Studies on a Bayesian Optimization Based Approach to Tune Hyperparameters of Matheuristics

Sophie Hildebrandt[1,2][0000-0001-9606-3695], Sina Nunes[2][0009-0000-8357-7435], Meik Franke[1][0000-0002-0382-0480] and Guido Sand[2][0009-0005-3162-4726]

[1] University of Twente, Drienerlolaan 5, 7522 NB Enschede, Netherlands
[2] University of Applied Science Pforzheim, Tiefenbronner Str. 65, 75175 Pforzheim, Germany
sophie.hildebrandt@hs-pforzheim.de

Abstract: Mixed-integer programming can handle optimization problems with complex constraints, but its computational cost often suffers from the combinatorial complexity of the problem. Decomposition-based matheuristics address this issue by splitting large-scale mixed-integer programs (MIPs) into smaller subproblems. Matheuristics typically exhibit hyperparameters that may affect their performance. An analysis of related work reveals that the optimization potential of hyperparameters is often left unexploited, leading to both inferior MIP-solutions and unnecessarily high computational costs.

This paper studies a novel algorithmic approach to tune hyperparameters of matheuristics by Bayesian optimization. Fundamental properties of the algorithmic approach are examined by computational experiments with small- and large-scale instances of the use case. The results exhibit two natural and competing objectives of the tuning problem: optimizing the MIP-objective and the computational cost. While the two objectives can be optimized separately for small-scale instances, they need to be handled jointly for large-scale instances.

In future research the multi-objective aspect of the hyperparameter tuning problem will be examined more deeply, and the single-instance approach will be extended to multiple instances.

Keywords: Mixed-integer programming, matheuristics, hyperparameter tuning, Bayesian optimization.

1 Introduction

Mixed-integer programs (MIPs) can be applied to solve real-world optimization problems with complex constraints like scheduling, routing or packing. An MIP comprises four components: Firstly, the degrees of freedom which are reflected by integer- or real-valued variables and whose values are determined during the solution process. Secondly, the algebraic objective function defines how the objective value, that should be minimized or maximized, depends on the variables. Thirdly, the algebraic equality and inequality constraints which restrict the feasible values of the variables. Fourthly, the

parameters which define instances of the MIP. Several general-purpose solvers for MIPs exist: CPLEX[1], Gurobi[2] and XPRESS[3] are among the most prominent ones.

Lots of real-world optimization problems are NP-complete such that the solution of large-scale instances is prohibited by the exploding computational cost. Heuristic decomposition schemes aim at reducing the computational cost while losing only a little optimization potential of the MIP. They split a large-scale monolithic MIP into a polylithic MIP comprising several smaller sub-MIPs. However, while solutions of monolithic MIPs provide optimality certificates (proof of optimality or a conservative optimality gap), solutions of polylithic MIPs, that are based on heuristic decomposition schemes, do not. Heuristic decomposition-schemes for MIPs are called matheuristics and typically comprise hyperparameters which steer the decomposition process.

The hyperparameters may affect the performance of matheuristics (see e.g. [1]) such that they should be optimized. However, their optimization potential is often left unexploited, leading to both inferior MIP-solutions and unnecessarily high computational costs. In contrast to the underlying MIP, the hyperparameter tuning is a black-box optimization problem for which no algebraic formulation is known. The authors propose to optimize the hyperparameters of matheuristics using Bayesian optimization (BO). In this paper such an approach is studied based on a particular use case.

The remainder of the paper is structured as follows: In section 2, related work on hyperparameter tuning of matheuristics and the motivation for choosing BO is discussed. In section 3, the BO-based approach for tuning matheuristics detailing the algorithmic framework is presented. In section 4, the MIP-problem and the decomposition-based matheuristic used as case study are described along with the considered hyperparameters. In section 5 computational experiments with small-scale MIP-instances and two separate objectives of the tuning problem are studied. In section 6 computational experiments for large-scale MIP-instances are studied, whereby in contrast to section 5 the two objectives are handled jointly. In section 7, the conclusions from the studies are summarized and future research directions are outlined.

2 Related Work

Hyperparameter tuning and algorithm selection is widely and successfully applied to various machine learning models, including neural networks, support vector machines, decision trees and random forests as well as gradient boosting machines. Numerous studies have demonstrated that proper hyperparameter tuning can enhance machine learning model accuracy, generalization and performance, as well as prevent overfitting (see e.g. [2]).

Common techniques for hyperparameter tuning for machine learning models include grid search and random search, as well as population-based optimization methods such as evolutionary algorithms. More recent approaches such as successive halving,

[1] https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer
[2] https://www.gurobi.com/
[3] https://www.fico.com/en/products/fico-xpress-optimization

hyperband, and BO have gained traction for their ability to allocate computational resources more efficiently (see e.g. [3]).

Beyond traditional machine learning models, hyperparameter tuning and algorithm selection techniques have also been applied to mathematical programming. At a high level, learning-based approaches can assist in selecting among different algorithmic strategies: For instance, Degroote et al. [4] used machine learning to choose between two tabu search variants and mixed-integer programming for solving the generalized assignment problem. Once an algorithmic strategy is chosen, further decisions arise, for example, whether to apply decomposition schemes or not. Mitrai and Daoutidis [5] demonstrated how machine learning can support this decision, and in a subsequent work, they proposed a hyperparameter tuning framework specifically for Benders decomposition, a general and exact method for solving MIPs [6]. The sub-MIPs resulting from the decomposition can be solved using standard MIP solvers. Extensive research has explored tuning strategies for standard MIP solvers (e.g. [7, 8]). Going even deeper into solver internals, machine learning has also been leveraged to guide key components of branch-and-bound algorithms, such as selecting branching variables, nodes, or cutting planes, as surveyed comprehensively by Bengio et al. [9].

While hyperparameter tuning is well-established for machine learning models and metaheuristics and is also applied to rigorous mathematical programming in various ways, there is no research on its use in matheuristics known. A key challenge in matheuristic tuning is the high computational cost associated with evaluating each hyperparameter configuration, as it involves solving one or more MIPs. Consequently, exhaustive methods such as grid search, random search, or even population-based approaches like evolutionary algorithms or particle swarm optimization often become impractical due to their high evaluation budgets and low sample efficiency [3]. To address this issue, more efficient techniques such as successive halving, hyperband, and BO have been proposed [3]. Both bandit-based methods like successive halving and hyperband and BO-based methods are promising approaches for the hyperparameter tuning of matheuristics. In this work, BO is applied to tune the hyperparameters of matheuristics.

3 Proposed Approach

A novel approach based on BO is proposed for the hyperparameter tuning of matheuristics. Fig. 1 visualizes the building blocks of the algorithmic architecture and the information exchanged between them. On the one hand, BO is a well-established method for hyperparameter tuning, particularly for expensive-to-evaluate objective functions. On the other hand, the use of matheuristics that decompose a monolithic MIP into a polylithic MIP, and the tuning of standard MIP solvers have both been widely studied. To the knowledge of the authors, studies on a BO-based approach to tune hyperparameters of matheuristics are not described in literature yet.

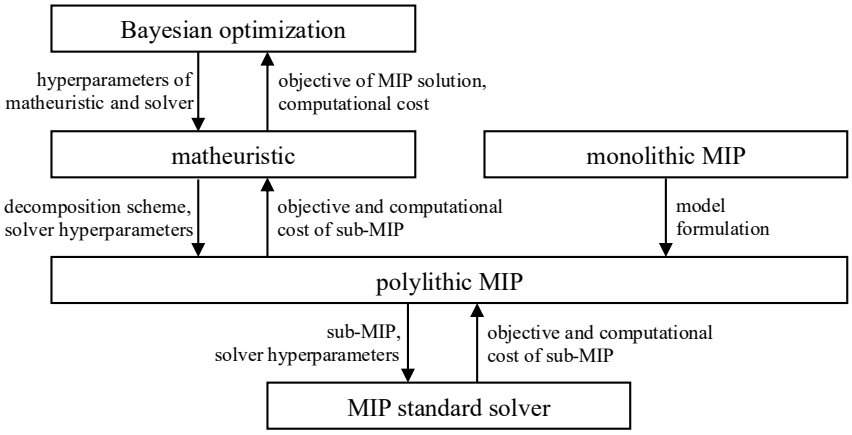

Fig. 1. Algorithmic architecture for the hyperparameter tuning of matheuristics

In each iteration, the BO proposes new hyperparameter values for both the matheuristic and the MIP standard solver. With the decomposition scheme of the parameterized matheuristic and the model formulation of the monolithic MIP a polylithic MIP is generated. The polylithic MIP consists of several smaller sub-MIPs which are solved by a MIP standard solver. The MIP standard solver computes a solution for the sub-MIP using the solver hyperparameters given. The objective value of the sub-MIP and the computational cost for solving the sub-MIP are returned to the matheuristic. After all sub-MIPs are solved, the matheuristic returns the objective value of the polylithic MIP solution and the total computational cost used to solve the polylithic MIP to the BO. The BO updates the surrogate model and calculates new promising hyperparameter values using an acquisition function. This is done until a termination criterion for the BO, such as the number of iterations, a time limit or the number of retries to propose new hyperparameter values is met.

The algorithm shown in Fig. 1 is implemented using the python package SMAC3[4] for the BO (with a gaussian process as surrogate model and the expected improvement

[4] https://github.com/automl/SMAC3

as acquisition function), the python package Pyomo[5] is used to implement the MIP and CPLEX[6] is used as MIP standard solver.

4 Case Study

4.1 Hoist Scheduling Problem

The BO-based approach is studied using a single hoist scheduling problem (HSP) described by Aguirre et al. [1] as a use case. HSPs appear in the production scheduling of electroplating plants, see Fig. 2.

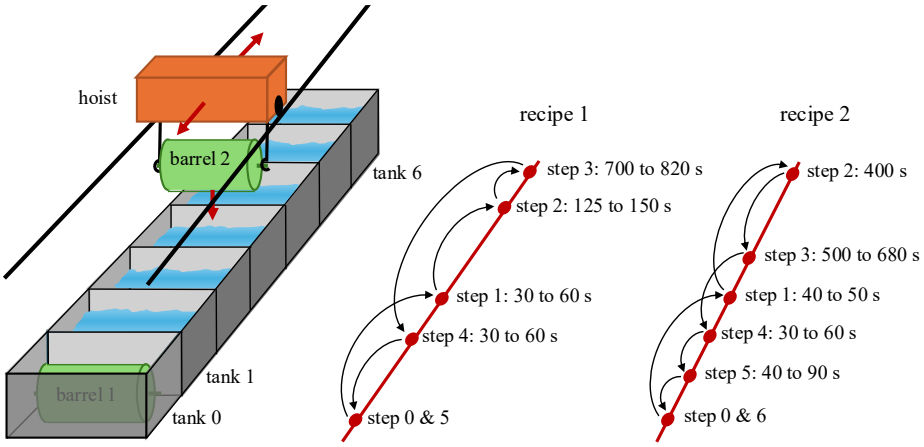

Fig. 2. Schematic visualization of an electroplating plant [10]

These plants consist of different tanks which are arranged in one line and contain different liquids. The products are filled into perforated barrels and immersed in the liquids for a limited time (see Fig. 2 for the lower and upper processing time limits). The products are processed in specific sequences of steps defined by recipes. Several products can be processed simultaneously according to different recipes. The barrels are transported between the tanks by a single hoist with finite speed.

The objective of the scheduling problem is to create a hoist schedule with minimum makespan (i.e. time to process all products), specifying which product is processed at which time for how long in which tank. [10]

4.2 Mixed-Integer Program

The HSP can be formulated as an MIP. The main variables of the MIP are the starting and the finishing times of each processing step of each product. The values of the variables must meet constraints such as:

[5] https://www.pyomo.org/
[6] https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer

- The start time plus the processing time equals the finishing time.
- The processing time must not exceed the minimum and the maximum allowed processing time.
- A processing step can only be started if the previous processing step and the transportation of the barrel to the subsequent tank are finished.
- At most one barrel can be processed within a tank at a time.
- The hoist can only carry one barrel at a time.
- The hoist arrives empty and in time at the tanks to pick up a barrel.

Further details on the formulation of the MIP can be found in the paper by Aguirre et al. [1].

4.3 Matheuristic

The monolithic MIP from above cannot be solved in reasonable times for large-scale instances. Therefore Aguirre et al. [1] developed a matheuristic which decomposes the monolithic MIP into a polylithic MIP with smaller sub-MIPs. The matheuristic comprises two steps: A construction step to create an initial schedule and an improvement step to improve the initial schedule.

The matheuristic can be explained using a window which is shifted over the *product sequence* (see Fig. 3).

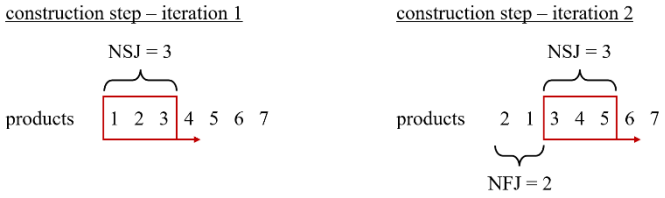

Fig. 3. Schematic visualization of the first two iterations of the construction step

In this example, the construction step starts with the *initial product sequence* 1, 2, 3, … 7. In the first iteration, a window of width *NSJ* (number of scheduled jobs) starts at the first product. All products within the window are scheduled by solving a sub-MIP, while the products to the right of the window are not considered. Afterwards, the window is shifted by *NFJ* (number of fixed jobs) products to the right. In the second iteration the products within and to the left of the window are scheduled by solving a sub-MIP. Again, the products to the right of the window are not considered. While the products within the window can be scheduled freely, the sequence of products to the left of the window are fixed to the result of iteration 1. This procedure is repeated until an initial schedule for all products is constructed.

The improvement step starts with the schedule from the construction step. A window of width *NRJ* (number of rescheduled jobs) is shifted over the product sequence. In each iteration a sub-MIP is solved considering all products. While the products within the window can be freely rescheduled, the sequences of the products outside the window remain fixed. In each iteration the schedule can either be improved or not (but it cannot deteriorate). In the first case the new product sequence is kept, and the window

is shifted back to the beginning of the sequence. In the second case the window is shifted by one product to the right. Fig. 4 visualizes these two cases.

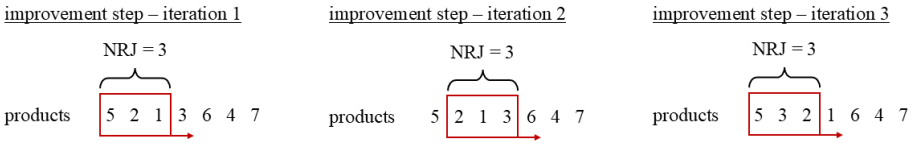

Fig. 4. Schematic visualization of the first three iterations of the improvement step

In iteration 1 the products 5, 2 and 1 can be rearranged during the optimization while the sequence of the other products is fixed, but the schedule is not changed (case 2). Therefor the window is shifted one product to the right. In iteration 2 the products 2, 1 and 3 are rescheduled leading to an improved schedule. Therefore, the window is shifted back to the beginning of the sequence. In iteration 3 the products 5, 3 and 2 can be rearranged. This procedure is repeated until the window is shifted completely over the product sequence with no improvement.

This matheuristic comprises at least four hyperparameters (of which only number 2 and 4 were considered in [1]):

1. The *initial product sequence* for the construction step,
2. the window width *NSJ* in the construction step,
3. the step size of the window *NFJ* in the construction step, and
4. the window width *NRJ* in the improvement step.

Additionally, the accepted optimality gap of the sub-MIP ("*MIPgap")* is considered as a hyperparameter of the MIP standard solver (see Fig. 1). The optimality gap is the difference between the makespan of the best feasible solution found so far and a conservative lower bound for the makespan. The hyperparameter *MIPgap*[7] is used as a termination criterion for the sub-MIPs. A *MIPgap* of 0 % means that the sub-MIPs must be solved to proven optimality. A *MIPgap* of 100 % means that the first feasible solution found by the MIP standard solver is used.

The hyperparameters *NSJ*, *NRJ* and *NFJ* are integer valued. The *initial product sequence* is a categorical and the *MIPgap* is a continuous hyperparameter.

5 Computational Experiments with Small-Scale MIP-Instances

The aim of the matheuristic tuning is to find hyperparameter values that lead to a good objective value of the MIP at reasonable computational cost. So, there are two natural objectives: The objective of the MIP, for example minimize the makespan, and the minimization of the computational cost, often measured as the CPU-time used on a given hardware. Intuitively, these two objectives are competing: If the MIP is decomposed into many sub-MIPs, the CPU-time is expected to decrease while the makespan

[7] https://support.gurobi.com/hc/en-us/articles/8265539575953-What-is-the-MIPGap

is expected to increase. In this paper, the two objectives are analyzed separately to understand how they vary with the hyperparameters, how the BO behaves and how the MIP-solutions differ for both objectives.

For small-scale MIP-instances the full matheuristic scheme can be applied and the sub-MIPs can be solved to proven optimality with reasonable computational cost for any hyperparameter values. In small discrete or discretized hyperparameter spaces a full grid search can be completed in reasonable computational time and locate all globally optimal hyperparameter values.

To limit the size of the hyperparameter space, a full grid search is conducted with only the two hyperparameters *NSJ* and *NRJ* while the remaining hyperparameters are fixed to specific values. The behavior of the BO is studied in two- and four-dimensional hyperparameter-spaces for the two objectives, makespan and CPU-time, separately. In the four-dimensional space the hyperparameters *NSJ*, *NRJ*, *NFJ* and *MIPgap* are tuned. The BO in the two-dimensional space is evaluated based on the globally optimal solutions known from the grid search. In the four-dimensional space full grid search becomes impractical due to its size and the continuous nature of the hyperparameter *MIPgap*.

All computational experiments were performed on a machine with an Intel Xeon E5-2630 v4 CPU running at 2.20 GHz.

5.1 Grid Search

Four small-scale MIP-instances are studied. They comprise a small number of products (6) and the monolithic MIPs can be solved in less than 3 hours CPU-time to proven optimality. Instance s1 is taken from Aguirre et al. [1] while instances s2, s3 and s4 are newly created. In Table 1 the main parameters of the small-scale instances are listed.

Table 1. Main parameters of the small-scale instances

Instance	Number of products	Number of stages	Number of recipes	Number of tanks
s1	6	4-8	3	36
s2	6	4-6	3	7
s3	6	6	2	5
s4	6	6	1	5

For the grid search the hyperparameters *NSJ* and *NRJ* are varied while *NFJ* is fixed to 1, *MIPgap is fixed to 0 %* and *initial product sequence* is either ordered or unordered. *NSJ* and *NRJ* are integer in nature and bounded between 1 to 6 such that 36 solutions exist. For smaller values of *NSJ* and *NRJ*, the MIP is decomposed stronger into more and smaller sub-MIPs. For larger values of *NSJ* and *NRJ*, fewer and larger sub-MIPs need to be solved. The largest possible value for *NSJ* and *NRJ* is the number of products (here: 6). For this value the monolithic MIPs are actually not decomposed, such that the monolithic MIP is considered as the (one) sub-MIP in the construction or the improvement step, respectively.

Fig. 5 shows for the representative instance s2 the results of the grid search for the two objectives, makespan and CPU-time, in the *NSJ*-*NRJ*-space. (Note that the integer-values are interpolated for a better visualization.) Two *initial product sequences* are studied: an ordered and an unordered one. In the ordered sequence the products with the same recipe are next to each other, while they are alternated in the unordered sequence.

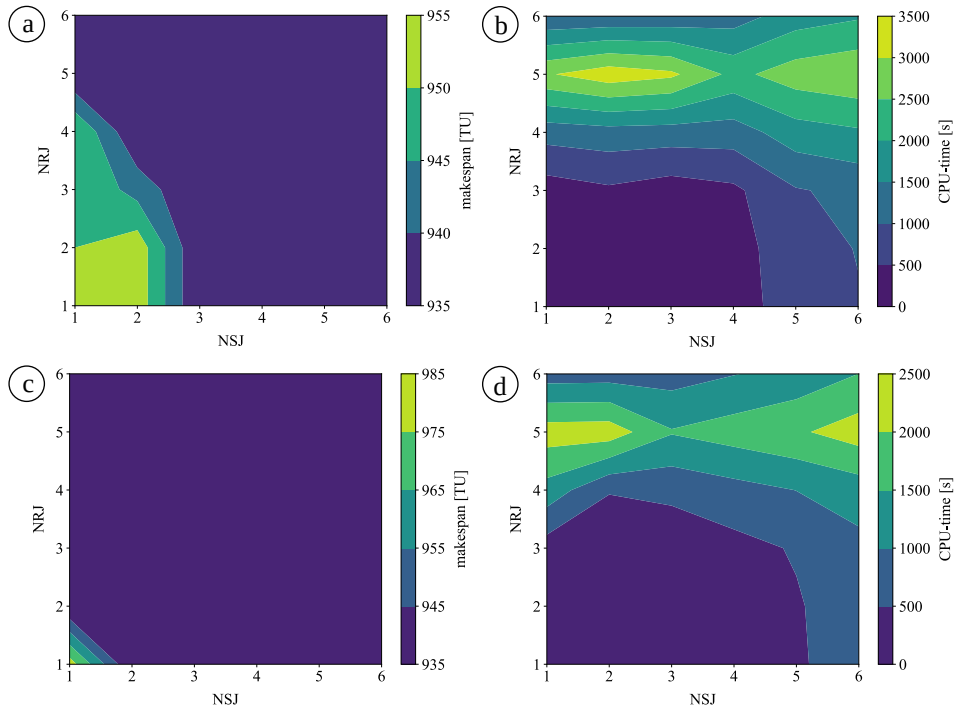

Fig. 5. Results of the grid search for small-scale instance s2 with ordered (top) and unordered (bottom) initial product sequence

As expected, the plots show that the two objectives are competing: A strong decomposition leads to a large makespan and to a small CPU-time and vice versa. The makespan exhibits a weak optimum with a lot of optimal solutions. In contrast, the CPU-time is neither monotonically increasing with *NSJ* nor with *NRJ* but exhibits local optima: For instance, a decomposition into two sub-MIPs in the improvement step ($NRJ = 5$) leads to a larger CPU-time than solving one monolithic MIP ($NRJ = 6$). The CPU-time is more sensitive to changes in NRJ than to changes in NSJ. This is because the number of iterations in the improvement step (controlled by NRJ) can be higher than the iterations in the construction step (controlled by NSJ), as the window can be reset to the beginning during the improvement step while the window is only shifted forward during the construction step.

A comparison of Fig. 5a and Fig. 5c shows that the *initial product sequence* has a significant impact on the set of optimal hyperparameter values for the makespan objective. With an unordered *initial product sequence* (Fig. 5c) only one of the window sizes (in the construction or the improvement step) needs only to be greater than 1 ($NSJ > 1$ or $NRJ > 1$) to achieve the optimal makespan. An ordered product sequence can result in sub-MIPs only comprising products with the same recipe, leading to sub-MIPs with little optimization potential. . If the *initial product sequence* is ordered (Fig. 5a) larger time windows are necessary to achieve the optimal makespan.

5.2 Bayesian Optimization

Fig. 6 shows the behavior of the BO in terms of the sequence of its incumbent solutions for instance s2 based on Fig. 5a and b.

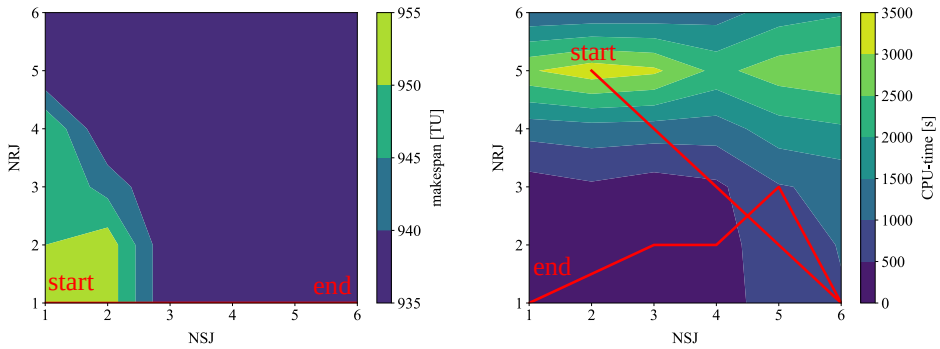

Fig. 6. Behavior of the BO in terms of its incumbent solutions (red lines) for instance s2

The BO is initialized with a single point[8] corresponding to the worst hyperparameter values for both objectives: $NSJ = NRJ = 1$ for the makespan and $NSJ = 2$ and $NRJ = 5$ for the CPU-time. The optimal hyperparameter values are found after the first incumbent solution for the makespan and after the fifth incumbent solution for the CPU-time.

To study the behavior of the BO in a higher-dimensional space, *NFJ* and *MIPgap* are considered as variable hyperparameters in addition to *NSJ* and *NRJ*. Fig. 7 and Fig. 8 show how the incumbents of the objective values evolve over the iterations of the BO. The minimum, the maximum and the mean value over all four instances for

[8] https://automl.github.io/SMAC3/v2.1.0/api/smac.initial_design.html

each iteration are displayed. To make the values of the different instances comparable, they are normalized between 0 and 1.

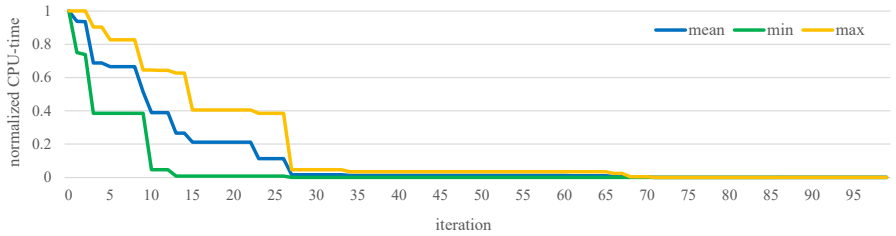

Fig. 7. BO convergence of the CPU-time in the 4D hyperparameter space

As can be seen in Fig. 7, the CPU-time improves mainly in the first 27 iterations. Afterwards it takes more effort – up to 68 iterations – until a better solution is found.

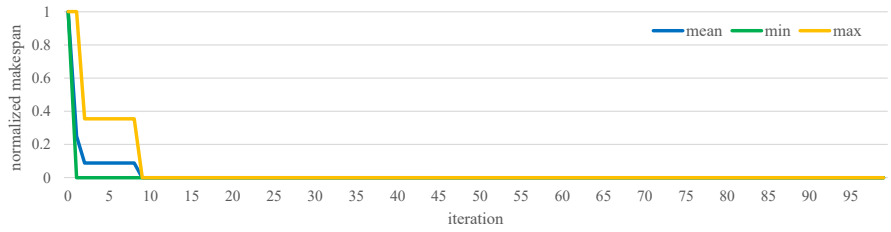

Fig. 8. BO convergence of the makespan in the 4D hyperparameter space

As can be seen in Fig. 8 the makespan improves only in the first 9 iterations. Afterwards no better hyperparameter values can be found.

Table 2 shows the best found hyperparameter values for the two objectives. "Best found" implies that the hyperparameter values are not necessarily optimal and that other hyperparameter values with the same objective values may exist.

Table 2. Best found hyperparameter values for both objectives

	Minimizing makespan				**Minimizing CPU-time**			
Instance	***NSJ***	***NRJ***	***NFJ***	***MIPgap***	***NSJ***	***NRJ***	***NFJ***	***MIPgap***
s1	6	2	2	0%	1	1	1	78%
s2	3	3	1	2%	4	6	3	86%
s3	3	3	1	2%	1	1	1	100%
s4	5	5	5	9%	6	6	1	91%

The best found hyperparameter values depend on both, the instances and the objective. The dependency on the objectives was intuitively expected, since makespan and CPU-time are competing. The dependency on the instance gives reason to assume that there is no one set of hyperparameters that is optimal for all instances.

Minimizing the CPU-time pushes several hyperparameters of the matheuristic (*NSJ*, *NRJ* and *NFJ*) to their boundaries. This is an intuitive result since the CPU-times are

relatively small if either the monolithic MIP is solved ($NSJ = 6, NRJ = 6, NFJ = any$) or the MIP is decomposed as strong as possible ($NSJ = 1, NRJ = 1, NFJ = 1$). In contrast, minimizing the makespan leads to several hyperparameters that are not at their boundaries. This can be explained by the weak optimum which was shown in Fig. 5.

The hyperparameter of the solver (*MIPgap)* is near 0 % if the makespan is minimized and close to 100% if the CPU-time is minimized. These results are also intuitive: Larger optimality gaps leave optimization potential unexplored and lead to solutions of the sub-MIPs with longer makespan, while smaller optimality gaps lead to longer CPU-times for the MIP-solver.

6 Computational Experiments with Large-Scale MIP-Instances

For large-scale MIP-instances the full matheuristic scheme cannot be applied or some sub-MIPs cannot be solved to proven optimality with reasonable computational cost for some hyperparameter values. To deal with this property a computational time limit as an additional termination criterion for the matheuristic is introduced. Consequently, three types of terminations are possible:

1. The computational time limit is not reached, and the matheuristic scheme is finished regularly like for the small-scale instances (construction and improvement steps are executed as described in Section 4.3).
2. The computational time limit is reached during the improvement step, such that the matheuristic scheme cannot be finished regularly. Nevertheless, a feasible schedule results from the construction step.
3. The computational time limit is reached during the construction step, such that the matheuristic scheme cannot be finished regularly. An infeasible schedule results (unless it terminates in the last iteration after a feasible solution of the sub-MIP was found).

If an infeasible schedule results such that no makespan can be calculated for termination-type 3, the following *infeasible solution value* is returned to the BO instead of the makespan value:

$$infeasible\ solution\ value = \left(1 + \frac{\#unscheduled\ products}{\#products}\right) \cdot bigM$$

The degree of infeasibility is reflected by the ratio between the number of unscheduled products and the total number of products. For instance, hyperparameter values leading to only one unscheduled product out of ten are considered better than hyperparameter values that lead to nine unscheduled products out of ten. The value of *bigM* is an upper bound to the makespan, such that the *infeasible solution value* is always greater than the makespan of a feasible solution.

To study this approach, a grid search was performed in analogy to section 5.1. Table 3 shows the main parameters of the four large-scale instances studied. In

comparison to the small-scale instances, the number of products increased from six to ten while the other parameters are adapted accordingly. Some rules are applied to ensure that the MIP-instances are feasible.

Table 3. Main parameters of the large-scale instances

Instance	Number of products	Number of stages	Number of recipes	Number of tanks
l1	10	6, 9	2	30
l2	10	12, 15	2	30
l3	10	5-10	5	30
l4	10	11-17	5	30

According to some pre-studies the computational time limit was set to 700 CPU-s such that the matheuristic terminates for most of the hyperparameter values during the improvement step (termination-type 2). Fig. 9 shows the results of the grid search for the two objectives, makespan (left) and the CPU-time (right), in the *NSJ*-*NRJ*-space for the representative instances l2 (top) and l3 (bottom).

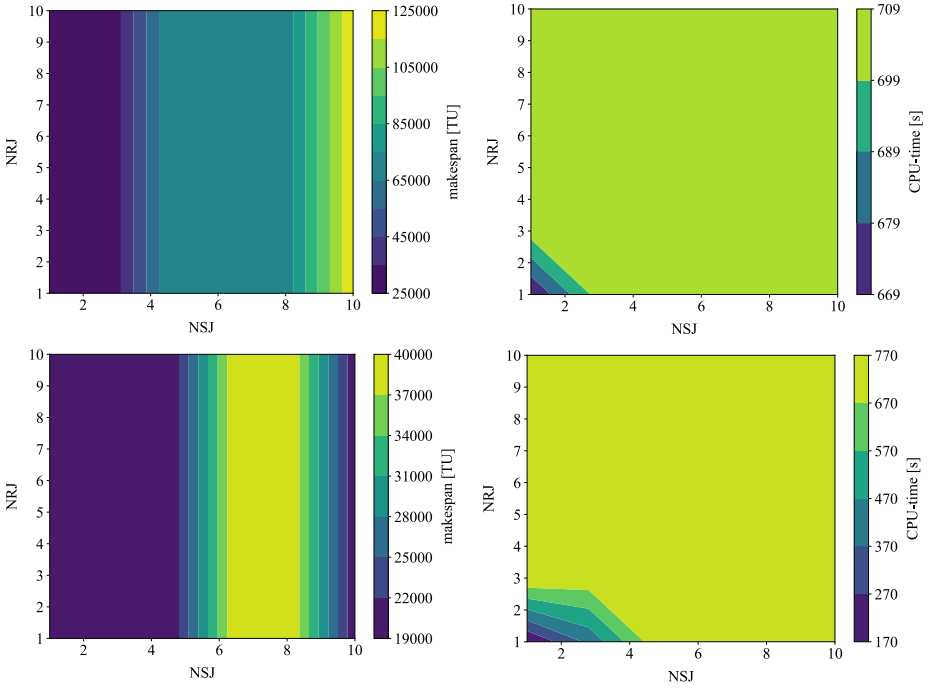

Fig. 9. Results of the grid search for large-scale instances l2 (top) and l3 (bottom)

Only solutions with small values of *NSJ* and *NRJ* belong to termination type 1 with CPU-times less than the time limit (700s). The other solutions are of termination-types 2 or 3 with the CPU-time being equal to the time limit; the additional CPU-time needed to complete the matheuristic scheme is not reflected.

Since for most solutions the improvement phase is not completed (termination-types 2 or 3) the makespan (or the corresponding infeasible solution value) does not depend on *NRJ*. In contrast to the makespan (compare the small-scale instances in Fig. 5), the infeasible solution value does not decrease with *NSJ*. For instance l2 makespan values can be calculated for *NSJ*-values up to 3; for *NSJ*-values from 4 to 10 the *infeasible solution value* is significantly larger than the makespan, depending on the number of unscheduled products. For instance l3 an infeasible solution value is calculated for NSJ-values from 6 to 9, while for NSJ-values up to 5 and – in particular – for 10 a makespan can be calculated.

Comparing the results of the grid search for the large-scale instances (Fig. 9) with those of the small-scale instances (Fig. 5) exhibits the following:

- The CPU-time increases with the values of the hyperparameters *NSJ* and *NRJ* for small-scale as well as for large-scale instances. The main difference is that the values of the CPU-time are cut off at the given time limit for large-scale instances.
- In contrast to the CPU-time, the makespan behaves differently due to the *infeasible solution value* for the large-scale instances. The *infeasible solution value* reflects that for large-scale instances the makespan cannot be treated independently from the computational cost anymore.

The approach described in this section was motivated by the unreasonable computational cost for some hyperparameter values. However, it can also be considered as a way to deal with multiple objectives of an optimization problem, namely constraining one objective (here: computational cost) while optimizing the other (here: MIP-objective).

7 Conclusions and Future Research Directions

Decomposition of large-scale mixed-integer programs (MIPs) into a series of smaller sub-MIPs using matheuristic schemes is a common approach to deal with high computational cost. Matheuristics comprise hyperparameters which steer the decomposition and affect their performance. The tuning of hyperparameters of matheuristics is a black-box optimization problem with an expensive to evaluate objective function; Bayesian optimization is a well-established method for these types of problems. An analysis of related work revealed that hyperparameter tuning of matheuristics by Bayesian optimization has not been studied yet.

A novel algorithmic architecture combining Bayesian optimization, matheuristic, MIP-formulation and MIP standard solver was proposed and applied to hoist scheduling as a case study. The matheuristic comprises four hyperparameters in addition to one hyperparameter of the MIP standard solver. Computational experiments with small- and large-scale MIP-instances expose fundamental properties of the algorithmic approach: The hyperparameter optimization problem exhibits two natural and competing objectives: optimizing the MIP-objective and the computational cost. For small-scale instances, Bayesian optimization converges to optimal hyperparameters for each

objective separately. For large-scale instances the two objectives need to be handled jointly; an approach was presented that constrains the computational-cost-objective and augments the MIP-objective appropriately. Future work aims at validating the findings on the algorithmic approach by extending computational experiments to grid search and Bayesian optimization in the full (5-dimensional) hyperparameter space.

Moreover, future research will aim at two directions: Firstly, the multi-objective aspect of the hyperparameter tuning problem will be examined more deeply. The current approach will be used to approximate the Pareto-set of solutions by varying the computational time limit. This approach will be compared with a "cost-sensitive Bayesian optimization" approach. Cost-sensitive BO determines whether a hyperparameter configuration is promising enough to justify a higher CPU-time, leading to a resource-efficient search process.

Secondly, the current single-MIP-instance approach will be extended to a multi-MIP-instance approach. Instead of considering one instance at a time multiple instances will be considered in two ways:

1. Hyperparameter values will be optimized on average for multiple instances.
2. The previously solved single instances will be used to predict hyperparameter values for the next instance.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Aguirre, A.M., Méndez, C.A., Gutierrez, G., De Prada, C.: An improvement-based MILP optimization approach to complex AWS scheduling. In: Computers & Chemical Engineering. 47, 217–226 (2012). https://doi.org/10.1016/j.compchemeng.2012.06.036.
2. Ilemobayo, J.A. et al.: Hyperparameter Tuning in Machine Learning: A Comprehensive Review. In: Journal of Engineering Research and Reports. 26, 388–395 (2024). https://doi.org/10.9734/jerr/2024/v26i61188.
3. Bischl, B. et al.: Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. In: WIREs Data Mining and Knowledge Discovery. 13, e1484 (2023). https://doi.org/10.1002/widm.1484.
4. Degroote, H., González-Velarde, J.L., De Causmaecker, P.: Applying Algorithm Selection – a Case Study for the Generalised Assignment Problem. In: Electronic Notes in Discrete Mathematics. 69, 205–212 (2018). https://doi.org/10.1016/j.endm.2018.07.027.
5. Mitrai, I., Daoutidis, P.: Taking the human out of decomposition-based optimization via artificial intelligence, Part I: Learning when to decompose. In: Computers & Chemical Engineering. 186, 108688 (2024). https://doi.org/10.1016/j.compchemeng.2024.108688.
6. Mitrai, I., Daoutidis, P.: Taking the human out of decomposition-based optimization via artificial intelligence, Part II: Learning to initialize. In: Computers & Chemical Engineering. 186, 108686 (2024). https://doi.org/10.1016/j.compchemeng.2024.108686.
7. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In: Proceedings of the 7th international conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems.

pp. 186–202. Springer-Verlag, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13520-0_23.
8. Himmich, I., et al.: MPILS: An Automatic Tuner for MILP Solvers. In: Computers & Operations Research. 159, 106344 (2023). https://doi.org/10.1016/j.cor.2023.106344.
9. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: A methodological tour d'horizon. In: European Journal of Operational Research. 290, 405–421 (2021). https://doi.org/10.1016/j.ejor.2020.07.063.
10. Sand, G., Hildebrandt, S., Nunes, S., Chung-On, Y., Franke, M.: Tune Decomposition Schemes for Large-Scale Mixed-Integer Programs by Bayesian Optimization. In: Systems & Control Transactions. 4 (2025) (to be published)