CHAPTER 5

Computational Cost of Robustness

Solving a robust optimization problem entails re-formulating the original optimization problem into a robust counterpart, e.g., by taking an average of the function values over different perturbations to a specific input (Chapter 3) (Kruisselbrink, 2012). Consequently, solving the robust counterpart is much more costlier as opposed to the original optimization problem. We refer to this aspect as the "computational cost of robustness". The "computational cost of robustness" has been overlooked in the literature to the best of our knowledge. Such an extra cost brought by robust optimization might depend on the problem landscape, the dimensionality, the severity of the uncertainty, and the formulation of the robust counterpart itself. This chapter devotes to such an extra cost brought by robustness in practical scenarios, as it can hinder the aim of computational efficiency set by the designer (Ullah et al., 2022).

Some of the most important questions that we will attempt to answer in this chapter are:

1. How can we rank commonly employed robustness criteria with respect to the trade-off of the "computational cost of robustness" and the quality of the robust solutions?

2. Which robustness criteria are recommended to practitioners for practical scenarios, in order to find robust solutions in an efficient manner?

Note that these questions are based on the fourth foundational question introduced in the first chapter. We will attempt to answer these questions with the help of an empirical investigation, which will focus on a broad spectrum of test cases. The test cases concentrate on the variability in problem landscape, dimensionality, severity of the uncertainty, and robustness criteria, among others.

The five robustness criteria discussed in this chapter have already been defined in Chapter 3. These include “mini-max robustness”, “mini-max regret robustness”, “expectation-based robustness”, “dispersion-based robustness”, and “composite robustness” respectively.

5.1 Cost of Robustness

Solving a numerical black-box problem subject to uncertainty and noise is challenging due to several reasons. These reasons include high dimensionality, problem landscape, and robustness criterion among others. We have already seen the impact of high dimensionality and problem landscape in Chapters 3 and 4 (Ullah et al., 2019, 2020a, 2021). This Chapter concentrates on the choice of the robustness criterion, which can also have a significant impact in efficiently solving the problem.

The choice of the robustness criterion is important for two main reasons.

- The chosen robustness criterion can have a significant impact on the efficiency. This is due to the fact that obtaining a robust solution requires additional computational resources as opposed to a nominal one, since the optimizer has to take into account the impact of uncertainty and noise as well. This need for additional computational resources is based on the robustness criterion[1] chosen, e.g., RO based on the “mini-max” principle requires an internal optimization loop to compute the worst impact of the uncertainty, whereas RO based on the “expectation” of a noisy function requires computing an integral (Beyer and Sendhoff, 2007; Ullah et al., 2019).
- The chosen robustness criterion can also have a significant impact on the performance, i.e., optimality. Recall that the aim for robustness can be achieved from two different schools of thought, which are often conflicting: Performance/Quality and Robustness/Stability (Kruisselbrink, 2012) (cf. 3.1.1). The former measures the robustness of a solution from the perspective of the overall performance, i.e.,optimality, under the variation of the uncertain parameters of the solution. As opposed to that, the latter measures the

[1]An underlying assumption in this context is the non-existence of hard constraints on the choice of RF. In some practical scenarios of RO, this assumption is not valid. Note, however, that, there are plenty of situations where the assumption is valid, and the lack of information on the computational aspects of the RFs makes it harder for practitioners to choose a suitable robustness criterion.

robustness of a solution from the perspective of the minimal performance variation under the variation of the uncertain parameters of the solution. Consequently, when choosing a robustness criteria, it is important to know that we maybe compromising on optimality to ensure robustness/stability, e.g., in the case of "mini-max" robustness.

In this Chapter, we only focus on the aspect of computational efficiency, based on the choice of robustness criterion. This is due to the fact that the "computational cost of robustness" (CCoR), i.e., the need for additional computational resources to find the "robust" instead of a "nominal" solution, depends on the robustness criterion chosen. Consequently, it is desirable to evaluate and compare commonly-employed robustness criteria with regards to computational cost, where the computational cost is mainly based on the cpu time taken to find the robust solution.

5.2 Empirical Investigation

Our aim in this Chapter is to understand the impact of robustness criterion in RBO with regards to computational efficiency. Intuitively, robustness criterion can have a significant impact on the performance of the RBO algorithm, since steps 5 and 6 in Algorithm 1 (cf. 1) require much more computational resources as opposed to the nominal BO algorithm (Jones et al., 1998). This need for additional computational resources is based on the chosen robustness criterion. Through our empirical investigation, we aim to better understand this impact for each of the five robustness criteria discussed in the thesis. To make our setup comprehensive, we take into account the variability in external factors such as problem landscape, dimensionality, and the scale/severity of the uncertainty.

For our study, we select ten unconstrained, noiseless, single-objective optimization problems from BBOB (Hansen et al., 2021). The set of selected test functions is given as: $\mathscr{F} = \{f_2, f_3, f_7, f_9, f_{10}, f_{13}, f_{15}, f_{16}, f_{20}, f_{24}\}$. An important thing to note is that each of the test functions in $\mathscr{F}$ is subject to minimization, and is evaluated on three different settings of dimensionality as: $D = \{2, 5, 10\}$. Apart from the test functions and dimensionality, we also vary the uncertainty level based on two distinct settings as: $\mathscr{L} = \{0.05, 0.1\}$, which indicate the maximum % deviation in the nominal values of the search variables.

For the deterministic setting of the uncertainty: MMR and MMRR, the compact set U is defined as: $\mathrm{U} = [-(L \times R), (L \times R)]$, where $L \in \mathscr{L}$ denotes the choice of the uncertainty level, and R serves as the absolute range of the search variables. For the test functions in $\mathscr{F}$, the absolute range of the search variables is 10, since all test functions are defined from -5 to 5. For the probabilistic setting of the uncertainty: EBR, DBR and CR, the uncertainty is modeled according to a continuous uniform probability distribution: $\Delta_{\mathbf{x}} \sim \mathcal{U}(a, b)$, where the boundaries a and b are defined similar to the boundaries of the the set U in the deterministic case.

In our study, the size of the initial training data is set to be $2 \times D$, where $D \in \mathscr{D}$ denotes the corresponding setting of the dimensionality. Likewise, the maximum number of iterations for the BO algorithm is set to be $50 \times D$. Note that our Kriging surrogate is based on the popular Matérn 3/2 kernel (Rasmussen and Williams, 2006), and we standardize the function responses: $\mathbf{y} = [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_N)]^\top$, before constructing the Kriging surrogate. In addition, we utilize the robust EI (cf. Eq. (4.13)) as the sampling infill criterion for our experiments.

For the parallel execution of the BO algorithm for each of the 300 test cases considered, we utilize the DAS-5 supercomputer (Bal et al., 2016), similar to the experimental setup discussed in the previous chapter. The performance assessment of the solutions in our experiments is based on 15 independent runs $\mathscr{R}$ of the RBO algorithm for each of the 300 test cases considered. Note that for each trial, i.e., the unique combination of the independent run and the test case, we ensure the same configuration of hardware and software to account for fairness. Furthermore, in each trial, we measure the cpu time for all iterations of the RBO algorithm.

After the successful parallel execution of all trials, we evaluate the performance of our robust solutions based on the quality difference $\mathcal{DQ}$ from the baseline (cf. Eq. (3.12))). Note that $\mathcal{DQ}$ in this case is based on the space of objective function values[1]. After this, we perform six different analyses to answer the questions outlined earlier. The first two type of analyses are referred to as the fixed cpu time analysis, and the fixed iteration analysis respectively. In fixed cpu time analysis, we fix 50 different settings of the cpu time, and report the best $\mathcal{DQ}$ (the lowest) for each trial. The $\mathcal{DQ}$ in this context is averaged over all 50 settings of the cpu time. For fixed iteration analysis, we fix 30 different settings of the

[1]In this study, we do not divide $\mathcal{DQ}$ with the number of independent runs $\mathscr{R}$ as Eq. (3.12) suggests, but rather report all trials.

iterations (checkpoints) to report the best $\mathcal{DQ}$ (the lowest) for each trial. The $\mathcal{DQ}$ in this context is also averaged over all 30 checkpoints.

After fixed cpu time and fixed iteration analysis, we perform a fixed target analysis. The fixed target analysis is also based on two different settings: by fixing a target $\mathcal{DQ}$ and reporting the cpu time as well as the number of iterations taken to reach that target. We fix ten different settings for the target in this context, and the corresponding cpu time and iterations are averaged over these target values. Note that each target describes the minimum desirable quality threshold of the robust solution. If such a quality is never achieved, we report the penalized cpu time and penalized number of iterations respectively. The penalized cpu time is set to be $D \times T_{\max}$, whereas penalized number of iterations is set to be $D \times N_{\max}$. Here D is the corresponding setting of the dimensionality, and $N_{\max}$ and $T_{\max}$ indicate the maximum number of iterations of the BO algorithm and the cpu time taken to execute it. After the fixed budget and fixed target analyses, we also report the average cpu time per iteration for the BO algorithm. In addition, we also report $T_{\max}$, the accumulated cpu time at the last iteration of the BO algorithm, for each trial.

5.2.1 Results

We share the results originating from our study in Figs. 5.1 – 5.5. Each of these figures contains the graphs for a particular type of analysis. In particular, Fig. 5.1 shares the results based on fixed cpu time analysis. The figure contains six different plots corresponding to two noise levels, and three different settings of the dimensionality. Each plot shares the empirical cumulative distribution function (ecdf) of $\mathcal{DQ}$ for five different robustness criteria considered. Note that each ecdf curve (for each robustness criterion in a plot) is based on 150 data points, owing to the combination of ten test functions and fifteen independent runs of the algorithm.

Likewise, Fig. 5.2 shares the ecdf plots corresponding to fixed iteration analysis, whereas the analyses based on fixed target is presented in Figs. 5.3. The average cpu time per iteration of the BO algorithm to find robust solutions is presented in Fig. 5.4. Lastly, we present the maximum accumulated cpu time: $T_{\max}$, for each trial in the form of box plots in Fig. 5.5.

In the following, we discuss the major findings of these results.

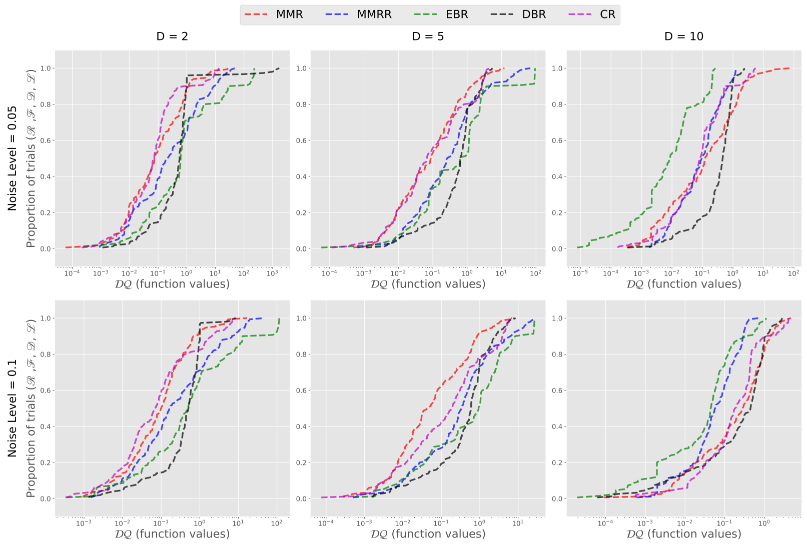

Figure 5.1: Fixed cpu time analysis. Rows distinguish between noise levels, whereas columns identify different settings of dimensionality. Each plot contains five Ecdf curves based on five robustness criteria discussed.

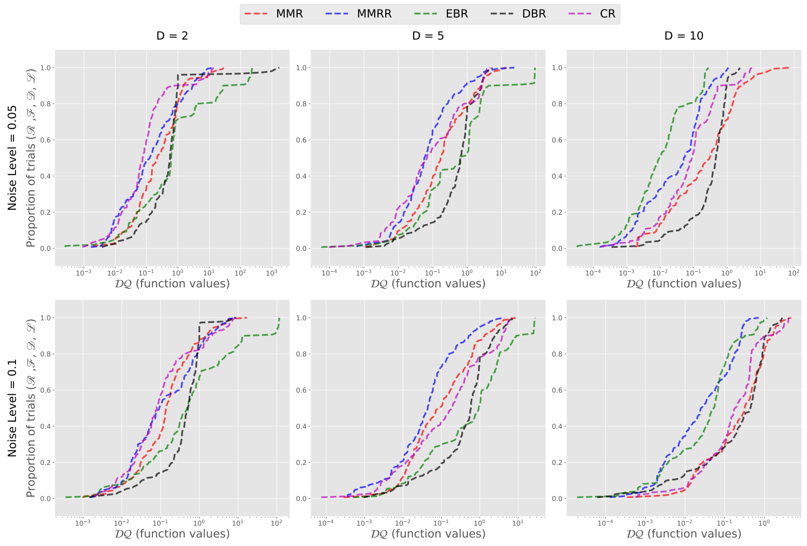

Figure 5.2: Fixed iteration analysis. Rows distinguish between noise levels, whereas columns identify different settings of dimensionality. Each plot contains five Ecdf curves based on five robustness criteria discussed.

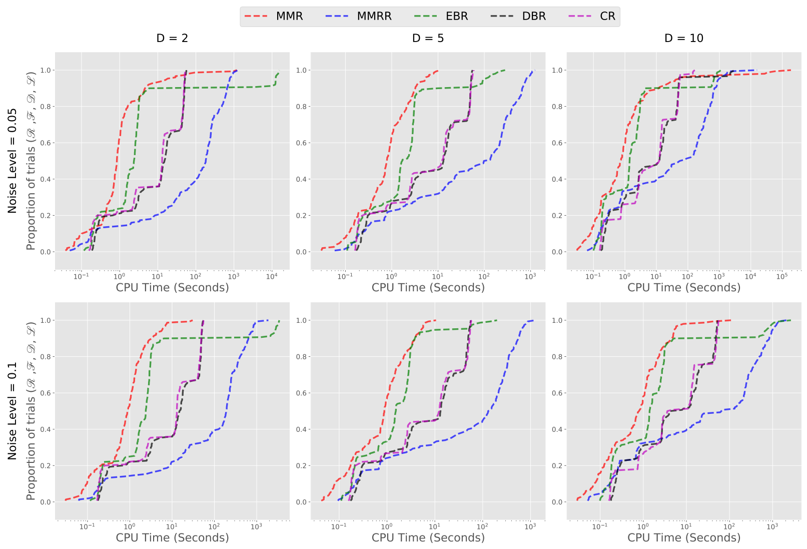

Figure 5.3: Fixed target analysis based on cpu time. Rows distinguish between noise levels, whereas columns identify different settings of dimensionality. Each plot contains five Ecdf curves based on five robustness criteria discussed.

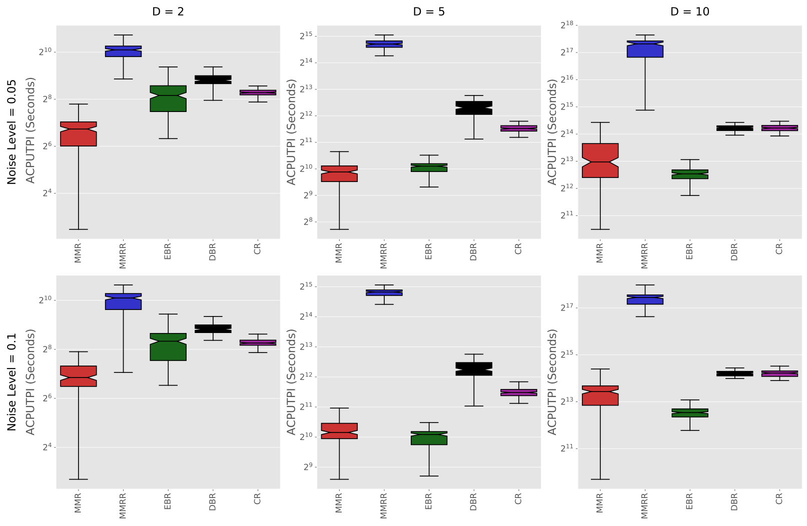

Figure 5.4: Average cpu time per iteration for the BO algorithm. Rows distinguish between noise levels, whereas columnsidentify different settings of dimensionality.

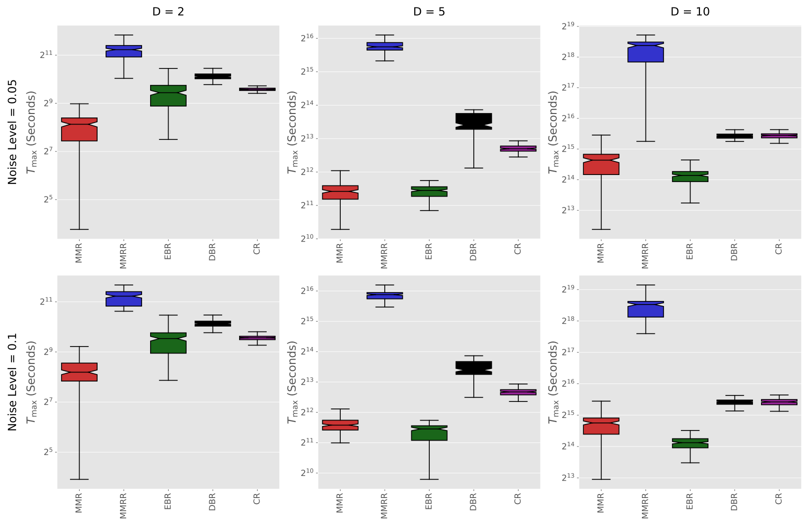

Figure 5.5: Maximum accumulated cpu time $T_{\max}$ for each trial. Rows distinguish between noise levels, whereas columns identify different settings of dimensionality.

In terms of performance comparison with respect to the fixed cpu time analysis, we note the promising nature of all RFs except DBR, which performs poorly compared to its competitors in most trials. Furthermore, we also note the highest variation in quality ($\mathcal{DQ}$) for DBR. Although no RF is deemed a clear winner for this analysis, we note that MMR, MMRR and CR have high empirical success rates. Likewise, we note the highest variation in quality for DBR also in the context of fixed iteration analysis. In this case, MMRR and CR perform superior to the other RFs as we observe a high empirical success rate for both. For the performance measure with respect to the fixed target analysis, we observe that MMR outperforms the competitors, albeit the variation in the running cpu time for MMR is also deemed higher. Here, we note a clear distinction in the empirical success rate between MMRR and other RFs. For instance, if we cut-off the running cpu time at 100 seconds, we observe that MMRR has an empirical success rate of around 40 %, whereas MMR, DBR, and CR achieve a success rate of more than 80 %.

In terms of average cpu time per iteration (ACPUTPI), we note that MMR and EBR perform excellently in most cases, whereas MMRR has the worst performance. In this case, none of the MMR and EBR can be deemed a clear winner, although both perform superior to other RFs in most trials.

When comparing the performance of RFs in the context of maximum cpu time spent: $T_{\max}$, we note that MMR and EBR in general perform superior to other RFs, whereas MMRR performs the worst for each setting of dimensionality. Furthermore, we deem that $T_{\max}$ increases rapidly with respect to dimensionality in the context of deterministic uncertainty, i.e., MMR and MMRR, when compared with the probabilistic uncertainty, i.e., EBR, DBR, and CR. Lastly, we note that in general, the variance in $T_{\max}$ for the probabilistic setting is also significantly lower when compared to the deterministic case.

5.2.2 Analysis

Based on the observations from the fixed budget analyses – fixed cpu time and fixed iteration analyses, we deem MMR, MMRR, EBR and CR to be suitable RFs with regards to the computational cost involved to find the robust solution. This validates their applicability in practical scenarios where the computational resources are limited, and the designer cannot spend more than a fixed amount of computational budget (whether measured in terms of cpu time or the number of iterations). Note that MMR appears to be the most promising RF also in the scenarios where the designer aims for a fixed quality solution – where the designer cannot compromise on the quality below a certain threshold. In those situations, MMR can yield the desired quality robust solution with considerably less cpu time. Apart from these analyses, we note that the ACTPI and $T_{\max}$ for MMR are also excellent alongside EBR.

In terms of performance, we find that MMRR poses an interesting situation as it performs competitively in the context of fixed budget analyses. However, its performance is significantly worse to other RFs in the context of fixed target analysis, the ACTPI, and the maximum cpu time $T_{\max}$ for running KB-RO. We believe this is aligned with the intuition of MMRR (as discussed in Chapter 3), since within an iterative optimization framework, we have to employ a quadrupled nested loop to find the robust solution based on MMRR, which in turn exponentially increases the computational cost per iteration. The MMRR, therefore, has the highest CCoR, and takes much more cpu time to reach the same target value as opposed to other RFs.

In terms of performance variance, we note that stochastic RFs, in particular EBR and DBR, have a higher variance in quality – when measured in terms of NMAE, and a comparatively lower variance in computational cost – when measured in terms of the ACTPI and $T_{\max}$. This can mainly be attributed to their intrinsic stochastic nature as they rely on numerical approximations. Since the sample size of the numerical approximations is fixed with respect to the corresponding setting of the dimensionality, we observe relatively lower variance in the cpu time. However, since we only ever approximate the robust response of the function, the quality of the solution may be deteriorated.

5.3 Summary and Discussion

This chapter analyzes the computational cost of robustness in Bayesian optimization for five of the most commonly employed robustness criteria. In a first approach of such kind, we attempt to evaluate and compare the robustness formulations with regards to the associated computational cost, where the computational cost is based on the cpu time taken to find the optimal solution under uncertainty. Our experimental setup constitutes 300 test cases, which are evaluated for 15 independent runs of Bayesian optimization.

A fixed budget analysis on our experimental results suggests the applicability of “mini-max robustness”, “mini-max regret robustness”, “expectation-based robustness”, and “composite robustness” in practical scenarios where the designer cannot afford the computational budget beyond a certain threshold. On the other hand, a fixed target analysis deems the ‘mini-max robustness” as the most efficient robustness criterion in the scenario where the designer cannot compromise the quality of the optimal solution below a certain threshold. The analysis on the average cpu time per iteration and maximum cpu timer per run also determines “mini-max robustness” as one of the most efficient robustness criteria. Overall, “mini-max robustness” is understood to be the most suitable robustness criterion with regards to the associated computational cost.

A limitation of our study is that we fix the internal computational budget for each robustness formulation in Kriging-based robust optimization. Visualizing the impact of variability in the internal computational budget, e.g., the internal optimization loop in the context of “mini-max robustness”, is also an important thing to do. Additionally, we note that each robustness formulation is intrinsically associated with another cost, namely the cost of compromising on optimality to ensure robustness or stability (cf. 5.1). Focusing on this cost of robustness will advance the state-of-the-art in this area, and help practitioners choose the most suitable formulation with regards to optimality.