Supplementary Material for Searching in the Forest for Local Bayesian Optimization

Difan Deng DENG@TNT.UNI-HANNOVER.DE
Marius Lindauer LINDAUER@TNT.UNI-HANNOVER.DE
Leibniz University Hannover

Editors: P. Brazdil, J. N. van Rijn, H. Gouk and F. Mohr

Appendix A. Algorithms to Extract Subregions

Algorithm 1: Subregion Selection with Random Forest

Input: Search Space $\mathcal{X} \in \mathbb{R}^n$; candidate selected by global Bayesian Optimization $\boldsymbol{x}_g$, random forest model $\hat{f}$ with n_{tree} trees; minimal number of points stored in the subregion n_{min}
Output: subregion $\mathcal{X}_{sub}$ extracted from $\mathcal{X}$
Initialization: $\mathcal{D}_{sub} \leftarrow \mathcal{X}$; a list of root nodes S of RF $\hat{f}$, a nodes indicator $I \leftarrow [False] * n_{trees}$ indicating that if we will stop deeper from the node.
while no element in I is False **do**
  for each node $s_p \in S$ **do**
    $s'_p \leftarrow child(s_p)$ with $\boldsymbol{x}_g \in s'_p$
    Let $\mathcal{D}_{s'_p}$ be all observed points in s'_p
    if $|\mathcal{D}_{sub} \cap \mathcal{D}_{s'_p}| > n_{min}$ **then**
      $s \leftarrow s'$
      $\mathcal{D}_{sub} \leftarrow \mathcal{D}_{sub} \cap \mathcal{D}_{s'}$
    else
      $I_p \leftarrow True$
    end if
  end for
end while
Return: An extracted subregion $\mathcal{X}_{sub}$

Algorithm 1 describes how a subregion is extracted with the help of a RF. Here we illustrate a minimal toy example in Figure 1. We train an RF with two trees which split the space accordingly. Suppose that at least $n_{min} = 3$ points are required to be included in the subregion and the subregion grows from $\boldsymbol{x}_g$ (marked in red). We note that, here $\boldsymbol{x}_g$ is not an actual sampled point, but it only indicates the position that the subregion should contain. We first step from a_0 to a_2 as it contains $\boldsymbol{x}_g$ and hence D, E and F are excluded from the subspace. Then we split the subspace with the split of b_0, where all the points are preserved. We further go deeper into the first tree and arrive at node a_3, in which case only 3 points lay in the subregion and we stop exploiting tree A. However, for tree B, as each split will not further shrink the subregion or exclude the existing points from the subregion, we arrive at leaf node b_5 and stop further shrinking the subregion.

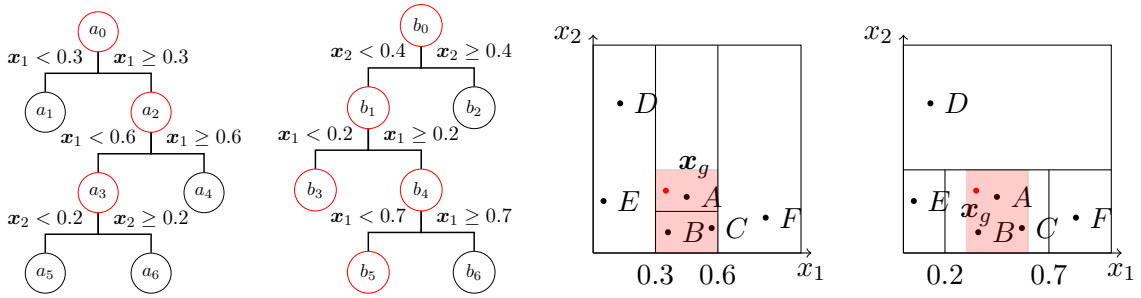

Figure 1: Subregion selection. The final extracted subregion is illustrated in the red-shaded region.

Appendix B. Implementation Details

We start with a Gaussian Process that is trained on all the previous evaluated data points until n_{min} points exist in the dataset. Then we start to extract a subregion, as described in Algorithm 1. It makes no sense to introduce LGPGA if the number of $\mathbf{f_o}$ is smaller than $n_{\mathbf{u}}$. In such a case, we still train a full GP model on all the previously evaluated points but only optimize the acquisition function values in the subregion. If we have more observations, we apply LGPGA to reduce the potential computation time.

B.1. LGPGA Details

Two sets of LGPGA hyperparameters need to be optimized: the kernel hyperparameters and the position of inducing points. Comparing with inference time, we optimize these two sets of hyperparameters in an inverse order: we first train a GP model to fit $\mathbf{f_i}$ to get the optimized kernel hyperparameters; thus LGPGA captures the local data distribution inside the subregion. We then train a sparse GP to fit $\mathbf{f_o}$. Specifically, we first train a vanilla GP model to fit $\mathbf{f_i}$ to acquire the kernel hyperparameters. Then we use this optimized kernel to initialize a sparse GP and keep its kernel hyperparameters fixed, i.e. we approximate $\mathbf{f_o}$ by only optimizing the position of the inducing points. The hyperparameters of LGPGA thus captures both $\mathbf{f_i}$ (the kernel hyperparameters) and $\mathbf{f_o}$ (the inducing points positions). We apply variational GP Titsias (2009); Hensman et al. (2013) to approximate $\mathbf{f_o}$ with its hyperparameters optimized with natural gradient descend Salimans et al. (2017). We train the sparse GP to approximate the predictive variational evidence lower bound (ELBO), as proposed in Jankowiak et al. (2020). A further advantage of a variational GP is that it allows stochastic variational inference (SVI), i.e. we could scale variational GP to even larger dataset as the number of evaluations grows. However, in this paper, we only consider batch optimization.

We set number of inducing points to be at least $min(2 \times n_{dims}, 10)$ and it grows linearly as number of total evaluations grows ($min(\frac{n_{\mathcal{D}}}{20})$ where $\mathcal{D}$ denote all the previous evaluations) and up to 50 points.

B.2. BOinG+ Details

When working with larger budgets (e.g. larger than 500 evaluations), as Random Forest might converge to a local minimum, we combine TuRBO Eriksson et al. (2019) and BOinG in the following ways: we start with BOinG and set a failure counter c_{fail}. We increase it every n_{dim} times when we have not found a better configuration or decrease it if a better configuration was found. The probability of switching to TuRBO is computed as:

$$p_{switch} = 0.1 * c_{fail} \tag{1}$$

As BOinG potentially acts as an exploitation mechanism, we will let TuRBO focus more on exploration: we randomly sample 20 configurations $\boldsymbol{X}_{random} \in \mathcal{X}$ and extract their subregions accordingly with Algorithm 1. We take the subspace $\mathcal{X}_{sub}$ with the largest volume and build a TuRBO optimizer inside $\mathcal{X}_{sub}$. The TuRBO optimizer is initialized with the points inside this subspace. Eriksson et al. (2019) restarts TuRBO if the length of the subregion is smaller than 2^{-7}. We restart TuRBO if the length of the subregion is smaller than 2^{-4} instead to allow more repetitions. Similarly, we adjust the probability of switching to BOinG with Equation 1. However, if TuRBO finds a better configuration, we switch back directly to BOinG for further exploitation. Each time when we switch between BOinG and TuRBO, we halve their failure counts c_{fail} accordingly to avoid too frequent switching.

Appendix C. Experiments Details

All the GP-related models use Mátern 5/2-kernel. We set the number of inducing points $n_{\mathbf{u}} := \min(10, 2 \cdot d)$. Both BOinG and full GP are implemented with GPyTorch v.1.2.1 Gardner et al. (2018); Balandat et al. (2020). RF and acquisition function optimizers of the above-mentioned BO models are implemented in the framework of SMAC3 v1.0.1 Hutter et al. (2012); Lindauer et al. (2022)[1], where a combination of random and local search for the optimization of the acquisition function is implemented.

We ran all methods multiple times with different random seeds on 4 Intel Xeon E5 cores running openSUSE Leap 15.1. [2]

For Adult and Cartpole problems, we allow $2 \cdot n_{dims}$ initial points and they are initialized with a Sobol sequence, except for TuRBO and LA-MCTS. The deterministic Sobol sequence is not applicable to TuRBO and LA-MCTS since they require the randomness by initialization to restart with different points. All the optimizer runs are repeated 30 times.

The PPO agent in the CartPole benchmark is implemented by Tensorforce Kuhnle et al. (2017) and the environment is implemented in OpenAI gym Brockman et al. (2016). For this benchmark, we optimized the reward value achieved by the agent after a maximum of 200 episodes or 6000 steps—whichever was reached first. The final performance is the mean reward of 20 episodes by the trained agent. To reduce the impact of noise, for each hyperparameter configuration, we repeatedly evaluate 9 runs and return the mean value of the final cost value.

Lunar Lander and Robot Pushing tasks are applied in Eriksson et al. (2019). We set the same search space as TuRBO did[3] due to the large amount of time required for each

1. https://github.com/automl/SMAC3
2. To ensure reproducibility, our code is publicly available at https://github.com/automl/SMAC3
3. https://github.com/uber-research/TuRBO

evaluation. We only run 20 repetitions for these two benchmarks. For the lunar lander problem, we allocate a budget of 1500 function evaluations and for robot pushing, a budget of 3000 function evaluations. .

Appendix D. Ablation Study

Given the various decision choices and additional hyperparameters introduced by BOinG, here we evaluate how different design choices affect the performance of BOinG.

D.1. LGPGA vs. GP

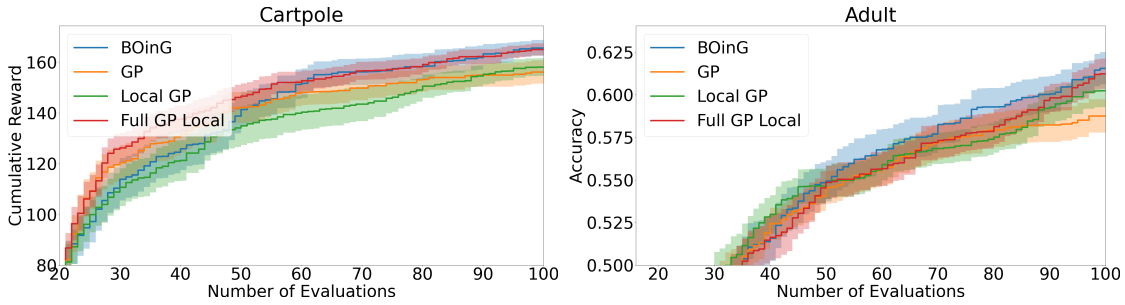

Figure 2: Ablation Study on different choices of GPs on (left) the cartpole benchmark and (right) the adult benchmark.

We first study the extra benefit that a LGPGA brings to BOinG. In addition to LGPGA, we train a GP model only with the evaluations inside the subregion (*local GP*) and another GP model that is trained with all previous evaluations but the candidates are restrained to be selected from the subregion(*full GP local*). We evaluate the local GP models on the cartpole problem.

The result is shown in Figure 2. Both LGPGA and *full GP local* has better final performances compared to a *local GP*. However, consider the potential benefit that a LGPGA can bring (it requires less resources), we still suggest to use LGPGA in BOinG.

D.2. Number of points inside subregion

BoinG introduces a special hyperparameter: n_{min}, the number of points inside the subregion. n_{min} determines the size of the subregion to be explored in the next stage. Setting this value larger (e.g. to infinity) makes BOinG closer to a GP and thus requires more resources; reducing this value (e.g. until 0) leads BOinG to behave as an RF. As RF is often considered as a poor extrapolator, a BOinG with a small subregion might easily fall into a local minimum. Here we do an ablation study on n_{min}: *BOinG_i_in* denotes a BOinG Optimizer that contains at least $i \cdot d$ points inside its subregion.

The result is shown in Figure 3. For the Adult task, all the BOinG-variation has similar performance. However, for the cartpole task, *BOinG_3_in* achieves a better reward in the beginning but is then surpassed by *BOinG_5_in*; *BOinG_7_in* only starts to outperform a

vanilla GP after ≈ 60 evaluations, which might be too late for a problem with 100-evaluation. We consider $5 \cdot d$ as a plausible number of points kept in the subregion.

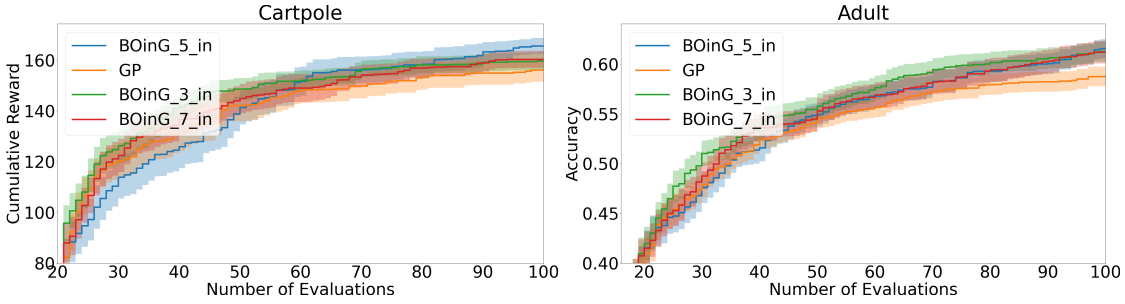

Figure 3: Ablation Study on n_{min} (Bottom) on (left) cartpole benchmark and (right) the adult benchmark.

D.3. The choice of acquisition function at the global level

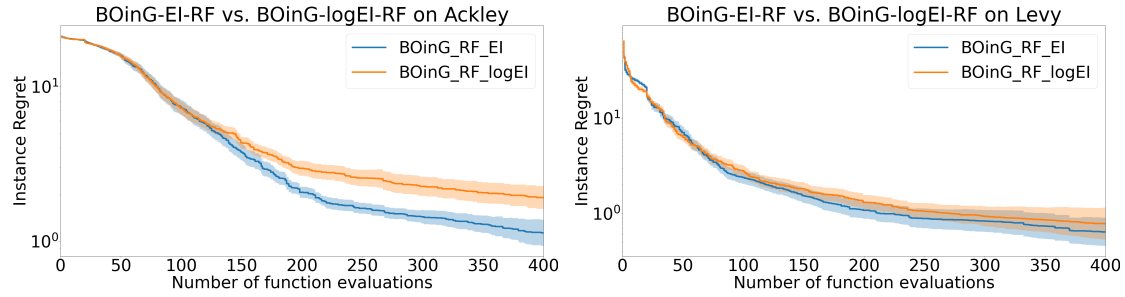

Figure 4: Ablation Study on the choice of the acquisition function at the global level on ackley (Left) and levy (right) function.

Instead of the default setting of logEI Hutter et al. (2012), we apply vanilla EI as the acquisition function at the global level on synthetic functions. These synthetic functions usually do not fit the requirement of "heavy-tailed cost distributions" Lindauer et al. (2022) and possess several local minima that an optimizer might easily fall into. Applying EI allows the optimizer to explore the unknown regions more and thus provides the optimizer a higher probability to find the global optimum. The result is illustrated in Figure 4. BOinG equipped with EI as the acquisition function at the global level generally has a better any time performance, especially on the Ackley benchmark. However, Levy has a large flat region near the optimum hence the benefit of replacing LogEI with EI is not so significant.

D.4. BOinG+ vs. vanilla BOinG

In Section 3.3, we proposed to randomly switch between BOinG and TuRBO depending on their results accordingly. Here we will study if BOinG+ achieves a better exploitation-

exploration tradeoff than TuRBO. As we restart TuRBO earlier to allow more exploration, we first check if this strategy helps TuRBO to find a better configuration. According to Section 3.3, we restart TuRBO if the length of the subregion is smaller than 2^{-4}, called *TuRBO_4*. Additionally, we will study the extra benefit that BOinG+ brings to BOinG and TuRBO, we compare it with vanilla BOinG where BOinG never switches to TuRBO (*BOinG_Vanilla*) and a BOinG+ version where TuRBO never switches to BOinG (*RF_TuRBO*). The results are shown in Figure 5. Vanilla BOinG underperforms compared to the other variants. TuRBO_4 becomes more explorative in the very beginning but cannot dig deeper as it restarts too early and is outperformed by RF_TuRBO as it cannot capture the global data distribution. BOiNG+ instead allows further exploitation and finds the best configuration in the end.

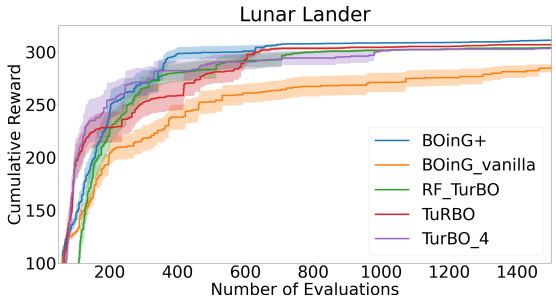

Figure 5: Ablation Study on BOinG+.

Appendix E. How LGPGA Guides the Search in Subspaces

To illustrate how LGPGA incorporates the information outside the subregion, we present a toy example in Figure 6 to show how LGPGA handles heteroscedastic noise. The data is generated according to Yuan and Wahba (2004) and follows a normal distribution with mean: $\mu(x) = 2(\exp(-30(x-1/4)^2)) + \sin(\pi x^2)$ and variance $\sigma^2(x) = \exp(2\sin(2\pi x))$. This distribution has low noise level with larger x values and high noise level with smaller x values. Here we only use our model to fit the distribution of the right side. We randomly sample 50 points from $[0, 1]$ and select the points indices from 35 to 45 as the points inside the subregion. The predicted mean and variances are illustrated in Figure 6. Fitting a GP on the entire data distribution (full GP) will not exactly describe the noise on the right part of the data distribution. The GP fitting only the data points inside the subregion (Local GP) and our LGPGA describe better the heteroscedastic noise inside the subregion.

Additionally, we illustrate how LGPGA influences the acquisition function value landscape and guides the search.

We train full GP, local GP and LGPGA on the same data distribution. The subregion is bounded by the rectangle. Without knowing the information of the points outside the subregion, the local GP will have a large chance to sample the points on the top right of the subregion for exploration. However, the high loss of the samples on the top right indicates that it might not be a good choice to sample a new point in this direction, as illustrated by the acquisition value loss landscape of the full GP model. However, LGPGA optimizes its inducing points to approximate the distribution outside the subregion and we see that two inducing points are located on the bottom left of the subregion while another two lay on the

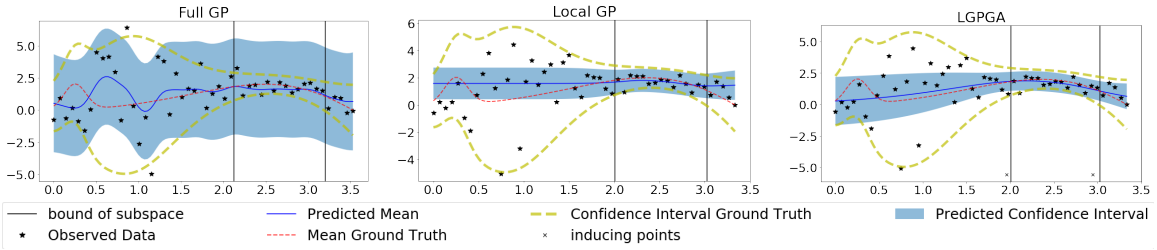

Figure 6: Predicted mean and variance of a **Left**: full GP **Middle**: local GP **Right**: LGPGA. A LGPGA is utilized to fit the data points inside the subregion and still model the overall data trend also at the subregion's boundaries. We note that the inducing points only contain the positional information $\mathcal{D}$. We put them at the bottom of the figure to to distinguish them from the training points.

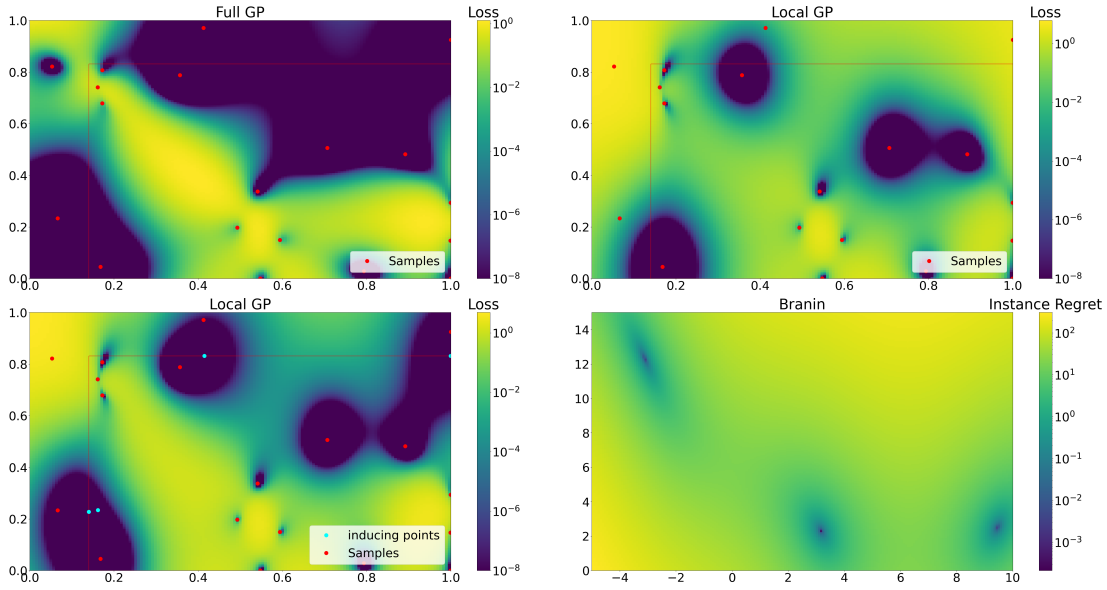

Figure 7: Predicted EI value of different GP models, red rectangles are the boundary of the subregion. From top left to bottom right: a GP model that is trained on all the previous evaluations; a GP model that is trained to only fit the distribution inside the subregion (the red rectangle); a LGPGA model trained with all the previous evaluations; the loss landscape of branin function.

top side. Thus we could avoid unnecessary exploration on the bottom left and focus more on the region near the optimum or the direction that is still not fully explored.

Appendix F. Scalability

One drawback of a Gaussian Process is its cubic complexity with respect to the number of the points ever evaluated. Similar to other local BO approaches Eriksson et al. (2019); Wang et al. (2018, 2020), our approach alleviates this problem by developing a model only with a subset of the previous evaluations. However, a local GP still scales cubically w.r.t. the number of points inside the subregion (and does not capture the overall trend). Hence the complexity of our algorithms mainly depends on the number of points inside the subregion.

F.1. Complexity of LGPGA

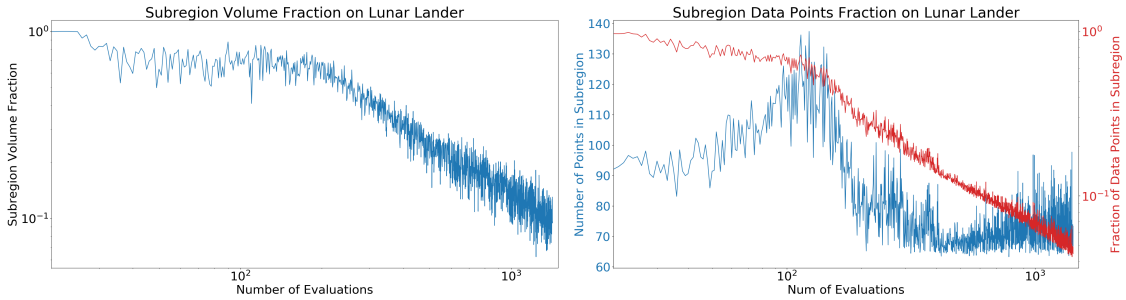

Figure 8: Left: fraction of subregion volume versus the entire region. Right: number of points inside subregion versus number of evaluated points.

Assuming that the size of $\mathbf{f_o}$, $\mathbf{f_i}$ and u are n_g, n_l and m respectively, the complexity of fitting a model in the first and second stages are $\mathcal{O}(m^2 n_g)$ and $\mathcal{O}((m+n_l)^3)$ respectively. Predicting the mean and variance takes the complexity of $\mathcal{O}(m+n_g)$ and $\mathcal{O}((m+n_l)^2)$. Oppositely, if we consider $\mathbf{f_i}$ as a subset of $\mathbf{u}$, then the complexity of fitting the models increases to $(O)(m+n_l)^2 n_g$, while the complexity for predicting the mean and variance stay the same.

Normally we have $m \ll n_g \ll n_l$, although the posterior distribution needs to be computed twice, we can still save a lot of resources by introducing LGPGA without loss of precision. Additionally, a sparse GP can only be trained with $\mathbf{f_i}$ and $\mathbf{f_o}$ jointly, which might underfit $\mathbf{f_i}$. Later we will show how different components of LGPGA emphasise different parts of training data.

Figure 8 illustrates how the number of data points varies as the number of evaluations grows. Here we take the evaluation result on the lunar lander problem as an example, where $n_{dims} = 14$ and thus the minimal number of points inside the subregion is 70. We could see that the number of points inside the subregion stays nearly constant as the number of evaluations grows. The subregion quickly shrinks after about 100 evaluations and hence our local model could focus more on the most promising region. However, this trend does not hold for the following evaluations. The subregion only shrinks smoothly in the following evaluations, which will prevent BOinG from converging to a local minimum too early.

Figure 9 illustrates the time spent for a BOinG iteration as the number of evaluations grows. Starting from the 200th evaluations on ackley 10D, BOinG takes less time in each

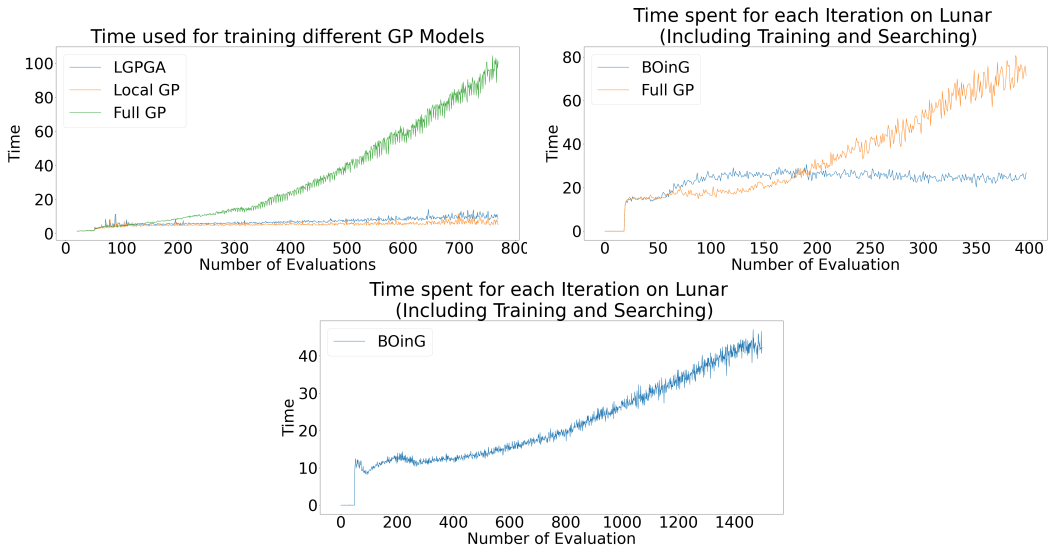

Figure 9: Scalability Analysis, we note that we optimize GP's hyerparameters with L-BFGS Liu and Nocedal (1989) optimizer and repeat optimization for 10 repetitions with different initial points on Lunar Lander (14D). **Upper Left**: time spent for training different GP models. **Upper Middle**: Time Spent by BOinG and Full GP for each BO iterations on Ackley (10D). **Bottom**: Time Spent for each BO iterations on Lunar Lander (14D).

iteration compared to a full GP. From the bottom plot in Figure 9, we could see that BOinG scales nearly linearly as the number of evaluations grows until up to 1500 evaluations.

Next we evaluate the saved resources and the additional overload that LGPGA brings to global and local GP models. Again, we train 3 different models on the data that we obtained when we optimize the hyperparameters on lunar task: full GP, local GP and LGPGA. Here, the GP's hyperparameters are optimized with 10 repetitions. The result is illustrated in the top part of Figure 9. Compared to a full GP model, LGPGA requires much less resources and scales nearly linearly as the number of the previous evaluations grows.

Appendix G. Will BOinG+ Give Better Suggestions compared to TuRBO?

BOinG+ switches between TuRBO and BOinG randomly according to their failure counts. In the ablation study, we show that BOinG+ has a better final performance compared to different variation of TuRBO. However, it is still unclear where the incumbent configuration comes from, i.e., if BOinG+ works as we expect: explore with TuRBO and exploit with BOinG? To answer this question, we show the fraction of the incumbents' origin as the number of evaluations grows on the two tasks where BOinG+ is applied.

The results are shown in Figure 10. Depending on the task, the performances are quite different. This shows that BOinG+ could adjust to different sorts of landscapes and not get stuck at one single optimizer. The share of TuRBO reaches peak at roughly 500 evaluations

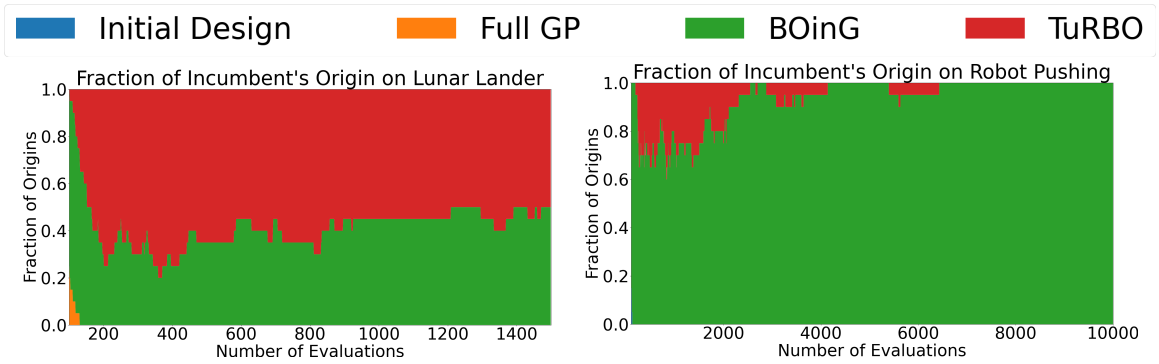

Figure 10: Share of different incumbents' origins

and then more incumbents are suggested by BOinG. Thus, as expected, TuRBO explores more in the mid term of the optimization process and thus finds more incumbents during this period; while BOinG exploits more in the most promising regions and thus gives more incumbents at the end.

References

Proc. of AISTATS'18, 2018.

M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020. URL http://arxiv.org/abs/1910.06403.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs.LG]*, 2016.

D. Eriksson, M. Pearce, J. Gardner, R. Turner, and M. Poloczek. Scalable global optimization via local bayesian optimization. In *Advances in Neural Information Processing Systems*, volume 32, pages 5496–5507, 2019.

J. Gardner, G. Pleiss, Q. Weinberger, D. Bindel, and A. Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, volume 31, pages 7576–7586. Curran Associates, Inc., 2018.

J. Hensman, N. Fusi, and N. D. Lawrence. Gaussian processes for big data. In *Proc. of UAI'13*, page 282–290, Arlington, Virginia, USA, 2013.

F. Hutter, H. Hoos, and K. Leyton-Brown. Parallel algorithm configuration. In *Proc. of LION'12*, pages 55–70, 2012.

M. Jankowiak, G. Pleiss, and J. Gardner. Parametric Gaussian process regressors. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*,

pages 4702–4712. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/jankowiak20a.html.

A. Kuhnle, M. Schaarschmidt, and K. Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. Web page, 2017. URL https://github.com/tensorforce/tensorforce.

M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization, 2022.

Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Program.*, 45(1-3):503–528, 1989. doi: 10.1007/BF01589116. URL https://doi.org/10.1007/BF01589116.

T. Salimans, J. Ho, X. Chen, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv:1703.03864 [stat.ML]*, 2017.

M. Titsias. Variational learning of inducing variables in sparse gaussian processes. In *AISTATS*, 2009.

L. Wang, R. Fonseca, and Y. Tian. Learning search space partition for black-box optimization using monte carlo tree search. In *NeurIPS*, 2020.

Z. Wang, C. Gehring, P. Kohli, and S. Jegelka. Batched Large-scale Bayesian Optimization in High-dimensional Spaces. In *Proc. of AISTATS'18* ais (2018), pages 745–754.

M. Yuan and G. Wahba. Doubly penalized likelihood estimator in heteroscedastic regression. *Statistics & Probability Letters*, 69(1):11 – 20, 2004.