Pareto-efficient Acquisition Functions for Cost-Aware Bayesian Optimization

Gauthier Guinet[*] [†]
MIT
gguinet@mit.edu

Valerio Perrone[*]
Amazon Web Services
vperrone@amazon.com

Cédric Archambeau
Amazon Web Services
cedrica@amazon.com

Abstract

Bayesian optimization (BO) is a popular method to optimize expensive black-box functions. It efficiently tunes machine learning algorithms under the implicit assumption that hyperparameter evaluations cost approximately the same. In reality, the cost of evaluating different hyperparameters, be it in terms of time, dollars or energy, can span several orders of magnitude of difference. While a number of heuristics have been proposed to make BO cost-aware, none of these have been proven to work robustly. In this work, we reformulate cost-aware BO in terms of Pareto efficiency and introduce the cost Pareto Front, a mathematical object allowing us to highlight the shortcomings of commonly used acquisition functions. Based on this, we propose a novel Pareto-efficient adaptation of the expected improvement. On 144 real-world black-box function optimization problems we show that our Pareto-efficient acquisition functions significantly outperform previous solutions, bringing up to 50% speed-ups while providing finer control over the cost-accuracy trade-off. We also revisit the common choice of Gaussian process cost models, showing that simple, low-variance cost models predict training times effectively.

1 Introduction

Bayesian optimization (BO) is a well-established methodology to find the global minimizer of an expensive black-box function $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X} \subset \mathbb{R}^d$ [30]. One of the major use cases is the automatic tuning of machine learning algorithms [18, 5, 32, 33, 30]. BO has been successful in several other applications, including user interfaces [7], robotics [6], environmental monitoring [25], sensor networks [14], adaptive Monte Carlo [36], experimental design [3], and reinforcement learning [4, 34]. In all these settings, we typically do not have access to an analytic form of f and only have a finite evaluation budget to sequentially query f at points $\boldsymbol{x} \in \mathcal{X}$.

A limitation of standard BO is the implicit assumption that evaluating different hyperparameter configurations incurs approximately the same cost, which is rarely the case in practice. For instance, evaluating hyperparameter settings corresponding to larger neural network architectures requires higher training cost. As cost can differ in orders of magnitudes [23], some works have proposed heuristics to make BO *cost-aware* [32, 2, 23]. Unfortunately, cost-aware BO has not been studied systematically, and commonly used acquisition functions have not been proved to work robustly.

In this paper, we revisit cost-aware BO through the lens of Pareto optimality. This allows us to highlight the shortcomings of commonly-used heuristics and directly control the improvement-cost trade-off at each BO iteration. Based on this, we introduce a novel, robust cost-aware acquisition function. In extensive experiments drawn from 144 real-world hyperparameter optimization (HPO) problems, we show that our solution outperforms both classical and recent cost-aware methods. We

[*] Joint first author.

[†] Work done during an internship at Amazon Web Services.

4th Workshop on Meta-Learning at NeurIPS 2020, Vancouver, Canada.

also show that simple linear models predict training times more effectively than GPs, the default choice in the literature.

2 Background and Related Work

Consider the problem of finding $\boldsymbol{x}_\star = \text{argmin}_{\text{x}\in\mathcal{X}}\ f(\boldsymbol{x})$, where $\boldsymbol{x}$ is a hyperparameter configuration, $\mathcal{X}$ the search space, and $f(\boldsymbol{x})$ a black-box function (e.g., the map from hyperparameter configurations to validation error). As the functional form of $f(\boldsymbol{x})$ is unknown, BO replaces it with a probabilistic surrogate model, with a popular choice being the Gaussian process (GP) [29]. The next point to evaluate is obtained by repeatedly optimizing an *acquisition function* until some budget (e.g., iterations, dollars or time) is exhausted [20].

Most acquisition functions used in BO implicitly assume that all hyperparameter evaluations cost approximately the same. For example, the Expected Improvement (EI), one of the most popular acquisition function for BO, is defined by $EI(\boldsymbol{x}) = \mathbb{E}[\max(0, f_{min} - f(\boldsymbol{x}|\mathcal{D})]$ for each hyperparameter configuration $\boldsymbol{x} \in \mathcal{X}$ [26]. To make the EI cost-aware, common practice is to normalize it by the cost $c(\boldsymbol{x})$ of evaluating $\boldsymbol{x}$ [32, 27, 33]. This replaces EI with *EI per unit cost (EIpu)* defined as follows:

$$EIpu(\boldsymbol{x}) = \frac{EI(\boldsymbol{x})}{c(\boldsymbol{x})}.$$

The modified form of EI is designed to balance the improvement with the evaluation cost [32]. To compute EIpu, it is necessary to learn the cost function $c(\mathbf{x})$, which is typically modeled with a warped GP [31] fitted on the log cost $\gamma(\mathbf{x})$ [32]. However, in [23] it was shown that EIpu does not consistently improve over EI, especially when the best hyperparameters are in the most expensive regions of the hyperparameter space. To mitigate this, that work introduced a cost-cooling approach. If τ_k of the total budget τ has been used at the kth BO iteration (at $k = 0$, $\tau_k = \tau_{init}$), EI-cool is defined as $EI^k_{\text{cool}}(\boldsymbol{x}) = \frac{EI(\boldsymbol{x})}{c(\boldsymbol{x})^\alpha}$, $\alpha = (\tau - \tau_k)/(\tau - \tau_{init})$. As the parameter α decays from one to zero, EI-cool transitions from EIpu to EI. As a result, cheap evaluations are obtained before expensive ones. While more robust than EIpu, experimental results in [23] still only showed modest improvements over conventional EI. Moreover, this method requires the budget τ to be defined *a priori*, with huge yet hard to study impact on performance. Other acquisition functions, such as entropy-based ones, are not designed to account for evaluation cost either [15, 16, 35].

Alternative approaches to cost-aware BO operate in a *grey-box* setting [13, 21, 27, 37]. Among these, multi-fidelity methods, such as Hyperband [24] and its BO extensions [12, 22], assume the presence of a fidelity parameter s. An example in the context of tuning neural networks is the epoch count. This parameter acts as a noisy proxy for high-fidelity evaluations in that increasing s decreases noise at the expense of runtime. Except when s is chosen to be the dataset subsampling factor, these methods are not applicable in a black-box setting as they require an algorithm-specific fidelity parameter s. In addition, by relying on parallel computing, multi-fidelity techniques target time-efficiency rather than compute time, cost, or energy efficiency. Existing sample-efficient, BO-based extensions of Hyperband are still built on cost-unaware acquisition functions such as the EI, and could be combined with the black-box, cost-aware approaches we propose in this work. Another grey-box, cost-aware technique was recently proposed in the context of multi-objective BO [2]. Yet, this requires as an input a partial order of the search space dimensions based on *a priori* cost preferences, a rather restrictive condition in practice.

3 Pareto Efficient Bayesian Optimization

Without loss of generality, assume that cost is the time required to evaluate the black-box function f at a given BO iteration. This is easily mapped to energetic or financial cost [19]. Assume BO is run for N iterations and let $\mathcal{BO}_{\boldsymbol{\mathcal{A}}}(\mathcal{X}^N)$ denote the set of all hyperparameter configurations queried by BO using an acquisition function from class $\boldsymbol{\mathcal{A}}$.

We aim to address the following problem: *Given some budget, how do we develop a BO algorithm that uses it optimally?* Based on how the budget is defined, this maps to either one of the two following sub-problems:

- **Bi-Optimization Problem.** The budget is the maximum number of BO iterations N. The goal is to find an optimal trade-off between the accuracy of the returned solution and the

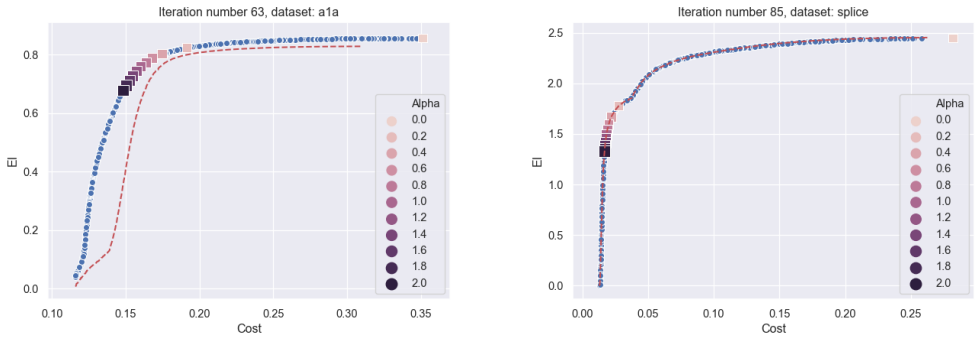

Figure 1: Representative examples of the EI-cost Pareto front at two different BO iterations. XGBoost is tuned on the `a1a` and `splice` datasets from the UCI machine learning repository [11]. The squares correspond to the maximizer of EI_α for different values of α; recall that $\alpha = 0$ corresponds to EI and $\alpha = 1$ to EIpu. The blue dots represent the Pareto front at the current BO iteration t, while the red dashed curve refers to the Pareto front at iteration $t-1$.

.

time required by BO. Formally, we want to determine the following Pareto Front:

$$\mathcal{PF}(\{(y^\star, c^\star) \mid \boldsymbol{x}_i \in \mathcal{BO}_{\mathcal{A}}(\mathcal{X}^N),\ y^\star = \min_{1 \le i \le N} f(\boldsymbol{x}_i),\ c^\star = \textstyle\sum_{i=1}^N c(\boldsymbol{x}_i)\}).$$

- **Optimal Time Allocation Problem.** The budget is the maximum wall-clock time τ. The goal is to maximize accuracy within the time budget τ, with no constraints on the number of iterations. In other words, this setting is a limited-resource allocation or constrained optimization problem. Formally, we are interested in

$$\inf_{N \in \mathbb{N}^*} \{y^\star \mid \boldsymbol{x}_i \in \mathcal{BO}_{\mathcal{A}}(\mathcal{X}^N),\ y^\star = \min_{1 \le i \le N} f(\boldsymbol{x}_i),\ \textstyle\sum_{i=1}^N c(\boldsymbol{x}_i) \le \tau\}.$$

Next, we introduce a family $\boldsymbol{\mathcal{A}}_\lambda$ of Pareto-efficient acquisition functions to address these problems.

3.1 Pareto Efficient Expected Improvement

Let $g : \mathcal{X} \to \mathbb{R}^2$ be a function over $\mathcal{X}$ mapping hyperparameters to the evaluation cost and negative EI. Given two points $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{X}$, $\boldsymbol{x}_1 \succeq \boldsymbol{x}_2$ if $\boldsymbol{x}_2$ is *weakly dominated* by $\boldsymbol{x}_1$, namely if and only if $g(\boldsymbol{x}_1)_i \le g(\boldsymbol{x}_2)_i$ for $i \in \{1, 2\}$. We write $\boldsymbol{x}_1 \succ \boldsymbol{x}_2$ if $\boldsymbol{x}_2$ is *dominated* by $\boldsymbol{x}_1$, namely if and only if $\boldsymbol{x}_1 \succeq \boldsymbol{x}_2$ and $\exists i \in \{1, 2\}$ such that $g(\boldsymbol{x}_1)_i < g(\boldsymbol{x}_2)_i$. The Pareto front of g is defined by $\mathcal{PF} = \{\boldsymbol{x} \in \mathcal{X} \mid \nexists \boldsymbol{x}' \in \mathcal{X} : \boldsymbol{x}' \succ \boldsymbol{x}\}$, that is, the set of all non-dominated points in terms of EI and cost. We argue that the next point to be evaluated at a given iteration should be in the Pareto front; otherwise, it would be possible to find a better point either in terms of cost or EI. To this end, we define the family $\boldsymbol{\mathcal{A}}_\alpha$ of acquisition functions EI_α, where

$$EI_\alpha(\boldsymbol{x}) = \frac{EI(\boldsymbol{x})}{c(\boldsymbol{x})^\alpha},\ \alpha \in \mathbb{R}^+. \tag{1}$$

In the context of multi-objective optimization, EI_α is a parametric scalarization technique [9]. The parameter α controls the trade-off between cost and EI and allows us to navigate the Pareto front, as illustrated by Figure 1. On the one hand, EI (i.e., $\alpha = 0$) picks the point with highest EI, which is necessarily in the Pareto front. However, the best point usually involves spending an extra supplementary budget for very little gain of EI due to a performance plateau. The marginal improvement from optimizing the EI is usually not worth the additional cost (see Appendix B for more details). On the other hand, EIpu (i.e., $\alpha = 1$) maximizes the expected improvement per unit of cost. By construction, the selected point will again belong to the Pareto front: it will be the point in the Pareto front with the highest slope. Yet, this may lead to a sub-optimal solution in terms of the accuracy that could have been achieved.

In order to achieve a better and more robust trade-off, we propose to leverage the Pareto front at each iteration to select a good α. To this end, we introduce the family $\boldsymbol{\mathcal{A}}_\lambda$ of *contextual EI* (CEI) acquisition functions, a cost-aware adaptation of the EI which takes into account the Pareto front at each iteration. For $\lambda \in [0, 1]$, CEI is defined as follows:

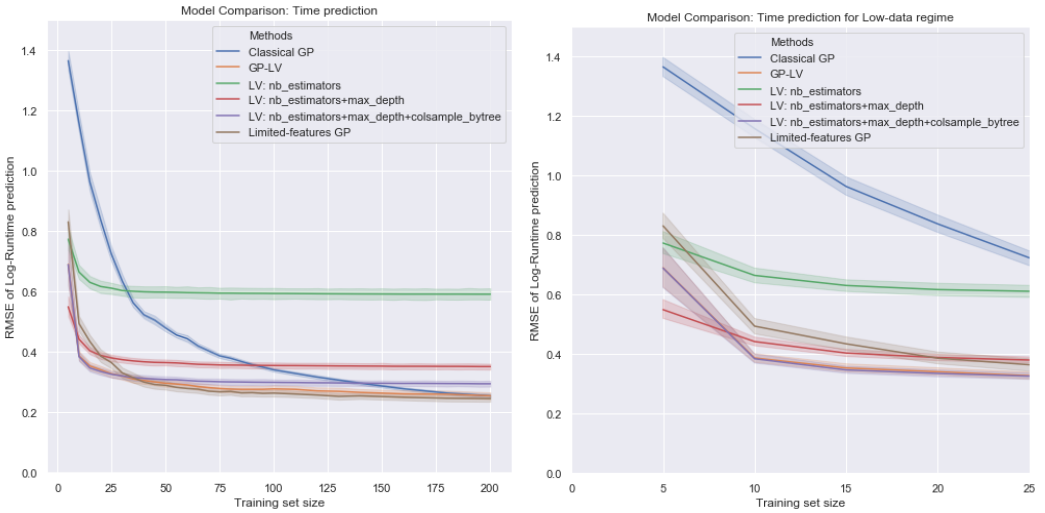

Figure 2: *Left:* Comparison of models to predict the XGBoost training time as a function of its hyperparameters. The y-axis is the RMSE averaged over 144 HPO problems and 10 seeds, while the x-axis is the number of available hyperparameter evaluations to fit the cost model. We consider three simple linear models, using 1, 2 and 3 of the most significant hyperparameters, respectively. Limited-features GP is a GP trained using only 3 features. For all models, the features are log-scaled, and we predict the log run-time. *Right:* Same experiment with a focus on the low-evaluation regime.

$$\mathrm{CEI}_\lambda(\boldsymbol{x}) := \begin{cases} -c(\boldsymbol{x}) & \text{if } EI(\boldsymbol{x}) \geq (1-\lambda)\max_{\boldsymbol{z}\in\mathcal{X}}(EI(\boldsymbol{z})), \\ -\infty & \text{otherwise.} \end{cases} \tag{2}$$

Intuitively, CEI minimizes the cost c among the points with a sufficiently high expected improvement. CEI only considers the $100 \times \lambda\%$ points with the highest accuracy and, among these, selects the one with the lowest cost i.e. optimizes the cost constrained by EI. It can be shown with continuity arguments that the solution obtained belongs to the Pareto Front. Similar to the α parameter in EI_α, λ controls the trade-off between cost and improvement. However, as we will show shortly, CEI is more robust if the shape of the Pareto Front is unconventional. It also reveals the core importance of cost models, as detailed in the next section.

3.2 Cost Modeling

The problem of predicting the cost of a computer program is well-studied by prior work. Applications include predicting system loads, dispatching computational resources, or determining computational feasibility [17, 19, 10, 28, 23]. In the black-box setting, the typical procedure is to model the cost $c(\mathbf{x})$ with a warped GP [31] that fits the log cost $\gamma(\mathbf{x})$ [32]. However, GPs extrapolate poorly, leading to high-variance cost predictions far away from data [23].

Consider the specific problem of modeling the evaluation time of XGBoost [8] as a function of its hyperparameters. We compare different algorithms: a classical GP, linear (low-variance) models with different number of features (LV) [23], and a GP trained on the residuals of a low-variance model (GP-LV). These regression models need to be data-efficient to reach good accuracy quickly in the initial BO iterations. Figure 2 shows that cost can be more efficiently captured by simple linear models, which we will use in the remainder of the paper. In the context of BO, the cost model is typically learned online as the optimization progresses. An alternative is to learn it using transfer learning from related tasks, as we detail in Appendix B.

4 Experiments

We considered 144 real-world black-box optimization problems, consisting of tuning XGBoost on 18 datasets, each processed with 8 different feature-engineering pipelines. Unless otherwise indicated,

each result is averaged across the 144 problems and 10 seeds, with 95% confidence intervals obtained via bootstrapping. The problems span multi-class, binary classification and regression tasks (more details in Appendix A). We ran all experiments on `AWS` with `m4.xlarge` machines.

Bi-optimization and Optimal Time Allocation We first study the problem of optimizing cost and performance through EI_α with a budget of 100 iterations. Figure 3 shows the impact of varying α, which controls the trade-off between cost and accuracy. The resulting front appears smooth and the relationship between performance gain and cost can be observed. Particularly in terms of median, the Pareto front is sharp with large time gains traded for limited accuracy loss. Hence, it is possible to choose a value for α based on one's cost aversion. In particular, for low α values, the mean performance loss is limited compared to the cost gains: 50% of wall-clock time is gained at 1% accuracy loss (for $\alpha = 0.1$), and 20% of wall-clock time is gained with no accuracy loss (for $\alpha = 0.01$). In contrast, standard practice of setting $\alpha = 1$ (EIpu) leads to severe accuracy degradation, while $\alpha = 0$ (EI) is unnecessarily costly.

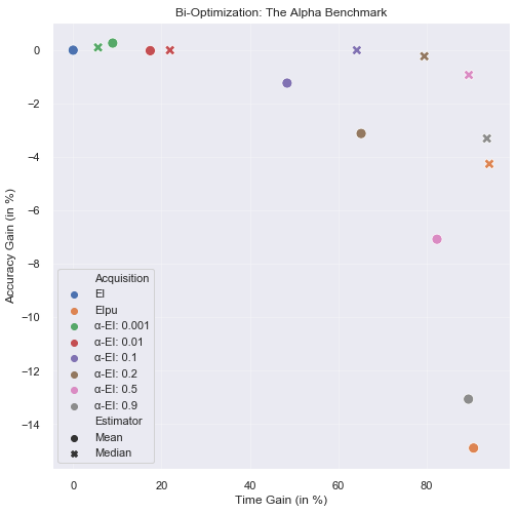

Figure 3: The accuracy-cost trade off for EI_α at a set of α levels. Percentages are calculated w.r.t the performance of EI on the same blackbox problem and then averaged across seeds and datasets. Circles and crosses correspond respectively to the mean and median value all the blackboxes.

Next, we consider the problem of optimizing performance under a time constraint. As time scales are distinct across datasets, we consider the minimum time required by one of the α-acquisitions function to reach 100 iterations. Figure 4 shows the results. Perhaps surprisingly, EIpu performs poorly even in a low-budget scenario. Generally, we observe a bell-shaped trend: α values close to 0 and 1 will produce less interesting results than intermediate values, with an optimum around 0.2. As we progressively increase the budget, EI progressively catches up with the different EI_α; for larger budgets we are less sensitive to the actual cost and an excessive cost penalty degrades the performance as expected.

The previous results and theoretical observations suggest allocating α dynamically rather than fixing it to a fixed value. We now demonstrate the benefits of adapting to the improvement-cost Pareto front

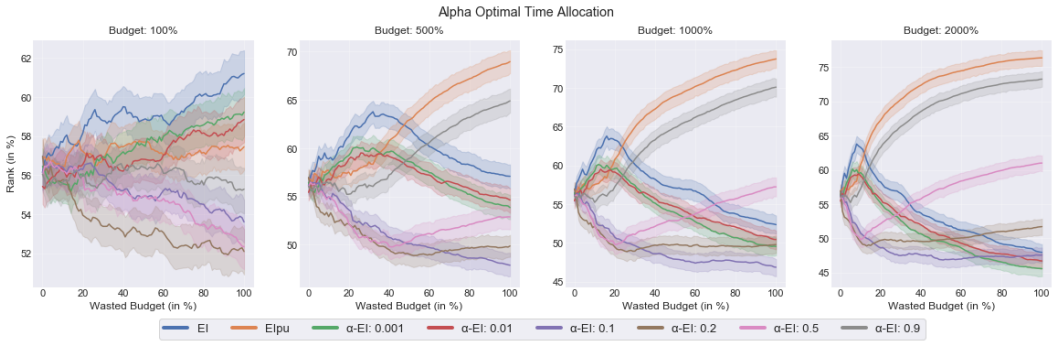

Figure 4: Comparison of EI_α with EI and EIpu in the optimal time allocation problem. Each plot corresponds to a different multiple of minimal budget (e.g. 2000% is 20 times this budget). This notion is necessary to aggregate meaningfully results across datasets. Results are ranked at each iteration based on the minimum found up to that point by each method, a lower rank corresponding to a better minimization performance. This rank is then averaged across seeds and datasets.

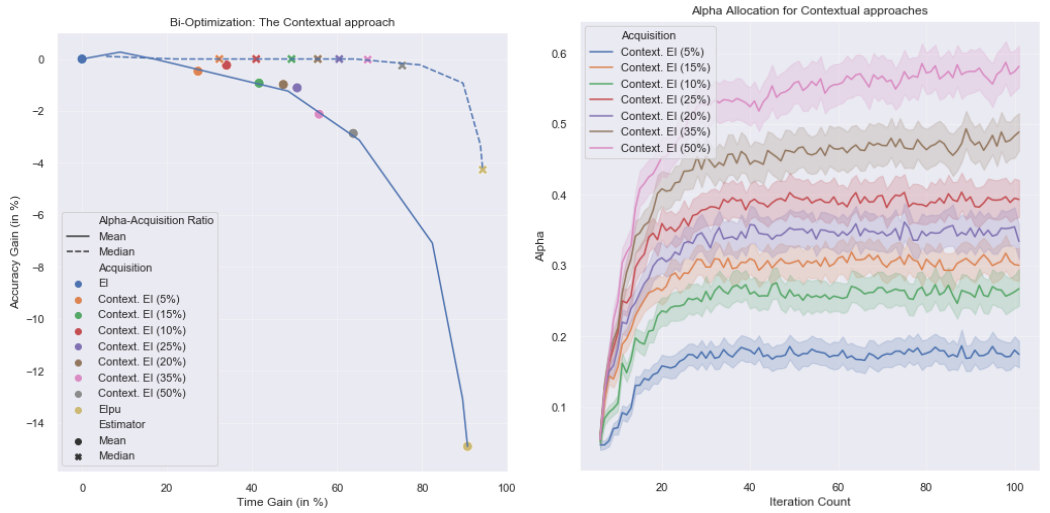

Figure 5: *Left:* Bi-Optimization results for different contextual strategies. The percentage refers to the value of the trade-off ratio. The two blue curves correspond to fixed α strategies (mean and median). *Right:* Average value of α picked by CEI against the number of BO iterations.

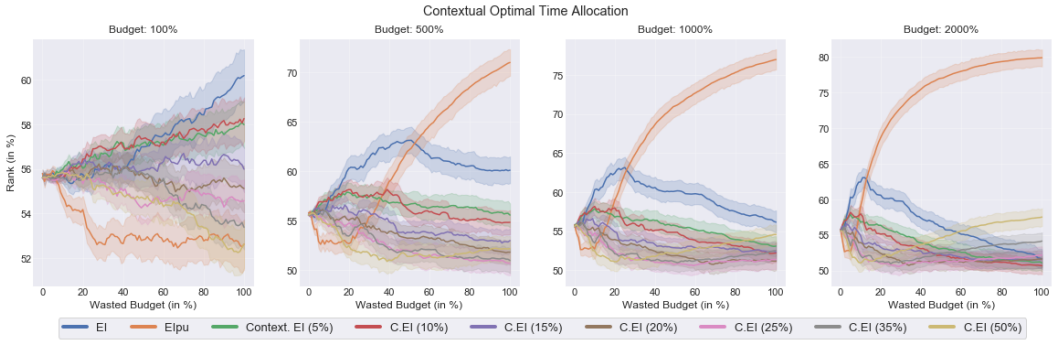

Figure 6: Comparison of CEI to the EI and EIpu acquisition functions in the optimal time allocation problem. The CEI acquisition functions consistently outperform the widely used EI and EIpu.

via CEI. Figure 5 (left) shows that CEI is an effective solution to the bi-optimization problem, with λ directly controlling the accuracy-cost trade-off. The trade-offs achieved by CEI are consistently on par or better compared to EI_α. Indeed, for a fixed λ, CEI adapts to the Pareto front at each iteration, unlike EI_α. Figure 5 (right) reports the average α value that would have led to the configuration selected by CEI, showing that CEI corresponds to dynamically allocating α at each iteration. Finally, we evaluate CEI in the context of the optimal time allocation problem, where the goal is to optimize performance within a fixed time budget. Figure 6 shows that, with the exception of extreme cost penalization, CEI consistently outperforms the popular EI and EIpu acquisitions. In particular, it is much more robust than EIpu and is able to outperform EI, even in the large budget regime.

5 Conclusion

We introduced a novel formulation of cost-aware BO based on the Pareto front of cost and expected improvement. This allowed us to highlight the shortcomings of commonly used cost penalization heuristics and develop a new family of cost-aware acquisition functions based on EI. In extensive experiments we showed that our approach outperforms both the popular EI and its cost-aware extension. Future work could leverage the Pareto front dynamics during BO to develop more adaptive schedules for the cost-sensitivity parameter α. Our method could also be applied to different acquisition functions and optimization problems.

References

[1] GPyOpt: A Bayesian optimization framework in Python. `http://github.com/SheffieldML/GPyOpt`, 2016.

[2] Majid Abdolshah, Alistair Shilton, Santu Rana, Sunil Gupta, and Svetha Venkatesh. Cost-aware Multi-objective Bayesian optimisation. *arXiv preprint arXiv:1909.03600*, 2019.

[3] Javad Azimi, Ali Jalali, and Xiaoli Fern. Hybrid batch Bayesian optimization. *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[4] Juan Cruz Barsce, Jorge A. Palombarini, and Ernesto C. Martínez. Towards Autonomous Reinforcement Learning: Automatic Setting of Hyper-parameters using Bayesian Optimization. *arXiv preprint arXiv:1805.04748*, 2018.

[5] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*. 2011.

[6] Felix Berkenkamp, Andreas Krause, and Angela P. Schoellig. Bayesian Optimization with Safety Constraints: Safe and Automatic Parameter Tuning in Robotics. *arXiv preprint arXiv: 1602.04450*, 2016.

[7] Eric Brochu, Tyson Brochu, and Nando de Freitas. A Bayesian Interactive Optimization Approach to Procedural Animation Design. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*. The Eurographics Association, 2010.

[8] Tianqi Chen and Carlos Guestrin. Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2016.

[9] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[10] S. Di, C. Wang, and F. Cappello. Adaptive algorithm for minimizing cloud task length with prediction errors. *IEEE Transactions on Cloud Computing*, pages 194–207, 2014.

[11] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[12] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.

[13] Alexander IJ Forrester, András Sóbester, and Andy J Keane. Multi-fidelity optimization via surrogate modelling. *Proceedings of the royal society a: mathematical, physical and engineering sciences*, 463(2088):3251–3269, 2007.

[14] Roman Garnett, Michael A. Osborne, and Stephen J. Roberts. Bayesian optimization for sensor set selection. In *Proceedings of the 9th ACM/IEEE Conference on Information Processing in Sensor Networks*, pages 209–219, 2010.

[15] Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, pages 1809–1837, 2012.

[16] José Miguel Henrández-Lobato, Matthew W. Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, page 918:926, Cambridge, MA, USA, 2014. MIT Press.

[17] Ling Huang, Jinzhu Jia, Bin Yu, Byung gon Chun, Petros Maniatis, and Mayur Naik. Predicting execution time of computer programs using sparse polynomial regression. In *Advances in Neural Information Processing Systems 23*, pages 883–891. Curran Associates, Inc., 2010.

[18] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, pages 507–523. Springer Berlin Heidelberg, 2011.

[19] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *arXiv preprint arXiv:1211.0906*, 2012.

[20] 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.

[21] Kirthevasan Kandasamy, Gautam Dasarathy, Jeff Schneider, and Barnabas Poczos. Multi-fidelity bayesian optimisation with continuous approximations. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1799–1808. JMLR. org, 2017.

[22] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, pages 528–536, 2017.

[23] Eric Hans Lee, Valerio Perrone, Cedric Archambeau, and Matthias Seeger. Cost-aware Bayesian Optimization. *arXiv preprint arXiv: 2003.10870*, 2020.

[24] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[25] R. Marchant and F. Ramos. Bayesian optimisation for intelligent environmental monitoring. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2242–2249, 2012.

[26] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2, 1978.

[27] Matthias Poloczek, Jialei Wang, and Peter Frazier. Multi-information source optimization. In *Advances in Neural Information Processing Systems*, pages 4288–4298, 2017.

[28] R. Priya, B. F. de Souza, A. L. D. Rossi, and A. C. P. L. F. de Carvalho. Predicting execution time of machine learning tasks using metalearning. In *2011 World Congress on Information and Communication Technologies*, pages 1193–1198, 2011.

[29] Carl Rasmussen and Chris Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[30] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104:148–175, 2016.

[31] Edward Snelson, Zoubin Ghahramani, and Carl E Rasmussen. Warped gaussian processes. In *Advances in neural information processing systems*, pages 337–344, 2004.

[32] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

[33] Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task bayesian optimization. In *Advances in neural information processing systems*, pages 2004–2012, 2013.

[34] Matteo Turchetta, Andreas Krause, and Sebastian Trimpe. Robust model-free reinforcement learning with multi-objective bayesian optimization. *arXiv preprint arXiv:1910.13399*, 2019.

[35] Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient bayesian optimization. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML, page 3627:3635, 2017.

[36] Ziyu Wang, Shakir Mohamed, and Nando Freitas. Adaptive Hamiltonian and Riemann Manifold Monte Carlo. Number 3, pages 1462–1470. PMLR, 2013.

[37] Jian Wu, Saul Toscano-Palmerin, Peter I Frazier, and Andrew Gordon Wilson. Practical multi-fidelity Bayesian optimization for hyperparameter tuning. *arXiv preprint arXiv:1903.04703*, 2019.

A Experiment settings

Our code is built on GPyOpt [1]. Kernel hyperparameters for the GP models are obtained via maximum marginal likelihood estimation [29]. We considered the problem of tuning the popular XGBoost algorithm (XGB) [8], as implemented in `scikit-learn`. This consists of a 7-dimensional hyperparameter space: number of boosting rounds in $\{1, 2, \dots, 256\}$ (log scaled), learning rate in $[0.01, 1.0]$ (log scaled), minimum loss reduction to partition leaf node `gamma` in $[0.0, 0.1]$, L1 weight regularization `alpha` in $[10^{-3}, 10^{3}]$ (log scaled), L2 weight regularization `lambda` in $[10^{-3}, 10^{3}]$ (log scaled), subsampling rate in $[0.01, 1.0]$, maximum tree depth in $\{1, 2, \dots, 16\}$.

We ran all experiments on 144 benchmarks, consisting of the problem of tuning XGBoost on 18 datasets, each processed with 8 different feature-engineering pipelines. These pipelines involve basic operations, such as categorical variable detection, one-hot encoding, as well as dimensionality reduction operations, such as PCA.[3] Results are averaged across 10 repetitions, with 95% confidence intervals obtained via bootstrapping. The problems span multi-class, binary classification and regression tasks. We used the following publicly available datasets:

- UCI datasets: `abalone`, `statloglandsatsatellite`, `turkiyestudentevaluation`, `insurancecompanybenchmarkcoil2000`, `parkinsonstelemonitoring`, `penbasedrecognitionofhandwrittendigits`.
- OpenML datasets: `3892`, `405`, `125922`, `14953`, `3007`, `287`, `503`, `189`, `558`, `1489`.
- Kaggle datasets: `team-ai-spam-text-message-classification`, `olgabelitskaya-classification-of-handwritten-letters`.

B Additional Experiments

The purpose of this appendix is two-fold. First, we give insights into the dynamics of cost and EI during BO. More precisely, we focus on its evolution and persistence. Second, we present additional results on cost learning and its impact on BO.

Evolution and Persistence of the Pareto Front We investigate the evolution of the Pareto front at each BO iteration. The difference in expected improvement across two subsequent BO iterations can be split as follows.

$$\begin{aligned} EI_{t+1}(\boldsymbol{x}) - EI_t(\boldsymbol{x}) &= \mathbb{E}_{t+1}[\max(0, y_{min}(t+1) - f(\boldsymbol{x}|\mathcal{D}))] - \mathbb{E}_t[\max(0, y_{min}(t) - f(\boldsymbol{x}|\mathcal{D}))] \\ &\simeq \underbrace{y_{min}(t+1) - y_{min}(t)}_{\textit{Constant \& Global}} + \underbrace{(\mathbb{E}_{t+1} - \mathbb{E}_t)[y_{min}(t) - f(\boldsymbol{x}|\mathcal{D})]}_{\textit{Non-linear \& Local}}, \end{aligned}$$

where $\mathbb{E}_t$ stands for the expected value at iteration t, taken with respect to the posterior mean and variance functions at t, and $y_{min}(t)$ is the current objective function minimum. Yet, the magnitude and relative impact of these two components is experimentally unknown.

The Pareto front can drastically evolve across iterations. Figure 7 shows this by comparing the Pareto front across two successive iterations, represented by the red and blue curves, respectively. This is mainly due to the non-linear and local component above. It is therefore complex to predict the state of the front at the next iteration. We observe the Pareto front tends to become stationary only for some HPO problems and after a large number of iterations.

Then, we studied the persistence of the Pareto front. Specifically, we show that the optimal hyperparameter configurations rarely remain optimal across several evaluations. Figure 7 illustrates this. This is expected due to the myopic nature of EI. Methods such as EIpu, which attempt to maximize a gain per cost, may suffer from this inconsistency.

Cost Transfer Learning In Section 3.2, we presented results on how to build a better cost model during BO. To reduce the overhead of updating two models online, an alternative is learning the cost

[3] The complete set of feature processing steps and their implementation is available at `https://github.com/aws/sagemaker-scikit-learn-extension`.

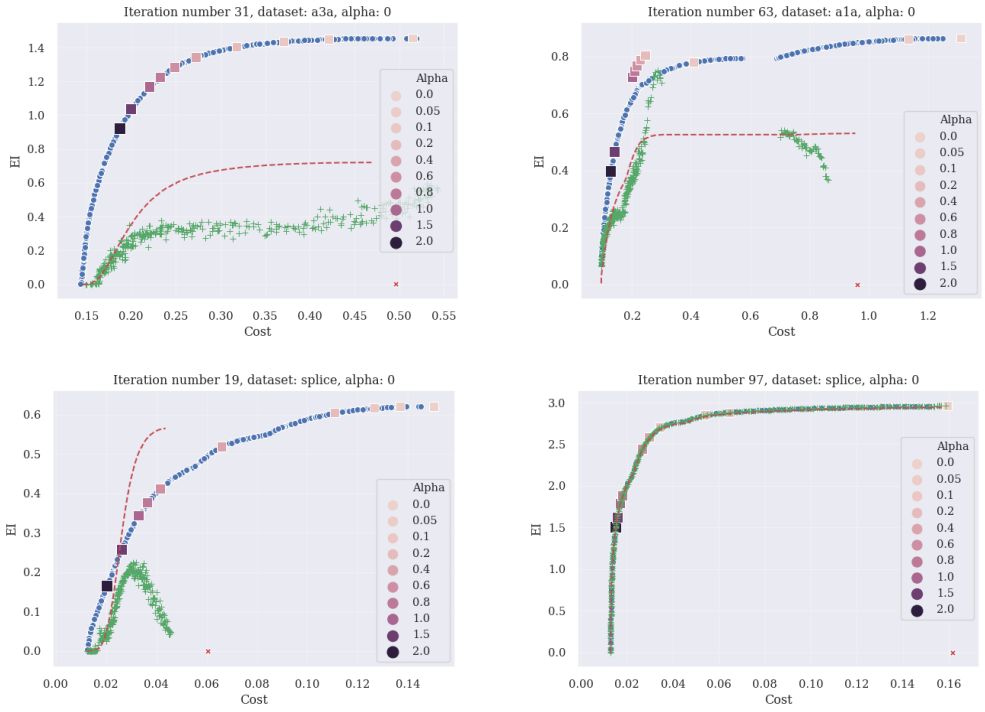

Figure 7: Representative examples of Pareto-Front visualization at different time-step t, focusing on its persistence across iterations. Each green cross corresponds to the new value of a point that belonged to the Pareto front at step $t-1$. The update is due to the fact that a new surrogate and cost model are used. Aside from this, legend and experiment settings are similar to Figure 1. The Pareto front remains stable only in the last example. XGBoost is tuned on the `a1a`, `a3a` and `splice` datasets from the UCI machine learning repository [11] and we use the EI acquisition function. .

model offline in a transfer learning fashion. Although a similar idea was explored in [28], learning a cost model across tasks has not been done in the context of BO. In terms of implementation, it is more practical to work with a fixed cost model, which does not evolve during BO. By learning a cost model offline, BO can immediately leverage an efficient cost-aware policy, without wasting any budget in an initial exploration phase.

To demonstrate this, we compare offline and online models in Figure 8. Although the cost model is easily captured by simple linear models, it does not generalize well across HPO problems. The models learned online achieve the same performance as the transfer learning models with only about ten hyperparameter evaluations. However, algorithms such as XGBoost or simple linear models can still model cost reasonably well despite not having access to any evaluations about the problem at hand.

Impact of Cost Learning on BO Performance We then evaluated the impact of the cost model on the overall BO performance. To this end, we focused on the bi-optimization problem and studied the cost-accuracy trade-off when plugging different cost models into BO (similar to [23]). Figure 9 shows that the cost model can have a strong impact on the performance-time gain. Transferring a cost model from offline datasets, which leads to a worse cost model, will also lead to a less interesting Pareto Front than the one obtained with a low-variance model learned online. More precisely, the time savings will be comparable but associated to higher accuracy losses. We also observe that the LV model performs at least as well as the GP model, at much lower computational complexity. In particular, for large values of α, the LV model leads to a better Pareto Front.

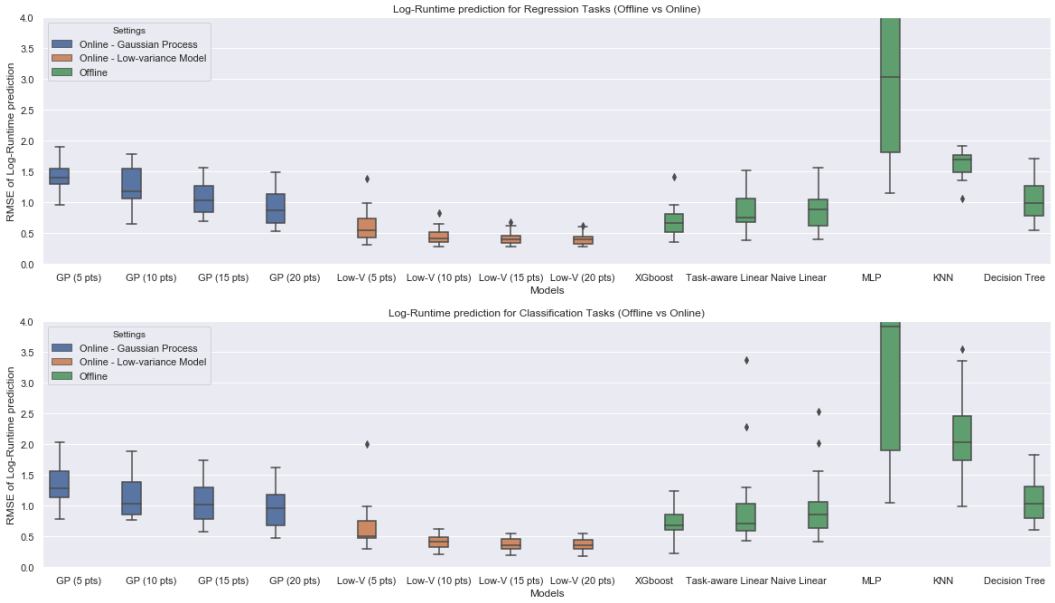

Figure 8: Comparison of different transfer learning and online cost models to predict the runtime of XGBoost as a function of its hyperparameters, averaged over 144 HPO problems. Results are split into regression and classification tasks. Blue boxes correspond to the classical GP and the orange ones to low-variance models using 3 features, with different training set sizes indicated in brackets. In green, different transfer learning regression algorithms are compared. We compare XGBoost, a low-variance linear model using data from related HPO problems (Task-Aware Linear), a LV model using all available data (Naive Linear), a multi-layer perceptron (MLP), a K-nearest neighbors regressor (KNN) and a Decision tree. Only core meta-features of datasets are used (number of classes, columns and lines) as supplementary features of the cost model. Yet, further experiments have shown that the impact of adding more complex meta-features is minor.

.

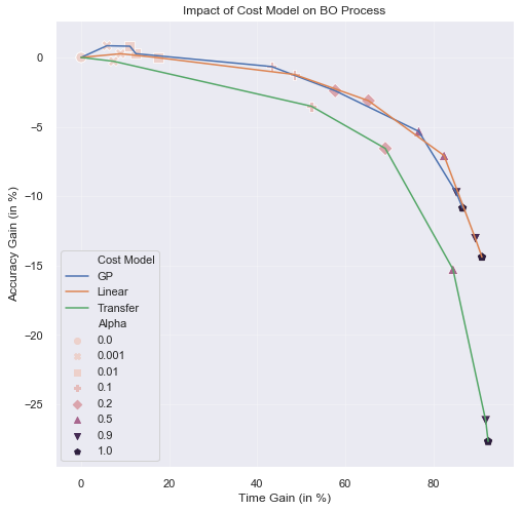

Figure 9: Comparison of different cost models when plugged into BO using EI_α, for different values of α (Bi-optimization problem). Linear refers to the LV model with 3 features, and Transfer to the offline LV model, with the same 3 features.