Faster Model-Based Optimization through Resource-Aware Scheduling Strategies

Jakob Richter[1,*], Helena Kotthaus[2,*], Bernd Bischl[3], Peter Marwedel[2], Jörg Rahnenführer[1], and Michel Lang[1]

[1] TU Dortmund University, Department of Statistics,
{richter,rahnenfuehrer,lang}@statistik.tu-dortmund.de
[2] TU Dortmund University, Department of Computer Science 12,
{firstname.lastname}@tu-dortmund.de
[3] LMU München, Department of Statistics,
bernd.bischl@stat.uni-muenchen.de

∗: Authors with equal contributions

Abstract. We present a Resource-Aware Model-Based Optimization framework `RAMBO` that leads to efficient utilization of parallel computer architectures through resource-aware scheduling strategies. Conventional MBO fits a regression model on the set of already evaluated configurations and their observed performances to guide the search. Due to its inherent sequential nature, an efficient parallel variant can not directly be derived, as only the most promising configuration w.r.t. an infill criterion is evaluated in each iteration. This issue has been addressed by generalized infill criteria in order to propose multiple points simultaneously for parallel execution in each sequential step. However, these extensions in general neglect systematic runtime differences in the configuration space which often leads to underutilized systems. We estimate runtimes using an additional surrogate model to improve the scheduling and demonstrate that our framework approach already yields improved resource utilization on two exemplary classification tasks.

Keywords: Black-Box Optimization, Hyperparameter Tuning, Model Selection, Model-Based Optimization, Resource-Aware Scheduling, Performance Management, Parallelization

1 Introduction

In the field of hyperparameter optimization for machine learning methods, efficient black-box optimization is often necessary to obtain a well-performing hyperparameter configuration for a given data set. A state-of-the-art optimization strategy for expensive black-box functions is the model-based optimization (MBO) [6]. MBO is an iterative optimization algorithm that starts on an initial set of already evaluated configurations. In each step a regression model is fitted on the so far available evaluations. It serves as a surrogate model to predict the outcome of the black-box on yet unseen configurations. The infill criterion of

the model guides the search to a new configuration which is usually a compromise between good predicted performance and uncertainty of the search space region – expected improvement is a popular choice. The new configuration is evaluated, appended to the current data and the next iteration step starts until the budget of evaluations is depleted. Many extensions to the basic MBO algorithm have been suggested for parallel point proposal [3].

One popular application for MBO is hyperparameter tuning [12, 10] where the objective function is defined as a resampled performance measure of a machine learning algorithm. Here, resource requirements like CPU utilization or memory usage heavily vary depending on the type and configuration of the applied machine learning algorithm. Heterogeneous runtimes have already been addressed in [11] where the authors suggest to model these with an additional surrogate leading to an "expected improvement per second" which favors less expensive configurations. We also use surrogate models to estimate resource requirements but instead of adapting the infill criterion, we use them for efficient scheduling of parallel point evaluations. Resource-aware scheduling is an active field of research which is often tailored specifically for different hardware platforms, from small embedded systems [13] up to heterogeneous clusters [4]. In contrast to these classical scheduling problems, we are in control of the job generation as we can query the resource model for jobs with suitable resource requirements and postpone or skip suggested jobs if deemed not promising enough.

2 Resource-Aware Model-Based Optimization

Our framework (`RAMBO`) is shown in Fig. 1. In the first of three steps, the **MBO Method** proposes a set of promising configurations w.r.t. the infill criterion. Each configuration forms a job with different resource demands. Based on all previous evaluations, we build surrogate regression models to predict the computational resources for arbitrary configurations. Such a model is called *Job Utility Estimator* and is used to create *Job Profiles*. Configurations to evaluate are selected in the **Job Selection** step. Jobs are prioritized depending on their estimated usefulness for optimization and their predicted resource requirements. The **Scheduling** step uses the estimated *Job Profiles* and a *System Description* (e.g., number of CPUs and free memory) to efficiently map the jobs to the available resources. The jobs are started and can be monitored by a **Job Tracker**. Since job profiles are only estimated, a job whose resource utilization deviates from its predicted requirements might need to be rescheduled or stopped to guarantee efficient resource utilization. We propose two possibilities to update the model with results. One way is the *synchronous* feedback, where the results of all jobs within one iteration are gathered before each model update. The other way is to update the model each time a job has finished its computation in an *asynchronous* fashion. Either way, the updated model is then used to propose new candidate points.

To demonstrate our general framework, we show a simple exemplary setup in this work. We pick kriging as surrogates to model the misclassification error

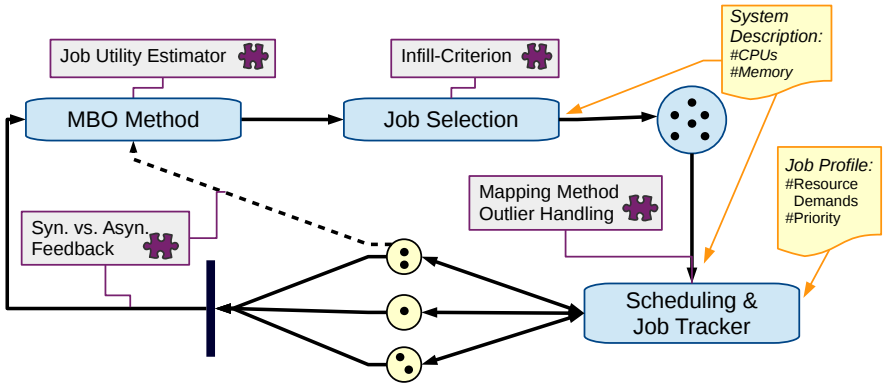

Fig. 1. Ressource-Aware Model-Based Optimization Framework.

and the logarithmic runtime. We opt for a multipoint *Lower Confidence Bound* (LCB), which is an optimistic estimate of the objective function, similar to [5] as infill criterion, which we call qLCB. qLCB simultaneously generates q configurations by drawing q random values λ_i $(i = 1, \ldots, q)$ from the exponential distribution with a mean of 2. Each λ_i results in a different trade-off between exploitation ($\lambda_i \downarrow$) and exploration ($\lambda_i \uparrow$) and thus leads to a different optimal configuration $\mathbf{x}_i^*$ after solving

$$\mathbf{x}_i^* := \arg\min_{\mathbf{x}} \left[\mathrm{LCB}(\mathbf{x}, \lambda_i)\right] = \arg\min_{\mathbf{x}} \left[\hat{y}(\mathbf{x}) - \lambda_i \hat{s}(\mathbf{x})\right]. \tag{1}$$

Here, $\hat{y}(\mathbf{x})$ denotes the posterior mean and $\hat{s}(\mathbf{x})$ the root of the posterior standard deviation of the regression model at point $\mathbf{x}$, respectively. Unfortunately, there is no direct ordering of the set of obtained candidates $\mathbf{x}_i^*$. Therefore, we assign candidates with a balanced exploration-exploitation trade-off a higher priority: $p_i = -\left|\log(\lambda_i) - \log(2)\right|$ is inversely proportional to the absolute distance of λ_i to its expected value 2 on a log-scale.

For scheduling, we use the synchronous approach. In each iteration we generate a list of $q = 3m$ proposed jobs with the help of qLCB. We then determine the job j_{i^*}, $i^* := \arg\max_i p_i$, with highest priority and run it CPU_1 exclusively. Accordingly, on a system with m homogenous CPUs the remaining jobs are scheduled on $\mathrm{CPU}_2, \ldots, \mathrm{CPU}_m$, limited by the upper time bound $\hat{t}_{i^*}$, which is directly derived from the estimated runtime of job j_{i^*}. Jobs which have an estimated runtime $\hat{t}_i \leq \hat{t}_{i^*}$ are mapped in decreasing order of their priorities to the remaining CPUs in a greedy first fit manner. A job j_i is mapped on CPU_k if its runtime $\hat{t}_i \leq \hat{t}_{i^*} - \sum_{i \in J_k} \hat{t}_i$ where J_k is the set of jobs already mapped to CPU_k. Jobs of the inital list that do not fit on any CPU are discarded. If any CPU is left without a job we query the surrogate model for a new job for each CPU with a runtime smaller or equal to $\hat{t}_{i^*}$ to fill the gaps. When all scheduled jobs are evaluated the surrogate model is updated and the iteration starts over.

3 Evaluation

The subject of the experimental setup is to apply our framework on the `w6a`[4] and `magic04`[5] data set to configure an SVM with the radial basis function kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \left\|\mathbf{x} - \mathbf{x}'\right\|^2) \tag{2}$$

as implemented in the `R` package `e1071` [7], based on `libsvm`. The kernel parameter γ and the cost C of constraint violations are both box-constrained to the interval $[-15, 15]$ on a $\log_2$-scale. We compare our approach to two established alternatives:

Random Search (`RS`): A simple parallelized random search. This relatively naive yet often effective [1] approach does not need a synchronization step like MBO, therefore the next random point will be scheduled immediately after each function evaluation which guarantees maximum load of all CPUs.

qLCB: A simple MBO approach with a multipoint LCB infill criterion [3], using a kriging model and naive scheduling. At each sequential step, $q =$ `ncores` points are selected minimizing the LCB (1) w.r.t. random $\lambda_i \sim \text{Exp}(\frac{1}{2})$ $(i = 1, \ldots, q)$.

Since the concept of a fixed budget of evaluations does not translate well into a scenario with heterogeneous runtimes, we define the budget via the elapsed time. We use a 3-fold cross validation to define the objective function for the tuner and an outer 10-fold cross validation to evaluate the optimization results. All variants start with an initial latin hypercube design with 10 points. To increase comparability, initial designs are fixed per outer cross-validation fold.

The software is implemented in `R` using `mlr`[6] to interface the machine learning algorithms and `mlrMBO`[7] as optimization toolbox. `BatchExperiments` [2] is used to parallelize the experiments on high performance computing cluster. The `traceR` framework [8, 9] guarantees reliable measures of computational resources.

Fig. 2 shows the mean misclassification errors (MMCE) of the best configuration after 1, 10, 120 and 180 minutes. The left hand side displays the tuning error, i.e. the over-optimistic error on the internal tuning set. The right hand side shows the MMCE on the outer cross-validation. Unfortunately, on these data sets only marginal improvements are achieved after evaluation of the initial design. Yet our RAMBO approach seems to perform well, yielding comparable performance and sometimes slightly less variance. The reasons for this can be found in Fig. 3 which visualizes the mapping of parallel jobs. We can observe unused CPU time for qLCB whereas `RAMBO` balances long execution times more evenly. The estimation of runtimes reliably estimates the runtimes so that only 2.3% of the evaluations exceed $\hat{t} + 2 \cdot s(\hat{t})$. qLCB often schedules four jobs

[4] Platt: `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/w6a`
[5] Bock: `https://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope`
[6] Bischl et al., `mlr`: Machine Learning in R. `https://github.com/mlr-org/mlr`
[7] Bischl et al., `mlrMBO`: Model-Based Optimization for mlr. `https://github.com/berndbischl/mlrMBO`

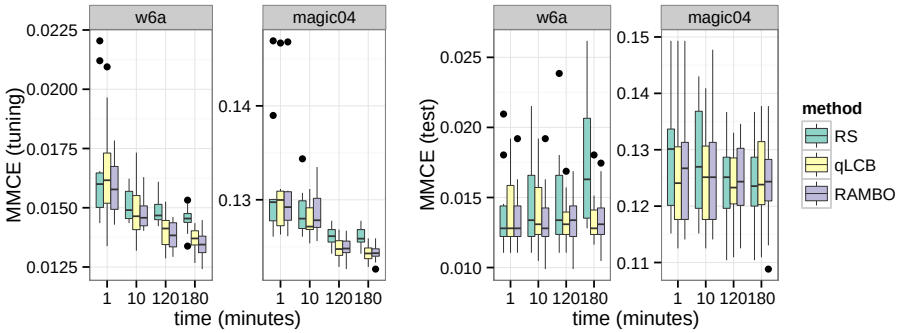

Fig. 2. Averaged misclassification errors (MMCE): tuning (left) and test data (right) for the best observed configuration after a given time budget.

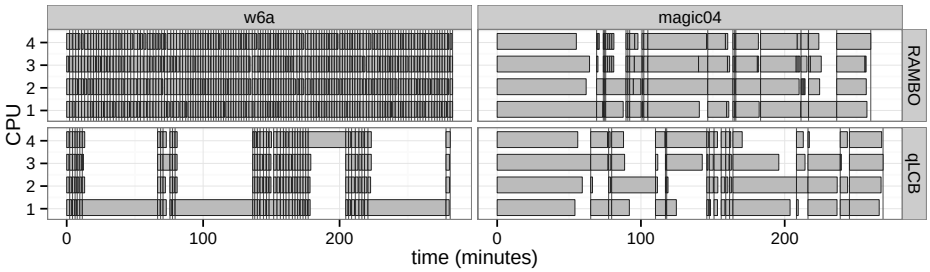

Fig. 3. Scheduling visualization for one run: The boxes show the mapping of jobs on CPUs. Less empty spaces indicate higher CPU utilization. Vertical lines indicate the end of one MBO iteration.

with vastly different runtimes and hence wastes available CPU time idling. Thus our results demonstrate that `RAMBO` achieves higher **CPU utilization**, meaning more evaluations which yields better knowledge of the hyperparameter space and thus higher confidence in the optimization result. It also shows on `magic04` that it not only prefers short jobs but is also able to schedule long jobs more efficiently. On the `w6a` dataset `RAMBO` is capable of evaluating twice as many configurations as the unscheduled baseline method qLCB. In contrast it only yields 25% more evaluations on the `magic04` dataset which indicates that promising configurations have longer runtimes then average and vice versa for `w6a`.

4 Conclusion

With our `RAMBO` framework we present a novel approach to perform a faster model-based optimization through resource-aware scheduling. We demonstrate that our yet heuristic mapping approach already leads to improved resource utilization and thus to more evaluations within the same time budget. This potentially yields a better knowledge of the hyperparameter space and thus higher confidence in the optimization result. In order to efficiently use hardware resources, we are planning further improvements. Firstly, further work will con-

centrate on integrating memory profiles since memory usage heavily influences runtime if the amount of RAM in the system is too small to hold all required data. Secondly, we aim to improve the resource estimation. Thirdly, we are planning to implement dynamic scheduling of jobs for cases of remaining deviations. Fourthly, we plan to implement a multi-objective approach with respect to hardware costs, memory, runtime and priority for performance optimization for an more optimized resource-aware scheduling strategy. This is especially important for an efficient utilization of heterogeneous architectures.

Acknowledgments. This work was partly supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876, A3.

References

1. Bergstra, J., Bengio, Y.: Random Search for Hyper-Parameter Optimization. The Journal of Machine Learning Research 13(1), 281–305 (2012)
2. Bischl, B., Lang, M., Mersmann, O., Rahnenführer, J., Weihs, C.: BatchJobs and BatchExperiments: Abstraction Mechanisms for Using R in Batch Environments. Journal of Statistical Computation and Simulation 64(11), 1–25 (2015)
3. Bischl, B., Wessing, S., Bauer, N., Friedrichs, K., Weihs, C.: MOI-MBO: Multiobjective Infill for Parallel Model-Based Optimization. In: Learning and Intelligent Optimization Conference. Florida (2014)
4. Delimitrou, C., Kozyrakis, C.: Quasar: Resource-efficient and QoS-aware Cluster Management. In: ASPLOS '14. pp. 127–144. ACM (2014)
5. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Parallel Algorithm Configuration. In: Learning and Intelligent Optimization, pp. 55–70. Springer (2012)
6. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient Global Optimization of Expensive Black-Box Functions. Journal of Global Optimization 13(4), 455–492 (1998)
7. Karatzoglou, A., Meyer, D., Hornik, K.: Support Vector Machines in R. Journal of Statistical Software 15(1), 1–28 (2006)
8. Kotthaus, H., Korb, I., Lang, M., Bischl, B., Rahnenführer, J., Marwedel, P.: Runtime and Memory Consumption Analyses for Machine Learning R Programs. Journal of Statistical Computation and Simulation 85(1), 14–29 (2015)
9. Kotthaus, H., Korb, I., Marwedel, P.: Performance Analysis for Parallel R Programs: Towards Efficient Resource Utilization. Tech. Rep. 01/2015, Department of Computer Science 12, TU Dortmund University (2015), SFB876 Project A3
10. Lang, M., Kotthaus, H., Marwedel, P., Weihs, C., Rahnenführer, J., Bischl, B.: Automatic Model Selection for High-Dimensional Survival Analysis. Journal of Statistical Computation and Simulation 85(1), 62–76 (2015)
11. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian Optimization of Machine Learning Algorithms. In: NIPS workshop on Bayesian optimization, sequential experimental design, and bandits. pp. 2960–2968 (2012)
12. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In: Proceedings of ACM SIGKDD. pp. 847–855 (2013)
13. Tillenius, M., Larsson, E., Badia, R.M., Martorell, X.: Resource-Aware Task Scheduling. ACM Trans. Embed. Comput. Syst. 14(1), 5:1–5:25 (2015)