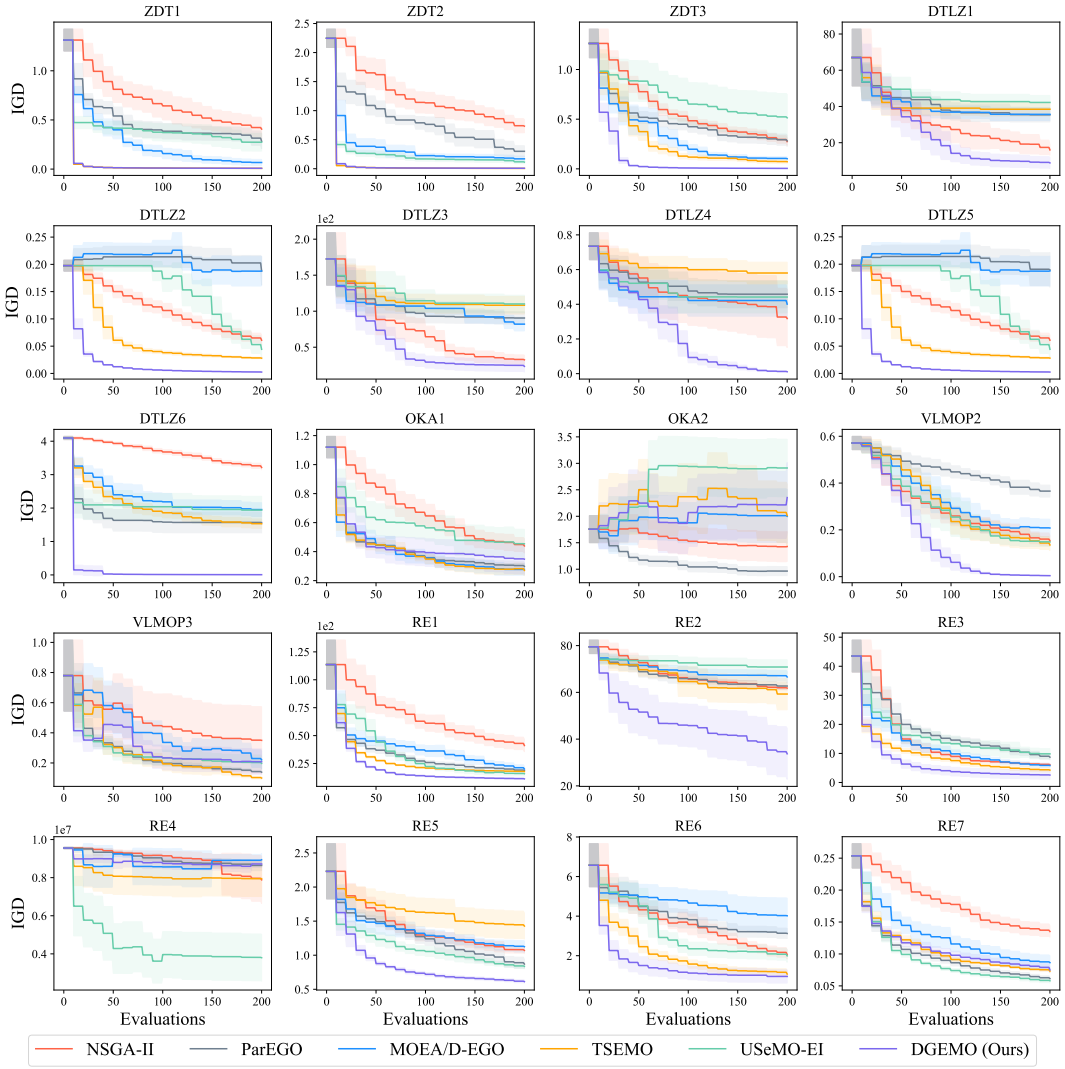

Figure 2: Inverted generational distance of different algorithms on all problems with batch size as 10, shown with respect to the different number of function evaluations.

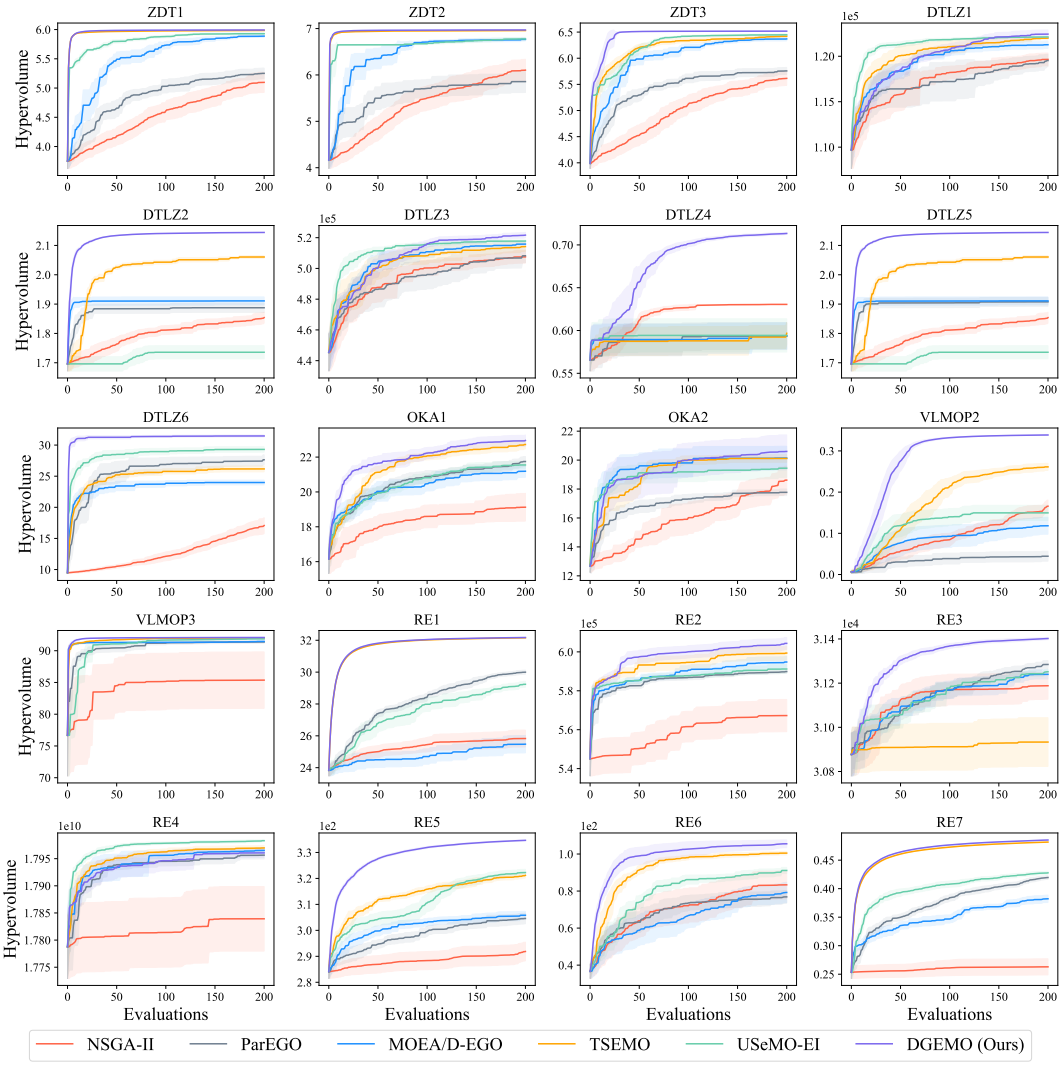

Figure 3: Hypervolume indicator of different algorithms on all problems with batch size as 1, shown with respect to the different number of function evaluations.

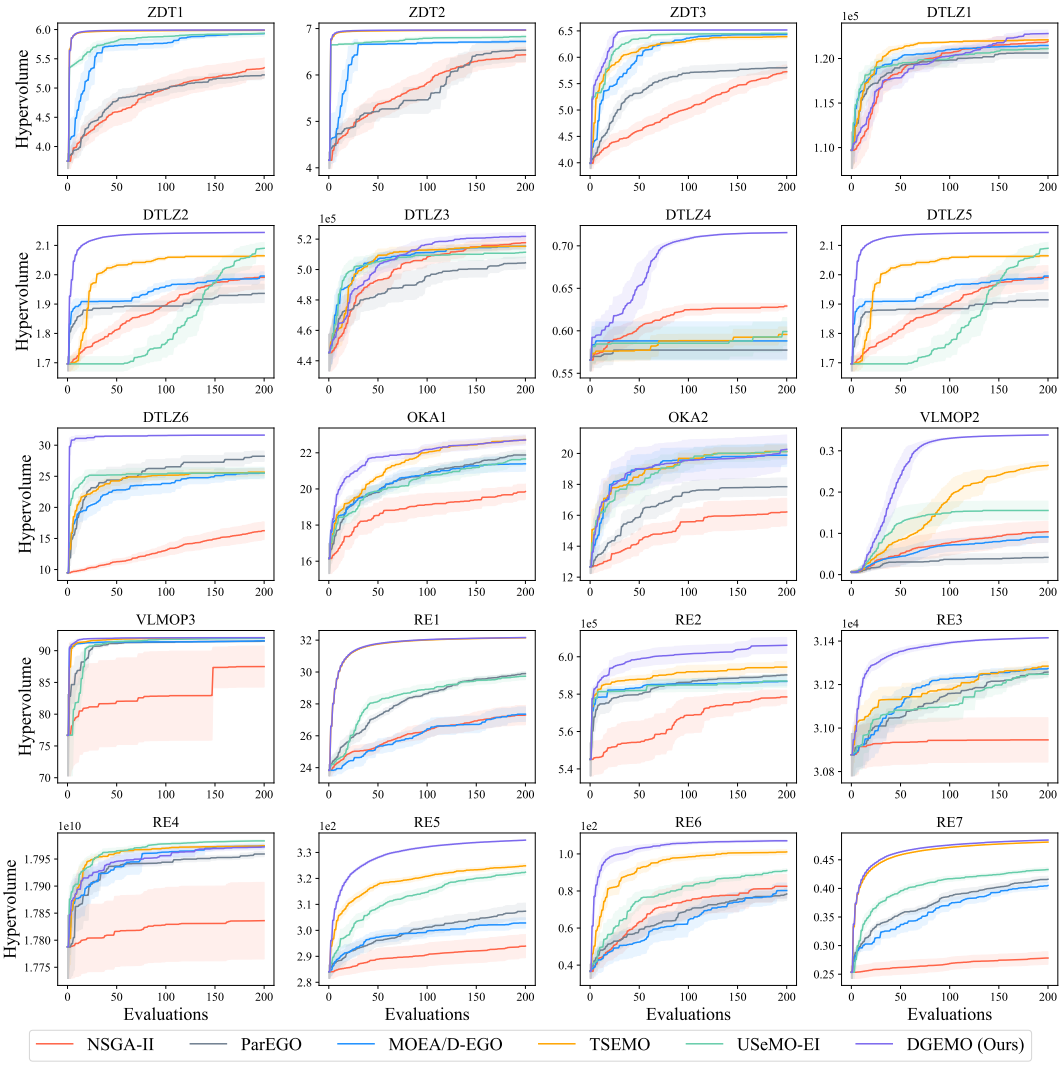

Figure 4: Hypervolume indicator of different algorithms on all problems with batch size as 2, shown with respect to the different number of function evaluations.

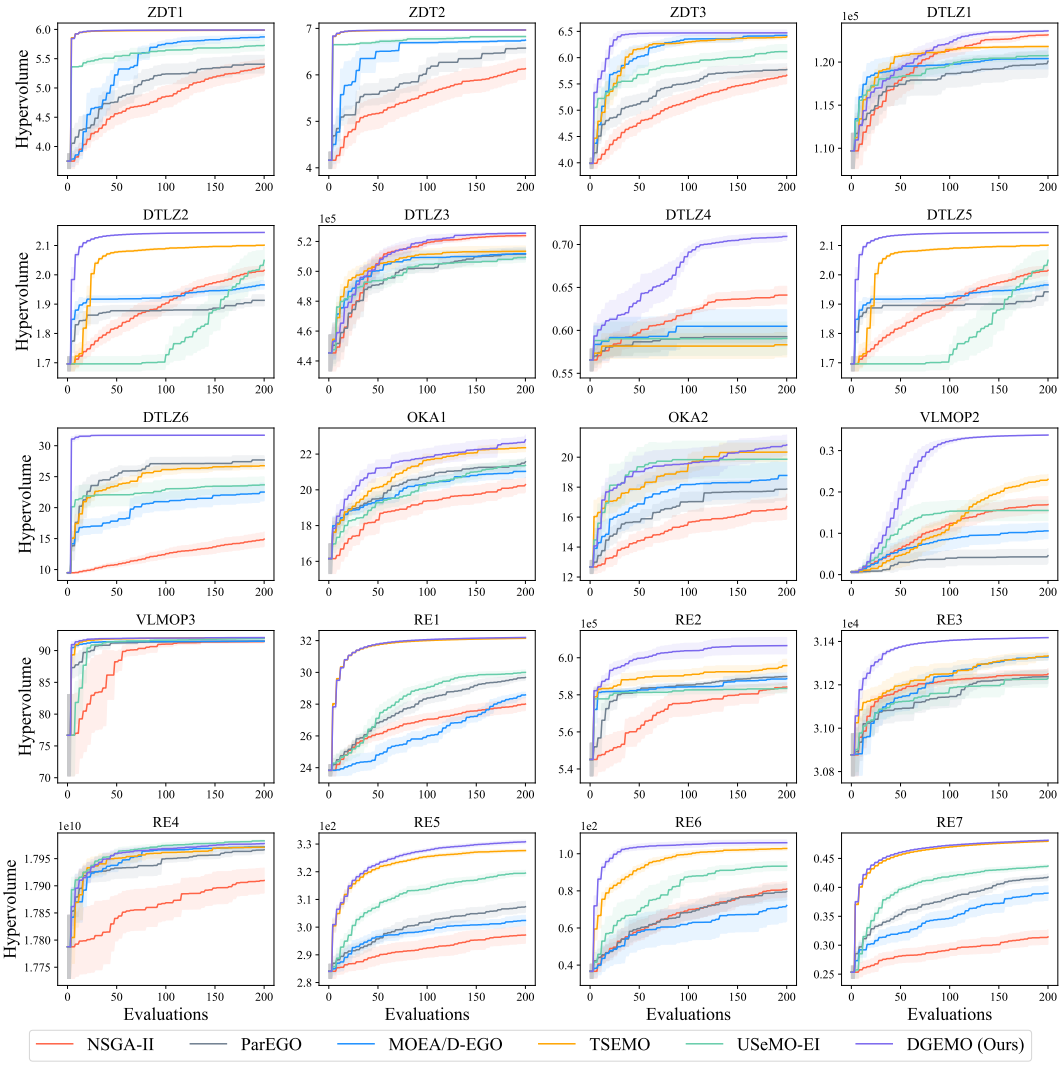

Figure 5: Hypervolume indicator of different algorithms on all problems with batch size as 4, shown with respect to the different number of function evaluations.

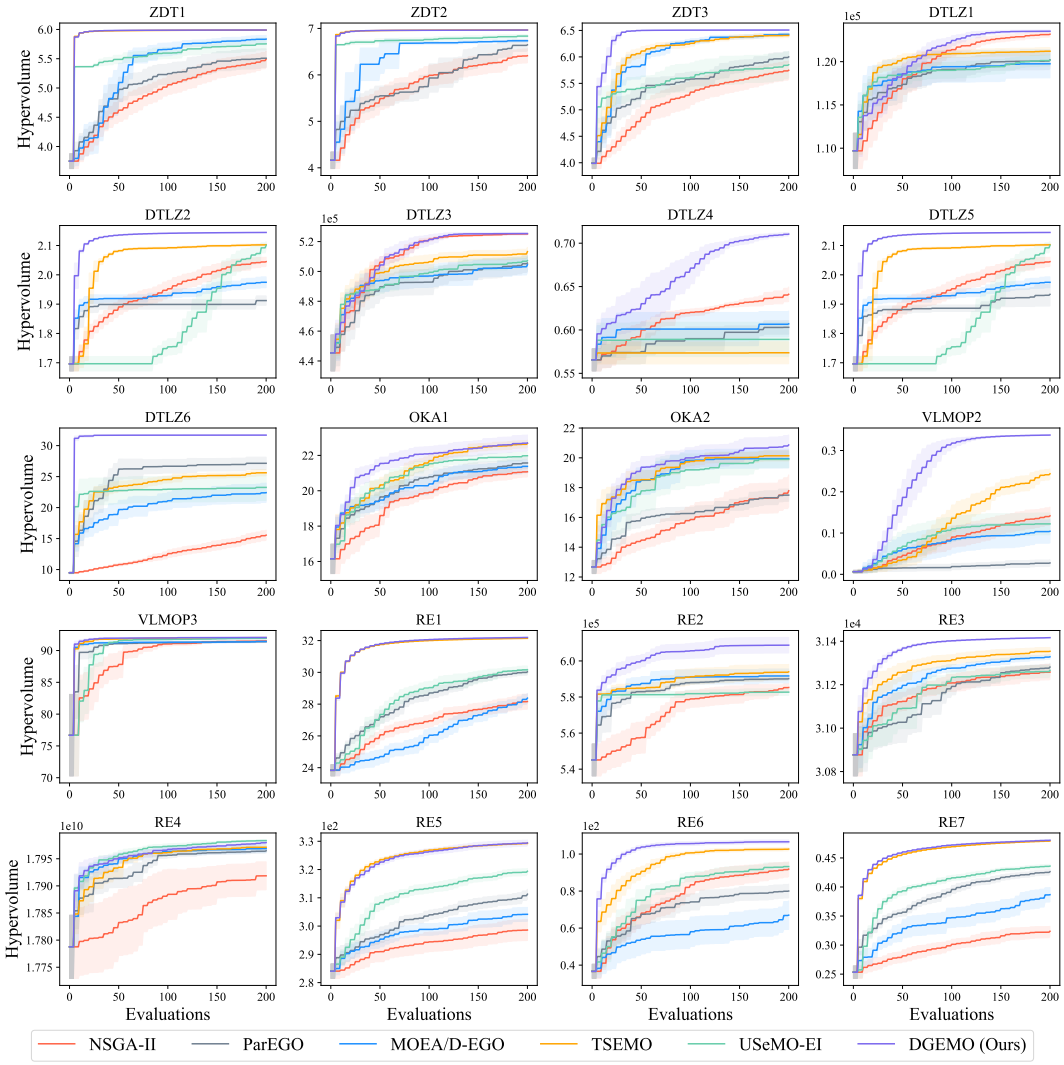

Figure 6: Hypervolume indicator of different algorithms on all problems with batch size as 5, shown with respect to the different number of function evaluations.

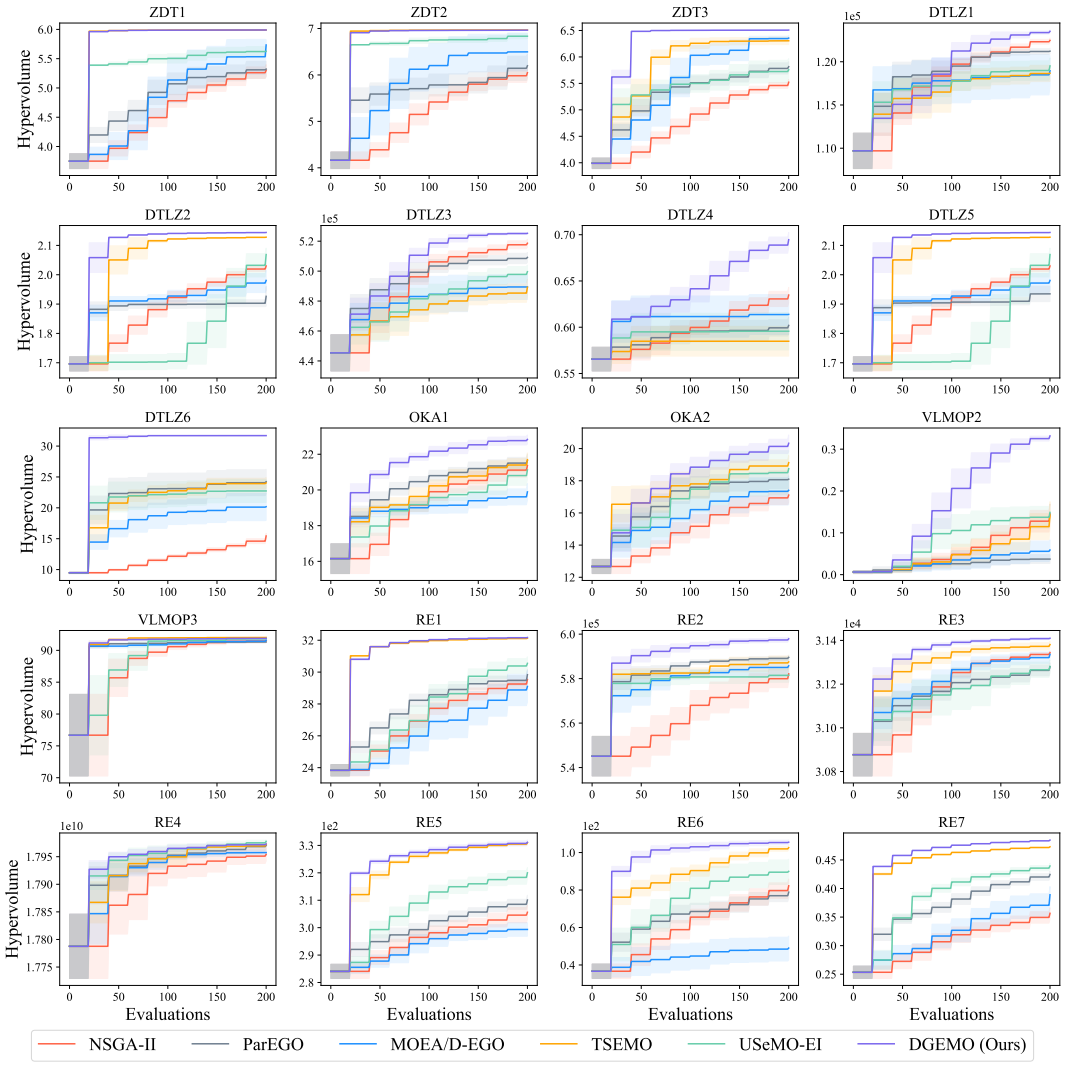

Figure 7: Hypervolume indicator of different algorithms on all problems with batch size as 20, shown with respect to the different number of function evaluations.

E.2 Pareto front approximation

To investigate whether the piecewise continuous Pareto front approximation obtained from Section 4.2 of the main paper helps improve the performance, we do another set of ablation experiments. We remove the last part of the Pareto front approximation algorithm with KKT conditions, and only get the solutions obtained after the local optimization step (see Section 4.2 of main paper). From these solutions we select the batch of samples to evaluate according to the maximal hypervolume improvement criterion, without the diversity metric. The comparison result is shown in Figure 8, which demonstrates that DGEMO highly benefits from the continuous Pareto front approximation, as denser candidate solutions are provided and the diversity information becomes available from this representation.

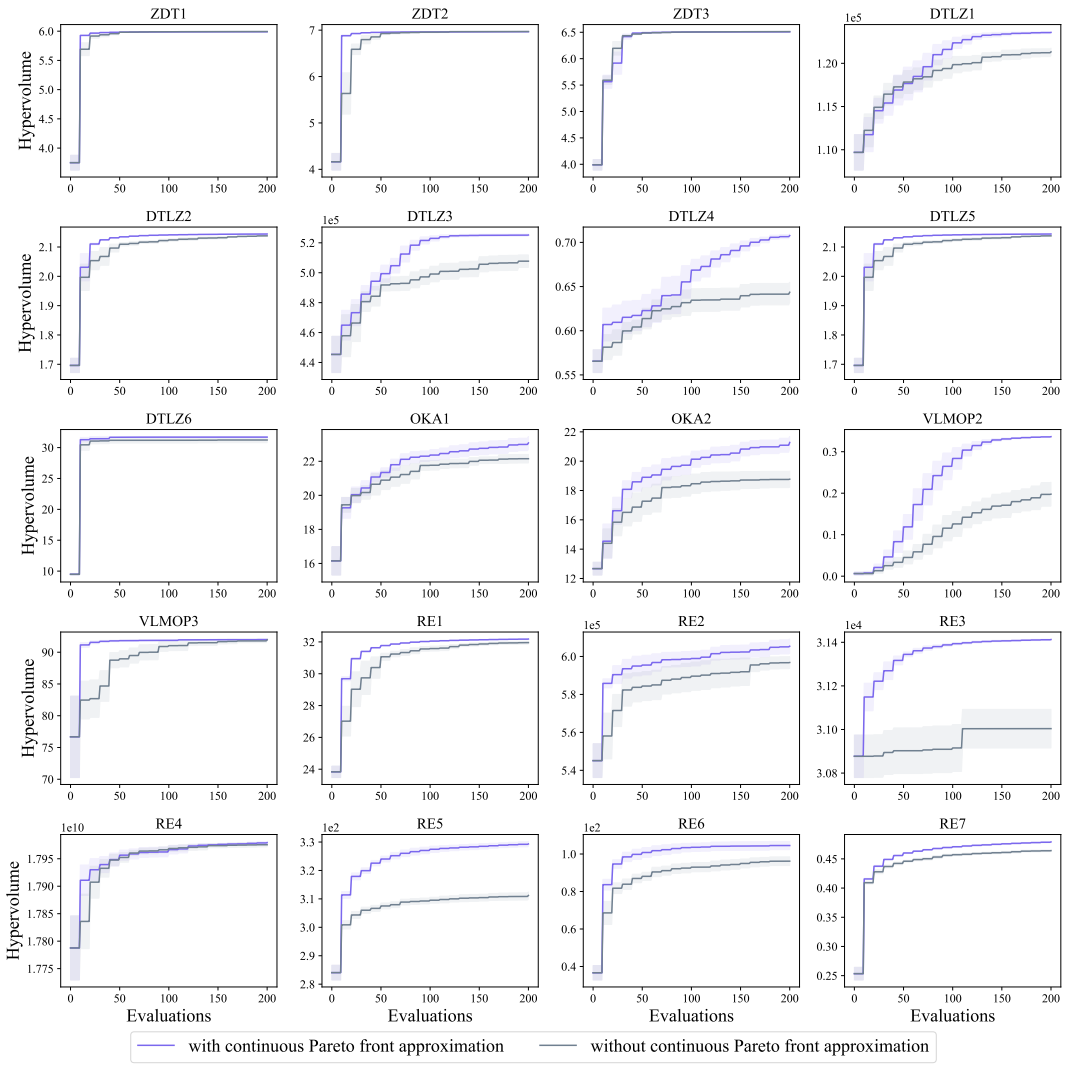

Figure 8: Hypervolume indicator of DGEMO and its version without continuous Pareto front approximation on all problems with batch size as 10, shown with respect to a different number of function evaluations. Note that the version without the continuous Pareto front approximation does not have the diversity metric incorporated in the selection strategy, but only uses the maximal hypervolume improvement information.

F Pareto set analysis

For many complex real-world design problems, there are obvious clusters in the Pareto-optimal solutions. It is therefore desirable to fully explore these diverse regions of interest for a more comprehensive understanding of the problem and having a wider range of solutions.

We take the rocket injector design problem (RE7) [2] as an example. In this problem, the primary objectives are performance and material sustainability. Specifically, the goal is to minimize the combustion length (X_{cc}) for better size and efficiency of the combustor, while at the same time minimizing the maximum temperature of injector face (TF$_{max}$) and post tip (TT$_{max}$) for a longer material life. The design variables consist of hydrogen flow angle (α), hydrogen area ($\triangle$HA), oxygen area ($\triangle$OA), and oxidizer post tip thickness (OPTT).

The Pareto-optimal solutions of this problem are grouped in several regions in design space, and these groups correspond to different patches on the Pareto front, see [9]. This phenomenon shows that for the rocket injector structure, there is no single optimal solution that guarantees the optimal performance on all the objectives involved. Instead, there are different design patterns that lead to optimal properties with different trade-offs. To understand the deeper connection between design space and performance space, how certain design variables lead to Pareto optimality, and the pattern difference between these groups, we explore diversity regions detected by our algorithm. Figure 9 presents a visualization of the 3-dimensional perofrmance space, where each diversity region is represented with a different color.

From Figure 9 we can see different clusters are indeed responsible for different Pareto-optimal regions on the Pareto front. By analyzing the design variables of the Pareto set, we list several interesting observations illustrated as follows:

- The widest blue region of the Pareto front includes solutions with the highest TT$_{max}$. This region is the only region with Pareto-optimal points that have the OPTT > 0, and these points are located in the upper part of the blue region. We note that larger OPTT design variable leads to larger TT$_{max}$. Hence, the oxidizer thickness has the most important contribution to the longer material life of the post tip and solely contributes to this objective. In addition, an important design characteristic of this region is that the oxygen area is always 0, while the other two design variables (hydrogen flow angle and hydrogen area) vary between 0 and 1. Varying the values of α and $\triangle$HA, controls the trade-off between X_{cc} and TF$_{max}$.
- When hydrogen flow angle and oxygen area are maximally utilized ($\alpha = 1$ and $\triangle$OA $= 1$), TT$_{max}$ has the minimal optimal value (gray region). By ranging the value of $\triangle$HA from 0.1 to 1, X_{cc} increases while TF$_{max}$ decreases, respectively. Hence hydrogen area controls the trade-off between the TF$_{max}$ and X_{cc}, similar to the pattern we found in blue region.
- When the hydrogen area is maximal ($\triangle$HA $= 1$), oxygen area is very high ($\triangle$OA > 0.9), oxydizer post tip is not in use (OPTT $= 0$), and the hydrogen flow angle ranges from 0.5 to 1, the changes in TF$_{max}$ are negligible, while TT$_{max}$ decreases from 0.1 to -0.3 as α increases (purple region). For applications targeting a design with minimal face temperature, we found the design variables should have maximal hydrogen area and oxygen area, at the same time with minimal hydrogen flow angle and zero OPTT, as shown in the green region.
- In general, among the Pareto-optimal solutions, a small flow angle indicates low face temperature and high tip temperature. Increasing the flow angle leads to an increase in the face temperature and a decrease in the combustion length. Large hydrogen area corresponds to low tip and face temperature but long combustion length. Most of the Pareto-optimal solutions have a relatively low tip thickness.

As a result, by analyzing the regions of on the Pareto set we can gain more understanding about the nature of the problem. In many complex real-world problems, such as this rocket injector design, the Pareto-optimal solutions are grouped in different regions and present different performance trade-offs. The DGEMO algorithm can discover these diverse sets of solutions by directly taking the diversity in

[2] The original rocket injector design problem [9] has four objectives defined. However, according to their analysis, two of them have strong correlation, hence we simplify it as a three objective problem as also suggested in [9].

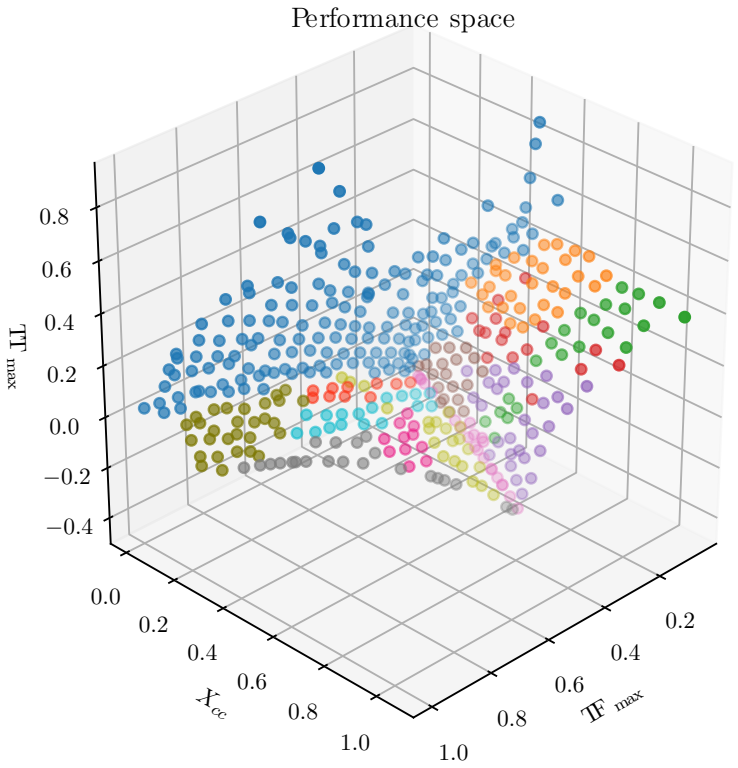

Figure 9: Performance space visualization of Pareto-optimal solutions and their diversity regions found by DGEMO on rocket injector design (RE7) problem (in 3D visualization, the opacity of points represent the depth from current point of view).

both design space and performance space into account. From the solutions found by our algorithm, we can easily extract the representative design patterns from these regions of Pareto-optimal designs, and potentially combine the knowledge concluded from these patterns with prior knowledge to discover an even better Pareto set.

References

[1] Hossain M Amir and Takashi Hasegawa. Nonlinear mixed-discrete structural optimization. *Journal of Structural Engineering*, 115(3):626–646, 1989.

[2] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239, 2001.

[3] FY Cheng and XS Li. Generalized center method for multiobjective engineering optimization. *Engineering Optimization*, 31(5):641–661, 1999.

[4] Carlos A Coello Coello and Margarita Reyes Sierra. A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In *Mexican international conference on artificial intelligence*, pages 688–697. Springer, 2004.

[5] Kalyanmoy Deb, Ram Bhushan Agrawal, et al. Simulated binary crossover for continuous search space.

[6] Kalyanmoy Deb and Mayank Goyal. A combined genetic adaptive search (geneas) for engineering design. 1996.

[7] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[8] Kalyanmoy Deb and Aravind Srinivasan. Innovization: Innovating design principles through optimization. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1629–1636, 2006.

[9] Tushar Goel, Rajkumar Vaidyanathan, Raphael T Haftka, Wei Shyy, Nestor V Queipo, and Kevin Tucker. Response surface approximation of pareto optimal front in multi-objective optimization. *Computer methods in applied mechanics and engineering*, 196(4-6):879–893, 2007.

[10] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[11] Dieter Kraft. A software package for sequential quadratic programming. *Forschungsbericht-Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[12] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[13] Michael D McKay, Richard J Beckman, and William J Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[14] Kaisa Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media, 2012.

[15] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[16] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[17] Tapabrata Ray and KM Liew. A swarm metaphor for multiobjective design optimization. *Engineering optimization*, 34(2):141–153, 2002.

[18] Adriana Schulz, Harrison Wang, Eitan Grinspun, Justin Solomon, and Wojciech Matusik. Interactive exploration of design trade-offs. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[19] David A Van Veldhuizen. Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Technical report, AIR FORCE INST OF TECH WRIGHT-PATTERSONAFB OH SCHOOL OF ENGINEERING, 1999.

[20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 2020.