4 Explore-Exploit Measures

4.1 Idea

Numerical optimization methods are characterized by a trade-off between exploration and exploitation. An optimization algorithm is said to explore when aiming at reaching various promising regions in the search space. Exploration prevents the algorithm from getting stuck in a single region. On the other hand, exploitation means optimizing in such a single region. It ensures finding optimal values within certain promising regions. Too much exploration can lead to inefficient optimization paths, whilst too much exploitation may cause strong dependence on the starting point and eventually getting stuck in a local optimum.

This concept can help understanding the proposals of the algorithm. In the human-in-the-loop-MBO, for instance, the human operator may want to know if the MBO found a new promising parameter constellation and thus exploited the respective area of the parameter space. If it - on the contrary - just explored some absurd parameter constellation, domain experts may be warned not to waste their time chasing the end of the rainbow. Long story short, the explore-exploit-trade-off is easy to understand and can help interpreting the MBO process. To this end, we develop several measures describing whether the MBO exploits or rather explores in a given iteration: Standard Error Ratio (SER), Standard Error Distribution Value (SED), Pairwise Distance, Mean Distance, Maximal Distance and Minimal Distance. We call them explore-exploit measures.

They are computed by the functions `xplxpl()` and `xplxplMBO()` and their common subfunction `computeXplxpl()` as well as various subsubfunctions that depend on the nature of the MBO: single- or multi-point proposals, single- or multi-objective functions, numeric or categorical parameter space and the `scope` argument. We start with single-criteria (i.e. single-objective) optimization with one proposed point per iteration (single-point proposal). We focus on this standard case, as our project partner is currently not using multi-point or multi-crit MBOs [7]. Yet, an extension to multi-point proposal is scheduled, which is why we adapt our explore-exploit measures to one such multi-point technique, namely Constant Liar. A simple but not complete extension to multi-objective is also shown later on.

4.2 Single-Objective Functions with Single-Point Proposal

4.2.1 Implementation

The variance-based measures SER and SED compare the actually proposed point in a given iteration with so-called candidate (or seen) points. Recall that MBO proposes points by optimizing the infill criterion (see page 6 in [3]). For single-objective functions, `mlrMBO` [3] offers three different infill optimization techniques: `infillOptFocus()` does Focus Search, `infillOptEA()`

uses Evolutionary Algorithms and `infillOptcmaes()` applies a Covariance Matrix Adaptation-Evolution Strategy (cf. package `cmaesr`). During this process within each iteration many candidate points are evaluated, which we need to compute our measures. As `mbo()` does not store these candidate points from infill optimization, we change the infill optimizers to `infillOptFocusSavepts()`, `infillOptEASavepts()` and `infillOptcmaesSavepts()`. We further edit their dependencies[1] , so that they store the candidate points of each infill optimization round (it is restarted some times) of each iteration in the resulting MBO object. These candidates and the fitted surrogate models of all iterations (set `store.model.at = 1:(iters + 1)` in `makeMBOControl()`) allow `xplxpl()` and `xplxplMBO()` to compute the candidate points' standard errors.

Users can either apply `xplxpl()` to an MBO result or directly run `xplxplMBO()` instead of `mbo()`. If `xplxpl()` is applied to an MBO result that does not contain both the candidate points from infill optimization and the surrogate models from each iteration, the user is asked if he would like to run `xplxplMBO()` with settings parsed from the MBO result.[2]

4.2.2 Variance-based Measures

Standard Error Ratio and Standard Error Distribution Value are variance-based measures, that is, they make use of the estimation of the posterior standard error $\hat{s}(\mathbf{x})$ by the surrogate models. In `mlrMBO` this is supported for the common surrogate regression methods Kriging (Gaussian Process) and Random Forest by setting `predict.type = "se"` in the learner object. While Kriging estimates posterior variance[3] intrinsically, Random Forest needs additional variance estimators, such as jackknife-after-bootstrap (default in `mlrMBO`). In case of other regression models, bagging can be applied to retrieve standard error estimators in `mlr`, see pages 5 and 9 in [3].

The idea behind the variance-based measures is simple: The standard error expresses the degree of uncertainty in the surrogate model. If the proposed point's standard error is high compared to those of other candidate points during infill optimization, the MBO aims at reducing uncertainty about a previously unknown area in the parameter space. That is, it *explores*. `xplxpl()` and `xplxplMBO()` compute the ratio of the proposed point's standard error $\hat{s}(\mathbf{x}^*)$ to the mean standard error of n other candidate points (Standard Error Ratio,

[1] `checkStuff()`, `extractInfoMBO()`, `makeMBOResult.OptState()` and most importantly `proposePointsByInfillOptimization()` as well as the two initializing functions `getInfillOptFunctions()` and `getSupportedInfillOptFunctions()`.

[2] This will not reproduce the same optimization path, as Kriging and Random Forest are of stochastic nature. It will rerun the MBO with same control parameters. To our knowledge, it is not possible to simply "roll back" the RNG as to access the previous seed that was used in initial MBO run.

[3] Variance of the mean prediction, hence also the standard error. Standard error and standard deviation are often used interchangeably, while they are related but different concepts. The standard error is defined as the root of an **estimator**'s variance, whilst the standard deviation refers to random variables of any kind, which makes the standard error a special case of the standard deviation.

SER) and its location in the distribution of candidate points' standard errors (Standard Error Distribution Value, SED):

$$SER = \frac{\hat{s}(\mathbf{x}^*)}{\frac{1}{n}\sum_{i=1}^{n}\hat{s}(\mathbf{x}_i)}, \quad SED = F_{\hat{s}(\mathbf{x})}(\hat{s}(\mathbf{x}^*)), \tag{2}$$

where $F_{\hat{s}(\mathbf{x})}(\bullet)$ is the empirical distribution function of $\hat{s}(\mathbf{x}_1), ..., \hat{s}(\mathbf{x}_n)$. The functions obtain the candidate points' standard errors by predicting on those points using the surrogate model from the respective iteration. The standard errors of the proposed points are provided by `mbo()` in `opt.path` in `mbo.result`, except for the most simple infill criteria Mean Response and Standard Error.[4] In these cases, the proposed points' standard errors are obtained the same way as for candidate points.

Both `xplxpl()` and `xplxplMBO()` have a `scope` argument that can be set to `"local"` (default) or `"global"`. The former starts `xplxplSESingleLocal()`, which compares the proposed point of iteration i to the seen points in that same iteration. The latter starts `xplxplSESingleGlobal()`, which compares the proposed point of iteration i to previously seen points from iterations $1, ..., i$. While the local approach seems more intuitive to explain the decision of the infill optimizer, the global measures might be helpful for understanding the overall explore-exploit behaviour. Howsoever, we recommend using the local (default) scope.

4.2.3 Distance-based Measures

Another way of measuring the explore-exploit-trade-off takes distances into account: The farther away the proposed point is from its predecessor(s), the more the algorithm explores. `xplxpl()` and `xplxplMBO()` compute the pairwise distance from each proposed point to its direct predecessor as well as the mean, maximal and minimal distance to all predecessors. The latter three distances are also computed for the first iteration, as an intuitive interpretation exists: These are simply the distances to the initial data. This is not the case for pointwise distances, as the first proposed point is not intuitively linked to the last point of the initial training data set.

In case of an exclusively numeric parameter space, we use the euclidean norm to compute distances. With one or more parameters being categorical, the gower distance ([5], chapter 3, slide 6/16) is applied. It computes the distance of data point $\mathbf{x}$ to $\tilde{\mathbf{x}}$ (both p-dimensional, i.e. with p features) as follows:

$$d_{gower}(\mathbf{x}, \tilde{\mathbf{x}}) = \frac{\sum_{j=1}^{p}\delta_{\mathbf{x}_j,\tilde{\mathbf{x}}_j} \cdot d_{gower}(\mathbf{x}_j, \tilde{\mathbf{x}}_j)}{\sum_{j=1}^{p}\delta_{\mathbf{x}_j,\tilde{\mathbf{x}}_j}}, \tag{3}$$

with $\delta_{\mathbf{x}_j,\tilde{\mathbf{x}}_j}$ being equal to 0 when the j-th variable is missing in at least

[4] Although opt.path contains a column named `se` in case of `infill.crit = "se"`, its values are negative. This is due to the fact that the infill criterion is minimized internally. See `mlrMBO` issue 488 on Github for more details.

one of the observations.[5] Otherwise it is 1. $d_{gower}(\mathbf{x}_j, \tilde{\mathbf{x}}_j)$ is the j-th variable's contribution to the total distance. For categorical variables the distance is 0 if both values are equal and 1 otherwise. The contribution of other variables is the absolute difference of both values, divided by the total range of that variable.

4.3 Application on Data and Analysis of Measures

We apply our explore-exploit measures on various MBO runs on the provided data sets (kapton, material design). We vary the infill optimizer as well as the infill criterion, using all seven built-in single-objective[6] infill criteria that come with `mlrMBO`: Expected Improvement (`ei`), Mean Response (`mr`), Standard Error (`se`), (Lower) Confidence Bound (`cd`), Adaptive Confidence Bound (`adacb`), Augmented Expected Improvement (`aei`) and Expected Quantile Improvement (`eqi`). We also vary the the control parameter λ in the (Lower) Confidence Bound infill criterion. Furthermore, we look into how strongly our measures correlate and compare them. We are aware the results depend on the RNG. Yet, the overall explore-exploit behaviour of the MBO was found to be robust towards the seed, which is why we report our results for only one given seed, see TODO. Please find all results in TODO.

4.3.1 Material Design

For material design data, we vary the infill optimization and use focus search (`focussearch`), evolutionary algorithm (`ea`) and Covariance Matrix Adaptation-Evolution Strategy (`cmaes`). As `cmaes` does not work on integer data yet (see `mlrMBO` issue 475 on Github), we exclude integer variables from the data set in this case.

Figure 2 and 3 show an example of the Explore Exploit Measures for material design data with `focussearch` as infill optimization and Expected Improvement (`ei`) as infill criterion. Both a local (2) and a global (3) scope haven been used. It becomes evident that the popular Expected Improvement criterion succeeds at exploring the parameter space first, before exploiting specific areas in later iterations.

Besides, the distance-based measures apparently increase in some of the later iterations (cf. Mean Euclidean Distance, Iteration 39), whilst variance-based measure decrease in a more or less stable way. This gives rise to the question whether our measures correlate or not. We will analyze this in more detail and discuss reasons later.

The (Lower) Confidence Bound (`cb`) infill criterion incorporates the explore-exploit-trade-off directly by its weighting parameter λ (see pages 3 and 4 in [4]).

[5] Or when the variable is asymmetric binary and both values are zero [5]. Both cases are irrelevant to our application, as to our knowledge mbo never proposes points with NA entries.

[6] Direct Indicater Based (`dib`) is applicale to multi-criteria optimiziation only, its single-objective analogue is `cb`.

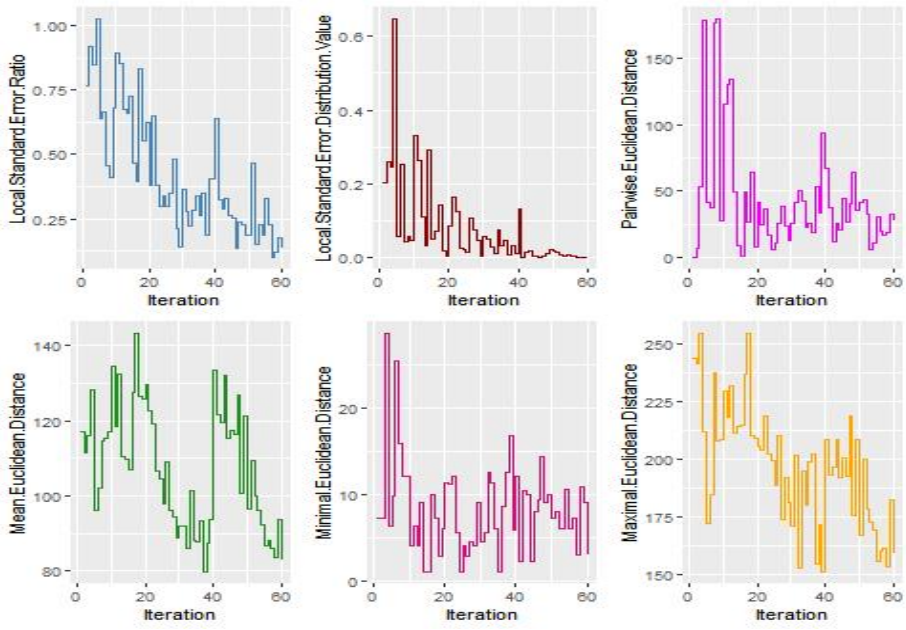

Figure 2: Loxal Explore-Exploit Measures (`focussearch`, `ei` and 60 Iterations) on material design data.

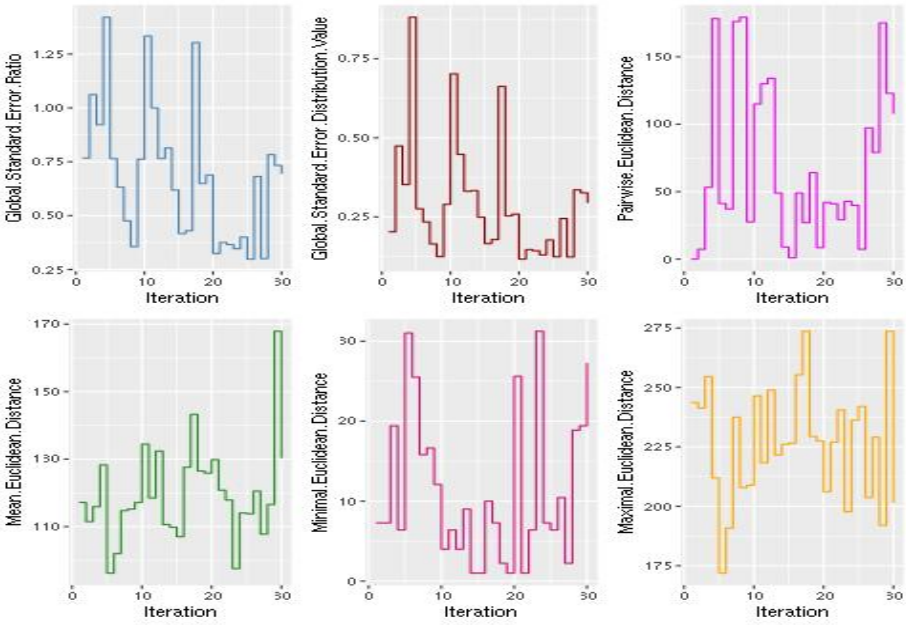

Figure 3: Global Explore-Exploit Measures (`focussearch`, `ei` and 30 Iterations) on material design data.

It proposes $\mathbf{x}^*$ according to

$$\mathbf{x}^* = \arg\min_{\mathbf{x} \in \mathcal{X}} \text{CB}(\mathbf{x}) = \arg\min_{\mathbf{x} \in \mathcal{X}} \hat{f}(\mathbf{x}) - \lambda \hat{s}(\mathbf{x}). \tag{4}$$

The bigger λ, the more variance reduction (i.e. exploration) is aimed at by the algorithm. We can verify our measures by applying them on several MBO runs with `cb` as infill criterion and varying $\lambda \in \{1, 3, 5, 10\}$, where other hyperparameters remain fixed (`focussearch`, local `scope` and 60 iterations).

Figures 4, 5, 6 and 7 show the results: Local Stand Error Ratio and Local Standard Error Quantile strongly increase with higher values of λ. As for the distance-based measures, this decrease is less clear, if at all visible. We again observe weak correlation between variance-based and distance-based measures and conclude that variance-based measures seem to better capture the explore-exploit trade-off than distance-based measures.

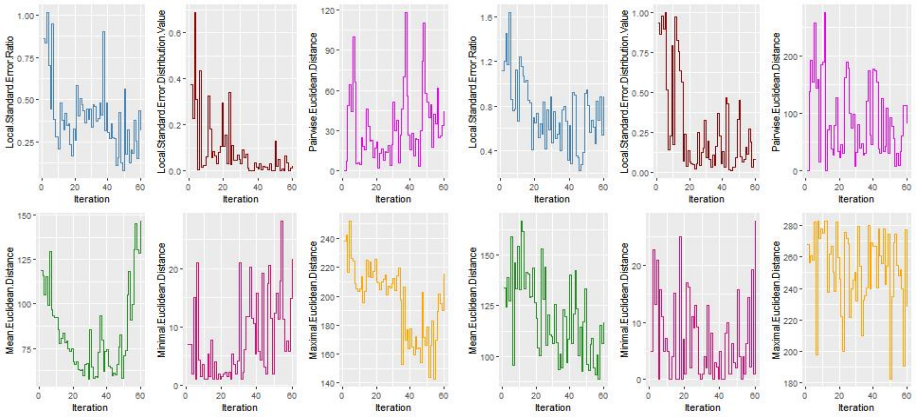

Figure 4: Local Explore-Exploit Measures (`focus search`, `cb` with $\lambda = 1$ and 60 Iterations).

Figure 5: Local Explore-Exploit Measures (`focus search`, `cb` with $\lambda = 3$ and 60 Iterations).

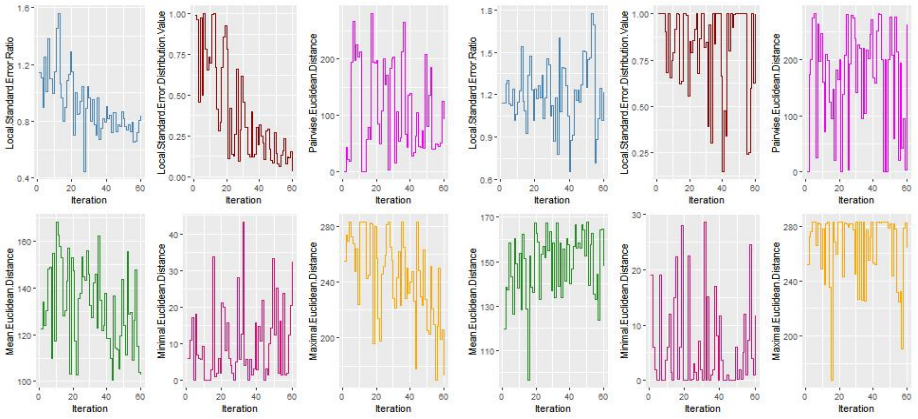

Figure 6: Local Explore-Exploit Measures (`focus search`, `cb` with $\lambda = 5$ and 60 Iterations).

Figure 7: Local Explore-Exploit Measures (`focus search`, `cb` with $\lambda = 10$ and 60 Iterations).

The two infill criteria Mean Response (`mr`) and Standard Error (`se`) represent extreme cases of `cb`. Mean Response only optimizes the mean prediction (equivalent to $\lambda = 0$), while Standard Error exclusively maximizes the predicted standard error (equivalent to $\lambda \to \infty$). Figures 8 and 9 show the results for `mr` and `se`. They confirm our previous analysis.

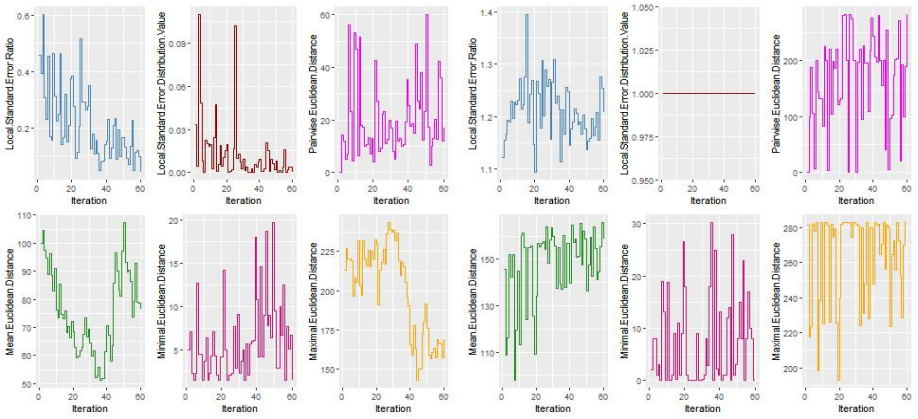

Figure 8: Explore-Exploit Measures for Mean Response (`mr`) infill criteria, `focus search`, and 60 Iterations.

Figure 9: Explore-Exploit Measures for Standard Error (`se`) infill criteria, `focus search`, and 60 Iterations.

4.3.2 Kapton

Since the kapton data contains a categorical variable (`gas`), we are restricted to infill optimization via `focussearch` and `computeXplxpl()` uses gower distance instead of euclidean distance. Figure 10 and 11 allow comparison of material design and kapton data. They show `xplxpl` results for both data sets using the exact same hyperparameters (`adacb`, `focus search`, 60 Iterations, $n = 20$).

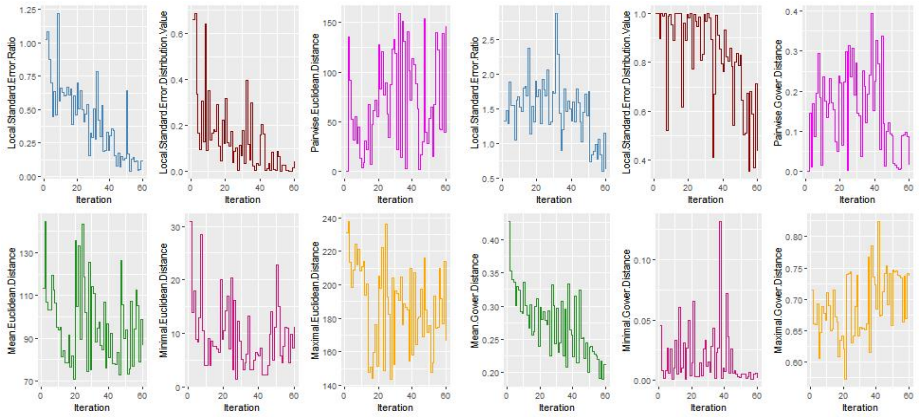

Figure 10: Explore-Exploit Measures for Material Design Data (`adacb`, `focus search`, 60 Iterations, $n = 20$).

Figure 11: Explore-Exploit Measures for Kapton Data (`adacb`, `focus search`, 60 Iterations, $n = 20$).

	Standard Error Ratio	Standard Error Distribution Value
Material Design	0.73	0.40
Kapton	1.51	0.82

Table 1: Mean values of explore exploit measures in provided data sets "Material Design" and "Kapton" with 60 iterations and initial data size $n = 20$.

The observable pattern is similar when using other infill criteria than `adacb`. When looking at the results, two things come to mind. First, the MBO tends to explore rather than exploit the kapton data compared to material design data. A quick comparison of explore exploit measures confirms this visual impression. We averaged over all applicable infill criteria and 60 iterations. Distance-based measures are incommensurable, as they rely on different metrics. Standard Error Ratio and Distribution, however, are shown in Table 1. One possible reason behind this difference in explore-exploit measures is the number of features. With a given size of the initial data, more features may provide more information on the functional relation. Hence, less exploration is required throughout the process. On the other hand, the curse of dimensionality [2] may make the initial data relatively sparse in higher-dimensional parameter space.

The second thing that occurs to the attentive observer is the degree of correlation between distance- and variance-based measures, which seems a little higher than in the material design case. Especially Mean Gower Distance seems to correlate with Standard Error Distribution Value in case of using `adacb`, see Figure 11. Again, a more detailed analysis confirms this presumption, see Tables 2 and 3 in the subsequent chapter. Reason???

4.3.3 Correlation of Measures

As indicated in previous sections, variance-based measures and distance-based measures appear to correlate only weakly, if at all. A detailed analysis confirms this suspicion, see Tables 2 and 3. There is correlation within variance-based and within distance-based measures

	Standard Error Ratio	Standard Error Distribution Value	Pairwise Distance	Minimal Distance	Maximal Distance	Mean Distance
Standard Error Ratio						
Standard Error Distribution Value	0.62					
Pairwise Distance	0.08	0.08				
Minimal Distance	-0.03	-0.00	0.28			
Maximal Distance	-0.03	0.12	0.13	0.02		
Mean Distance	0.11	0.28	0.23	0.18	0.59	

Table 2: Mean Correlations of Local Explore-Exploit Measures in 28 MBO runs with varying infill criteria and optimizers on Data Set "Material Science".

	Standard Error Ratio	Standard Error Distribution Value	Pairwise Distance	Minimal Distance	Maximal Distance	Mean Distance
Standard Error Ratio						
Standard Error Distribution Value	0.66					
Pairwise Distance	0.15	0.12				
Minimal Distance	-0.01	-0.05	0.15			
Maximal Distance	0.27	0.29	0.01	-0.37		
Mean Distance	0.31	0.30	0.21	-0.16	0.62	

Table 3: Mean Correlations of Local Explore-Exploit Measures in 14 MBO runs with varying infill criteria and optimizers on Data Set "Kapton".

As stated in the section on material design, altering λ in Confidence Bound (`cb`) and observing our measures suggests that variance-based measures capture the explore-exploit trade-off quite well, whilst distance-based measures do not. This explains the weak correlation and raises a new question: Why do distance-based measure say little about explore-exploit behaviour?

For illustration purposes, consider the Branin Function, which maps from $\mathbb{R}$ to $\mathbb{R}^2$ and has three global optima. We let `mbo()` optimize this function. Figure 12 shows the proposed points of each iteration. The MBO apparently succeeds at locating the global optima. Now focus on Iteration 18: The MBO "jumps" to another region of the parameter space, yielding relatively high distance-based measures (see Table 4). However, the MBO does not really explore in such iterations. It has visited the region before, so it does not aim at exploring previously unknown territory, but it *returns* to a well-known area and continues to exploit it. This behaviour of iteratively exploiting multiple regions is presumably common for MBOs applied on multimodal objective functions. In case of the Branin function, it can be observed in iterations 12, 15, 18 and 20, as can be seen in Table 4.

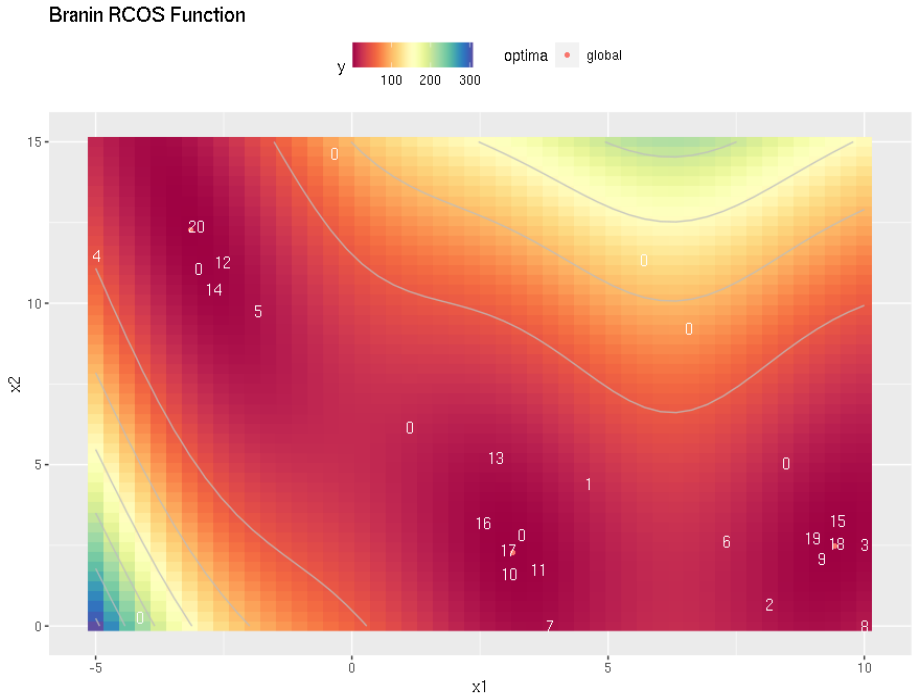

Figure 12: Optimization Path of MBO applied on Branin Function. Numbers represent iterations and indicate proposed points. Zeros represent initial data.

Standard Error Ratio	Standard Error Distribution Value	Pairwise Distance	Minimal Distance	Maximal Distance	Mean Distance	Iteration
0.49	0.15	11.34	0.51	16.84	9.35	12
0.87	0.51	8.08	1.93	10.01	5.90	13
0.49	0.31	7.59	0.69	16.42	8.63	14
0.68	0.45	14.13	0.71	16.64	7.66	15
0.70	0.40	6.93	0.86	11.80	6.00	16
0.29	0.28	0.98	0.55	12.74	5.89	17
0.29	0.24	6.42	0.50	16.97	7.37	18
0.36	0.25	0.50	0.50	16.48	6.82	19
0.88	0.50	15.40	1.21	17.96	10.21	20

Table 4: Explore Exploit Measures for selected Iterations of an MBO run on the Branin Function.

In light of these results, we recommend using variance-based measures as opposed to distance-based measures. They obviously better capture the explore-exploit trade-off. In addition, distance-based measures are vulnerable to the Curse of Dimensionality: The proposed points might be equally far away in very high-dimensional parameter spaces [2]. Besides, distance-based measures lack intuition, at least in mixed parameter spaces, where they deploy the gower-distance. Variance-based measures, on the contrary, provide an intuitive explanation of the explore-exploit consideration, as they incorporate the proposed point's ratio/position in the distribution of candidate points during infill optimization.

Why low correlation with min dist? Min distance actually should be a good measure... (cf. Branin example)

4.4 Multi-Point Proposals

To improve speed of the model-based optimization, `mlrMBO` allows for exploiting multicore infrastructures by proposing multiple points per iteration and thus parallelizing the optimization. Our project partner is planning to use multi-point proposal MBO soon. This is why we extend our `xplxpl`-framework to multi-point proposals. `mlrMBO` provides three distinct multi-point methods: Multi-Objective Infill (`moimbo`) Optimization [4] as well as a straightforward extension of Confidence Bound (`cb`), described on page 10 in [3], and the so-called Constant Liar (`cl`). We implement the explore-exploit functions for the latter method, because `cl` is recommended for real-values parameter spaces[7] as in the experimental set-up of our project partner [7]. Like in the single-point case, our implementation works for all applicable single-objective infill optimizers.

To get the main idea of Constant Liar, we refer the reader to [6] and page 4

[7] See Parallelization Tutorial by `mlr-org`.

in [4]. For interpreting the explore-exploit behaviour it is important to note that the algorithm can both explore and exploit within one iteration j by proposing m different points $\mathbf{x}^{(j+1)}, \ldots, \mathbf{x}^{(j+m)}$. We therefore restrain from averaging over them like in the multi-objective case and analyze each of them separately.

Although multiple surrogate models are fitted within one iteration, only the first one is stored in the resulting mbo object by `mbo()`. Hence, we again edit some functions[8] in `mlrMBO` so that all surrogate models are stored. Furthermore, we integrate our modified infill optimizers `infillOptFocusSavepts()`, `infillOptEASavepts()` and `infillOptcmaesSavepts()` into the multi-point framework by creating `proposePointsConstantLiarXplxpl()`. This ensures `xplxplMBO()` - from where all edited functions are sourced - can access all necessary information to compute the explore exploit measures for $J \cdot M$ proposed points, where J is the total number of iterations and M the amount of proposed points per iteration.

4.5 Multi-Objective Functions

We trivially extend our explore-exploit-measures to some cases of model-based multi-objective (MBMO) optimization. However, we forego a complete implementation, since most applications in material science as provided by our project partner are currently single-objective optimization problems. We rather show that a straight-forward extension is possible without much ado.

When more than one target is optimized, `mlrMBO` offers three different optimization methods: Direct-indicator based (`dib`), ParEGO (`parego`) and direct multicriteria optimization via model-based infill criteria through EMOA (`mspot`), see page 11 in [3] for more details. We implement `xplxpl()` and `xplxplMBO()` for direct indicator based methods with infill optimization via `focussearch`, `ea` and `cmaes`. tbd. debug cmaes for dim(x) = 1 For this purpose, we slightly adapt the above mentioned variance-based explore-exploit measures Standard Error Ratio (SER) and Standard Error Distribution Value (SED). We now have to average over the (multiple) standard errors $\hat{s}(\mathbf{x}^*)_j$ of the (single) proposed point $\mathbf{x}^*$ of all m objective functions, giving us:

$$SER_{multi} = \frac{\overline{\hat{s}(\mathbf{x}^*)}}{\frac{1}{n}\sum_{i=1}^{n} \hat{s}(\mathbf{x}_i)}, \quad SED_{multi} = F_{\hat{s}(\mathbf{x})}(\overline{\hat{s}(\mathbf{x}^*)}), \tag{5}$$

where

$$\overline{\hat{s}(\mathbf{x}^*)} = \frac{1}{m}\sum_{j=1}^{m} \hat{s}(\mathbf{x}^*)_j. \tag{6}$$

Note that we do not need to average over n and m simultaneouly via double summation, as one and the same point $\mathbf{x}^*$ is proposed for **all** objective functions within one iteration. Another approach would have been to calculate SER and

[8] `checkStuff()`, `extractInfoMBO()`, `makeMBOResult.OptState()`, `proposePointsByInfillOptimization()` and `getSupportedMultipointInfillOptFunctions()`

SED separately for each objective function. This would enable a more differentiated analysis of the explore-exploit-behaviour, but would fail to describe the process as a whole. We leave it to further projects. Within `xplxplMBO()` `xplxplSEMultiLocal()` and `xplxplSEMultiGlobal()` do the necessary computation.