Exploring Interesting Regions with Bayesian Inference and Active Learning

SALA Raphaël

July 12, 2024

1 Context

Active learning focuses on improving predictive models by selectively choosing the most informative data points for evaluation. The primary goal is to enhance model predictions by using Bayesian inference to understand the behavior of an unknown objective function, f_{obj}. This approach is particularly valuable when f_{obj} is costly or time-consuming to evaluate, and information about it is limited to a few sampled points.

Our work is concerned with exploring and identifying points within a user-defined region of interest in as few steps as possible. This is distinct from traditional optimization problems, which aim to maximize or minimize a given objective function. Instead, we focus on efficiently exploring a specified interval, denoted as K.

2 Previous works

2.1 Gaussian Process Regression

Due to the high-dimensionality problem, we work in low-dimensional spaces. This report benchmarks our model on an applied physics case study : Targeted nano-energetic material exploration using a Bayesian algorithm [1]. We define $K = [L_p; U_p] \times [L_T; U_T] \subset [0, 1]^2$, where L_p and U_p are the lower and upper bounds for pressure, and L_T and U_T are the bounds for temperature. The input points belong to $\mathbb{R}^3$ (e.g., radii of CuO and Al particles, r_{CuO}, r_{Al}, and stoichiometry, ϕ).

The main idea of the paper [1] is to consider values returned by a simple Gaussian Process, $GP : \mathbb{R}^3 \mapsto \mathbb{R}^2$ and to evaluate points based on their variance and interest. The interest function is defined as $I : x \mapsto \mathbb{P}\left(GP(x) \in K\right)$. Indeed, considering only the variance of each point allows us to understand the target space whereas we are specifically interested in the region K. On the other hand, focusing only on I directs sampling towards interesting regions but results in exploitation without sufficient exploration: points sampled by this truncated algorithm may be too close to each other, providing limited information about the behavior of f_{obj} in K. Then, with the variance and interest score, new point is sampled with simplicial homology global optimization. This model is referred to as GPR+I.

The idea to have new points sampled far than already choosen is implemented in the model GPR+I+C. The choice of a new point x is according to its variance, $\mathbb{V}\left(GP(x)\right)$, its interest, $I(x)$ and a additional term, $C(x)$, designed to encourage exploration. The function C is defined as follows,

$$C_{s,d,A}(u) = 1 - \frac{1}{1 + \exp(A - s\Gamma_d(u))}$$
$$\Gamma_d(u) = \sum_{x \in X_k} \exp(-d\|u - x\|^2)$$

Here, $x \in X_k$ are the already explored points, and d, s, A are parameters detailed in [1]. The function C penalizes points that are near those already sampled.

We give algorithm for both models, with parameters required θ including s, d, A. The numbers of points to evaluate, N_{iters}, and D_n a dataset with n points evaluated by f_{obj}. For GPR+I, remove the line 9 and so, the last term in line 10 to compute $Acq(x)$

Algorithm 1 GPR+I+C: Gaussian Process Regression with Interest and Coverage Function

Require: N_{init}, N_{iter}, θ

Initialize X_{init} using Latin Hypercube Sampling
Initialize $D_{N_{init}} = (X_{init}, y_{init})$
for $n = N_{init}$ to $N_{init} + N_{iter} - 1$ **do**
    Train GP model $\mathcal{GP}$ on D_n
    Initialize an empty list $Acq\_values = []$
    for $x \in D_n[0]$ **do** ▷ Get the first component of each tuple in D_n
        Compute $V(x) = \mathbb{V}(GP(x))$
        Compute $I(x) = \mathbb{P}(GP(x) \in K)$
        Compute $C(x) = 1 - \frac{1}{1+\exp(A - s\Gamma_d(x))}$
        Compute $Acq(x) = V(x) + I(x) + C(x)$
        Append $-Acq(x)$ to Acq_values ▷ "-" because SHGO minimizes the function
    end for
    Optimize $Acq\_opti = \text{shgo}(D_n[0], Acq\_values)$
    Compute $x_{n+1} = \arg\max_x Acq\_opti(x)$
    Evaluate $y_{n+1} = f_{obj}(x_{n+1})$
    Update dataset $D_{n+1} = D_n \cup (x_{n+1}, y_{n+1})$
end for
return $D_{N_{init}+N_{iter}}$

2.2 ParEGO: Multi-Objective Optimization

The Efficient Global Optimization (EGO) algorithm is designed for optimizing expensive black-box functions. EGO uses a Gaussian Process (GP) to model the objective function and selects new points to evaluate by maximizing the expected improvement (EI). The expected improvement balances exploration and exploitation by considering both the predicted value and the uncertainty of the prediction [2].

ParEGO extends EGO to multi-objective optimization. Instead of optimizing a single objective, ParEGO converts multiple objectives into a single scalar objective using a parameterized scalarizing weight vector. This scalarization allows the GP to handle multiple objectives by creating a single aggregated objective function. At each iteration, a weight vector λ is drawn uniformly at random from the set

$$\Lambda = \left\{ \lambda = (\lambda_1, \lambda_2, \ldots, \lambda_k) \mid \sum_{j=1}^{k} \lambda_j = 1 \wedge \forall j, \lambda_j = \frac{l}{s}, l \in \{0, \ldots, s\} \right\}$$

where k is the number of objectives, here $k = 2$. The parameter s is a hyper-parameter that controls the size of Λ. A smaller, compared to the number of algorithm's iteration, s increases the probability of obtaining extreme weight vectors, while a larger s allows for more balanced weight vectors. Benchmark in section 4 use $s = 1000$. The scalar objective function $f_\lambda(x)$ is then computed using the augmented Tchebycheff function:

$$f_\lambda(x) = \max_{i \in [\![1,k]\!]} \{\lambda_i f_i(x)\} + \rho \sum_{i=1}^{k} \lambda_i f_i(x)$$

where $f_i(x)$ are the individual objective functions, and ρ is a small positive value, the paper [**?**] suggest $\rho = 0.05$. The term with ρ is added because:

> The nonlinear part of the function means that points on nonconvex regions of the Pareto front can be minimizers of this function and, thus, nonsupported solutions can be obtained, while the linear part of the function ensures that solutions that are weakly dominated by Pareto optimal solutions are rewarded less than Pareto optimal ones [3].

The next point to evaluate is chosen by maximizing the expected improvement with respect to the scalar objective function f_λ using a genetic algorithm. This involves initializing a temporary population of solution vectors, some of which are mutants of previously evaluated points while others are generated randomly. The genetic algorithm iteratively evaluates the expected improvement of

these solutions using the Gaussian Process model, on f_λ, and applies selection, recombination, and mutation operations to form a new population. This process continues until a predefined number of evaluations is reached, ultimately returning the solution that maximizes the expected improvement after a large number of population's generation [4].

3 Methodology

3.1 Improving Scalarization

ParEGO uses a parameterized scalarizing weight vector to explore the search space by considering different objectives to varying extents during each iteration. This approach typically maximizes the objectives using the function $\max_{i\in\{1,\dots,k\}}\{\lambda_i f_i(x)\}$. However, since we are interested in exploring the region K, we need to transform the values given by f_i.

We define a function $g : \mathbb{R}^k \to \mathbb{R}$, that maps x to a scalar value indicating the distance with the region K, as,

$$g(x) = \begin{cases} 1 & \text{if } x \in K, \\ 1 - \|x - K\|_1 \cdot \text{slope} & \text{if } x \notin K \end{cases}$$

Slope is a parameter. For the section 4, we set slope=10 to accentuate the importance to be in region K. The expression of g resembles a tent, which is reflected in the algorithm's name. The hyper-parameters for this algorithm include s and slope. Additionally, there are parameters inherited from the genetic algorithm [4], with the only one referenced later being *num_gen*, which determines the number of generations to iterate and converge to an optimal population. This is summarized as follows,

Algorithm 2 parEGO_tent

Require: N_{init}, N_{iter}, θ

Initialize X_{init} using Latin Hypercube Sampling
Initialize $D_{N_{init}} = (X_{init}, y_{init})$
for $n = N_{init}$ to $N_{init} + N_{iter} - 1$ **do**
    Choose a random $\lambda \in \Lambda$
    Initialize an empty list $y_\lambda = []$
    for $y \in D_n[1]$ **do** ▷ Get the second component of each tuple in D_n
        Append $\max_{i\in[\![1,k]\!]}\{\lambda_i g(y)\} + \rho\sum_{i=1}^{k}\lambda_i g(y)$ to y_λ
    end for
    Train GP model $\mathcal{GP}$ on $(D_n[0], y_\lambda)$
    Select a population of points, X_{pop} from $D_n[0]$ with some mutants
    for i=1 to num_gen **do**
        Compute score = $EI \circ \mathcal{GP}(X_{pop})$
        Reproductive selection on X_{pop} : Replacement, crossover, mutation according to score
    end for
    Compute $x_{n+1} = \arg\max_{x\in X_{pop}} EI \circ \mathcal{GP}(x)$
    Evaluate $y_{n+1} = f_{obj}(x_{n+1})$
    Update dataset $D_{n+1} = D_n \cup (x_{n+1}, y_{n+1})$
end for
return $D_{N_{init}+N_{iter}}$

3.2 Improving Inputs

Another approach is to avoid scalarization as it can lead to information loss. Instead of using SHGO for optimization, we can utilize a genetic algorithm. This method focuses solely on the interest function. The issues of point proximity are mitigated because mutations and crossovers in the genetic algorithm facilitate continuous and natural exploration, rather than just exploitation. So we replace f_λ by a Gaussian Process like GPR+I. This approach allows us to directly maximize the probability of being in the interest region without unnecessary complexities. The algorithm is described as follows,

Algorithm 3 parEGO_GP

Require: N_{init}, N_{iter}, θ

Initialize X_{init} using Latin Hypercube Sampling
Initialize $D_{N_{init}} = (X_{init}, y_{init})$
for $n = N_{init}$ to $N_{init} + N_{iter} - 1$ **do**
    Train GP model $\mathcal{GP}$ on D_n
    Initialize an empty list $Itr\_values = []$
    for $x \in D_n[0]$ **do** ▷ Get the first component of each tuple in D_n
        Compute $I(x) = \mathbb{P}(GP(x) \in K)$
        Append $I(x)$ to Itr_values
    end for
    Select a population of points, X_{pop} from $D_n[0]$ with some mutants
    for i=1 to num_gen **do**
        Compute interest $= EI \circ \mathcal{GP}(X_{pop})$
        Reproductive selection on X_{pop} : Replacement, crossover, mutation according to interest
    end for
    Compute $x_{n+1} = \arg\max_{x \in X_{pop}} EI \circ \mathcal{GP}(x)$
    Evaluate $y_{n+1} = f_{obj}(x_{n+1})$
    Update dataset $D_{n+1} = D_n \cup (x_{n+1}, y_{n+1})$
end for
return $D_{N_{init}+N_{iter}}$

4 Results

4.1 Metrics

To compare the performance of these algorithms, we used two metrics: n_interest and coverage. At the end of the experiment, we count the number of points in the interest region K, referred to as n_interest. To determine if these points are close to each other, we compute the volume of spheres with a predefined radius centered at each point.

Analytically, calculating the volume of spheres (in our case, in $\mathbb{R}^3$) is not straightforward. Therefore, we approximate this by identifying clusters of spheres. Simple clusters, consisting of one or two spheres, are computed analytically if they do not intersect the boundary limits. Complex clusters include those intersecting the domain boundaries (with normalized data, points outside $[0, 1]^3$ are ignored) and clusters with more than two spheres.

For these complex clusters, we use the octree technique. This involves creating a parallelepiped around the clusters and subdividing it into smaller equal parts. We sum the volumes of the subdivisions that are entirely within the clusters and ignore those outside. For subdivisions intersecting the clusters, we recursively subdivide them further. This process is repeated to obtain a precise volume estimate.

4.2 Experiments

Based on physical considerations, the radius of the spheres used to compute coverage is set to 0.025. Our objective is to understand the behavior of f_{obj}, which in our case is time-intensive. To reduce simulation time, we parallelize the algorithms when we have to evaluated f_{obj}, so we have to obtaining multiple new input points at each iteration instead of just one. We use a batch size of 8 for these experiments. The impact of this choice is discussed in Section 4.5.

We explore two different interest regions to evaluate how the algorithms perform under varying constraints:

- $R1 : [1e6, 2e6] \times [2e3, 4.5e3]$
- $R2 : [4e7, 5.6e7] \times [5.8e3, 7e3]$

Each experiment starts with the same 100 initial points and concludes after finding 200 new values, resulting in a total of approximately 300 points.

4.3 Results for R1

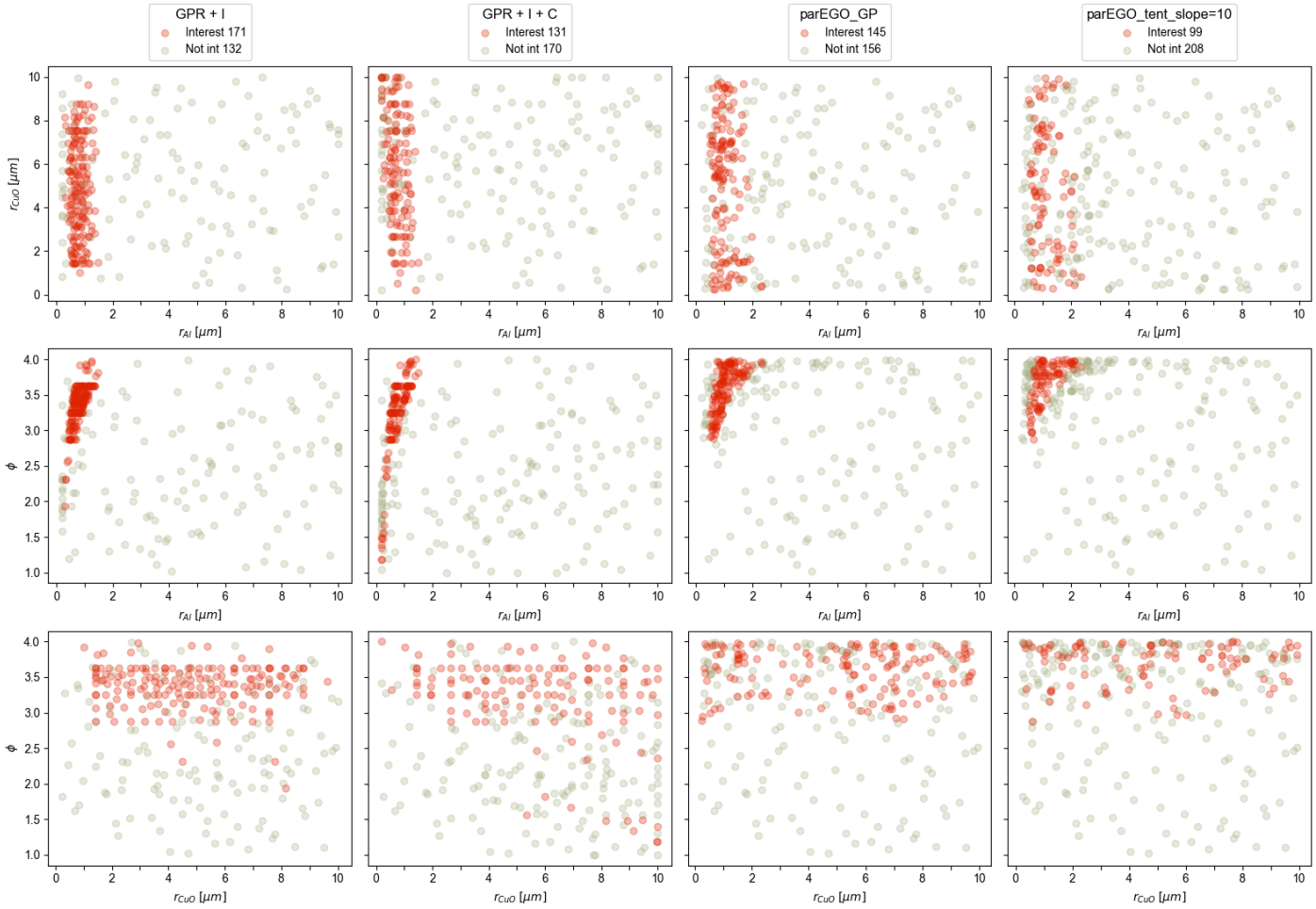

Figure 1: FIG. 3. Spatial distribution of the total number of sampled points (300) projected onto main feature scale planes. Red balls correspond to points of interest, while gray balls represent non-interesting points.

Added points	**GPR+I**		GPR+I+C		parEGO_GP		*parEGO_tent*	
	in #	in %	in #	in %	in #	in %	in #	in %
0	4	4	4	4	4	4	4	4
100	**87**	**43.5**	81	40.5	78	39	*52*	*26*
200	**168**	**56**	131	43.4	144	48	*95*	*31.7*

Table 1: Evolution of the number of points of interest (in #) and their ratio in % over the total number of sampled points (in italics) calculated after each increment of 100 additional points.

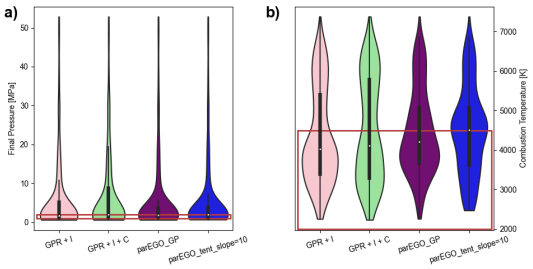

Figure 2: Violin plots of the targets: a) Final pressure, b) Combustion temperature obtained with each active sampling scenario. The red lines mark the limits of the region of interest.

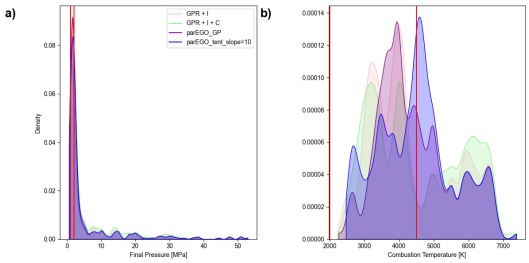

Figure 3: KDE plots of the targets: a) Final pressure, b) Combustion temperature obtained with each active sampling scenario. The red lines mark the limits of the region of interest.

Volume	**GPR+I**	GPR+I+C	parEGO_GP	*parEGO_tent*
×	**0.010133**	0.007667	0.007690	*0.005346*

Table 2: The covering volume of interest points.

4.4 Results for R2

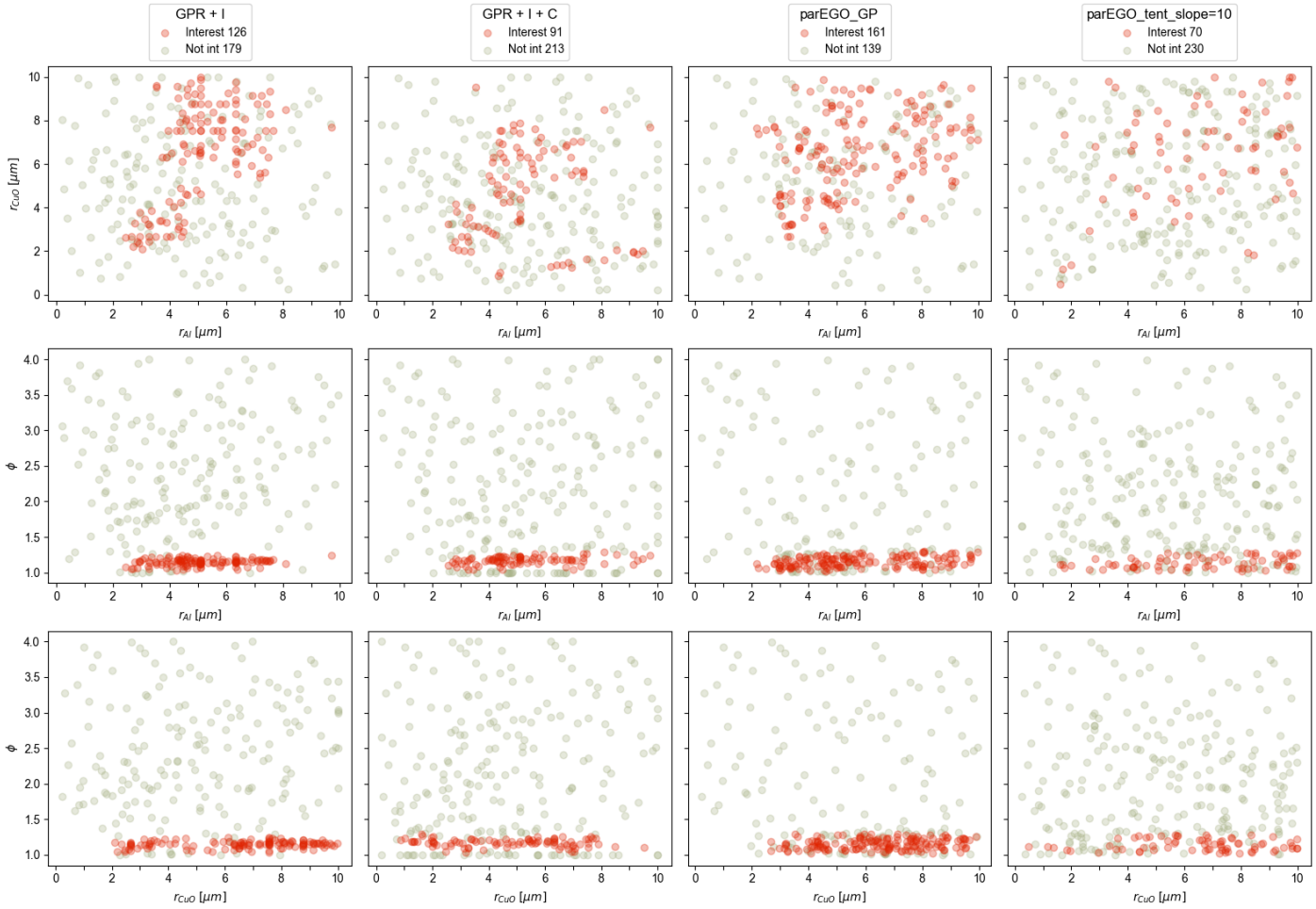

Figure 4: Enter Caption

Added points	GPR+I		GPR+I+C		**parEGO_GP**		parEGO_tent	
	in #	in %	in #	in %	in #	in %	in #	in %
0	5	5.0	5	5.0	5	5.0	5	5.0
100	44	22	*34*	*17*	**80**	**40**	37	18.5
200	121	40.3	89	29.7	**161**	**53.7**	*70*	*23.3*

Table 3: Evolution of the number of points of interest (in #) and their ratio in % over the total number of sampled points (in italics) calculated after each increment of 100 additional points.

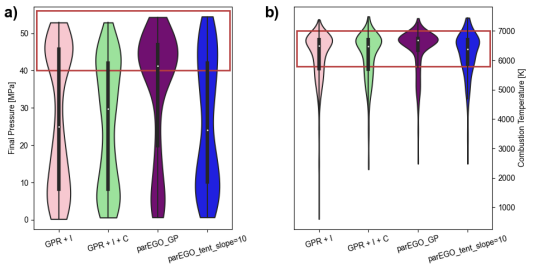

Figure 5: Violin plots of the targets: a) Final pressure, b) Combustion temperature obtained with each active sampling scenario. The red lines mark the limits of the region of interest.

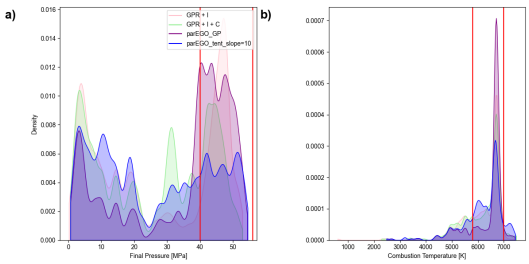

Figure 6: KDE plots of the targets: a) Final pressure, b) Combustion temperature obtained with each active sampling scenario. The red lines mark the limits of the region of interest.

Volume	GPR+I	GPR+I+C	**parEGO_GP**	*parEGO_tent*
×	0.007011	0.005006	**0.008742**	*0.004049*

Table 4: The covering volume of interest points.

4.5 Discussion

The GPR+I algorithm combines Gaussian Process Regression, incorporating a variance term with an interest function to guide the sampling process. It effectively identifies a high number of points within the region of interest. However, GPR+I+C, which includes an additional coverage term, reduces the number of interest points as expected by choosing new samples meticulously, but this also impacts the overall coverage.

We introduced two variations of the initial ParEGO algorithm. ParEGO_tent was able to find some points of interest, but the results were not sufficient to warrant further study. It seems the lack of information in each dimension appears to have impacted its exploitation capabilities. On the other hand, ParEGO_GP, which uses a genetic algorithm to optimize the Gaussian Process directly based on the interest function, showed promising results in maintaining a balance between exploration and exploitation.

Future work should focus on further refining these algorithms. The current hyper-parameters are not well understood and were chosen based on intuition and limited benchmarking, which may not be truly representative. A more mathematical approach is needed to deduce which algorithm is best suited for this type of problem. For instance, the role of the ρ term in the scalarized objective function of ParEGO_tent may be unnecessary since this term helps convergence towards a Pareto front, which our problem does not have. Emphasis should be placed on tuning the genetic algorithm to leverage known information from the Gaussian Process effectively.

In conclusion, while returning multiple points per iteration can save time, reducing the batch size may significantly improve results. For the first two algorithms working with Gaussian Process Regression, the batch size does not greatly impact the results because a single point in such a space does not drastically change the prediction. However, for ParEGO_tent, each iteration involves selecting a different λ to explore various configurations and then revisiting to assess new insights. Thus, reducing the batch size allows for computing different configurations of weight vectors more effectively. Additionally, in genetic algorithms, selecting the best member of the final population is crucial. If too many members are selected, some may be mutants or results of crossover operations, which do not accurately represent the genetic algorithm's learning process.

References

[1] Leandro Carreira, Lea Pillemont, Yasser Sami, Nicolas Richard, Alain Esteve, Matthieu Jonckheere, and Carole Rossi. Targeted nano-energetic material exploration through bayesian algorithm

implementation. *Fuel*, June 28 2024.

[2] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[3] Joshua Knowles. Parego: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. Technical Report TR-COMPSYSBIO-2004-01, University of Manchester, September 2004.

[4] Joshua Knowles and Evan J. Hughes. Multiobjective optimization on a budget of 250 evaluations. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 176–190. Springer, 2005.