Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

pycombina: An Open-Source Tool for Solving Combinatorial Approximation Problems Arising in Mixed-Integer Optimal Control ⋆

Adrian Bürger [*,**] **Clemens Zeile** [***] **Mirko Hahn** [***]
Angelika Altmann-Dieses [*] **Sebastian Sager** [***]
Moritz Diehl [**,****]

[*] *Institute of Refrigeration, Air-Conditioning and Environmental Engineering, Karlsruhe University of Applied Sciences, Germany.*
[**] *Systems Control and Optimization Laboratory, Department of Microsystems Engineering, University of Freiburg, Germany.*
[***] *MathOpt Group, Institute for Mathematical Optimization, Faculty of Mathematics, Otto-von-Guericke University Magdeburg, Germany.*
[****] *Department of Mathematics, University of Freiburg, Germany.*

Abstract: Application of Model Predictive Control (MPC) for nonlinear switched systems often leads via discretization to Mixed-Integer Non-Linear Programs (MINLPs), which in a real-time setting can be solved approximately using a dedicated decomposition approach. One stage within this approach is the solution of a so-called Combinatorial Integral Approximation (CIA) problem, which is a Mixed-Integer Linear Program (MILP) that can be solved either approximately or to global optimality. The applicability of these decomposition methods depends strongly on efficient implementations, while many practical applications also require the consideration of a variety of additional and complex combinatorial constraints. In this work, we provide a comprehensive introduction to the open-source software tool pycombina, which enables users to automatically formulate CIA problems and provides methods for fast and efficient solution of these problems. In a case study, the usage of the tool is exemplified for input data from a real-life MPC application.

Keywords: Nonlinear predictive control, Control of switched systems, Numerical methods for optimal control

1. INTRODUCTION

For nonlinear switched systems, application of Model Predictive Control (MPC) typically leads to Mixed-Integer Optimal Control Problems (MIOCPs), cf. Kirches (2011). After application of direct methods, this gives rise to Mixed-Integer Non-Linear Programs (MINLPs) that need to be solved on real-time suitable time scales. An approach based on partial outer convexification and relaxation that facilitates an approximate but real-time suitable solution of such MINLPs is presented by Sager et al. (2012). This decomposition approach consists of solving a sequence of subproblems of which each one is easier to solve than the original MINLP. Successful applications of this method have been presented, e.g., in the area of renewable energy systems (Bürger et al., 2019b, 2020) and traffic light optimization (Göttlich et al., 2017).

One of these subproblems can be considered as a combinatorial approximation problem and is the so-called Combinatorial Integral Approximation (CIA) problem. This problem is a Mixed-Integer Linear Program (MILP) that can be solved either by standard MILP solvers, a tailored Branch-and-Bound (BnB) scheme (Sager et al., 2011) or approximately by Sum Up Rounding (SUR) (Sager et al., 2012). In many practical applications, additional combinatorial constraints such as maximum switching and dwell time constraints or limitations in transitions from one mode to another need to be considered, which can have a major impact on the complexity of formulation and solution of a CIA problem.

In this work, we provide a comprehensive introduction to the open-source software tool pycombina, which is designed to simplify the formulation and solution of CIA problems. After a presentation of the available solution methods and the possibilities for enforcing additional combinatorial constraints, utilization of the tool is exemplified for input data coming from the real-life MPC operation

⋆ A. Bürger, A. Altmann-Dieses and M. Diehl received funding from INTERREG V Upper Rhine, project ACA-MODES. S. Sager, M. Hahn and C. Zeile received funding from the European Research Council (ERC), grant agreement No 647573, from German Research Foundation – 314838170, GRK 2297 MathCoRe, through Priority Programme 1962, grant KI1839/1-1, and from German Federal Ministry of Education and Research, program "Mathematics for Innovations", grant P2Chem. M. Diehl received funding from the German Federal Ministry for Economic Affairs and Energy (BMWi) via DyConPV (0324166B) and by DFG via Research Unit FOR 2401.

of a nonlinear switched thermal energy supply system (Bürger et al., 2019a). We compare the performance of the available methods under different problem formulations and provide application guidelines.

The remainder of this work is organized as follows. Section 2 describes the decomposition approach for approximate solution of MIOCPs. Then, the pycombina software tool and its functionalities are presented in Section 3, followed by a case study in Section 4. Section 5 concludes this work and presents future work suggestions.

2. DECOMPOSITION ALGORITHM FOR APPROXIMATE SOLUTION OF MIOCPs

2.1 Problem description

We consider an MIOCP on a given time horizon $t \in [t_0, t_\mathrm{f}]$ for a switched dynamic system with differential states $x(t) \in \mathbb{R}^{n_x}$, continuous controls $u(t) \in \mathbb{R}^{n_u}$ and binary controls $b(t) \in \{0,1\}^{n_b}$ with $n_x, n_u, n_b \in \mathbb{N}$ as in

$$\min_{x(\cdot),u(\cdot),b(\cdot)} \quad \int_{t_0}^{t_\mathrm{f}} L\big(x(t),u(t),b(t)\big)\mathrm{d}t + M\big(x(t_\mathrm{f})\big) \tag{1a}$$

$$\text{s.t. for } t \in [t_0, t_\mathrm{f}]:$$

$$\dot{x}(t) = f_0\big(x(t),u(t)\big) + \sum\nolimits_{i=1}^{n_b} b_i(t)\cdot f_i\big(x(t),u(t)\big), \tag{1b}$$

$$r_\mathrm{lb}(t) \le r\big(x(t),u(t),b(\cdot)\big) \le r_\mathrm{ub}(t), \tag{1c}$$

$$\sum\nolimits_{i=1}^{n_b} b_i(t) = 1, \tag{1d}$$

$$x(t_0) = x_0, \quad x(t) \in \mathcal{X}, \quad u(t) \in \mathcal{U}. \tag{1e}$$

The objective (1a) consists of the sum of a continuous time Lagrangian cost functional $L(\cdot)$ and a Mayer term $M(\cdot)$. We assume that the system is described by nonlinear Ordinary Differential Equations (ODEs) in partial outer convexified form, i.e., the ODE right hand side is given by a sum of functions f_0 and f_i, $i = 1,\dots,n_b$, so that the binary controls b appear affinely. In function $r(\cdot)$, different kind of path constraints such as maximum switching constraints, dwell time constraints etc. with possibly time-dependent lower bounds $r_\mathrm{lb}(t)$ and upper bounds $r_\mathrm{ub}(t)$ can be specified. We write $b(\cdot)$ to indicate that combinatorial constraints can be coupled over time. Constraint (1d) contains the so-called Special Order Set 1 (SOS1) condition which ensures that exactly one binary control is active at any time.

2.2 CIA decomposition algorithm

Application of direct methods such as direct multiple shooting (Bock and Plitt, 1984) or direct collocation (Tsang et al., 1975) for solution of MIOCP (1) results in an MINLP with a time grid $T_N = \{t_0 < t_1 < \ldots < t_N = t_\mathrm{f}\}$ that contains $N+1$ discrete time points. For solution of such problems and especially within MPC applications, the CIA decomposition algorithm by Sager et al. (2012) can be used whose main steps are given in Alg. 1. First, a relaxed MINLP with binary constraint dropped is solved to obtain a relaxed solution $q_k \in [0,1]^{n_b}$ for the binary controls b_k, $k = 0,\dots,N-1$ [1]. This problem

[1] Note that, as customary in optimal control, no control inputs are associated with the final time point t_N.

Input : Discretized MIOCP instance with grid T_N, initial guesses for x, u, b.
Output: (Local) optimal variables x^*, u^*, b^*, with objective $\mathcal{L}^* = \mathcal{L}(x^*, u^*, b^*)$.
1 Solve relaxed MINLP ($\mathrm{NLP_{rel}}$) $\rightarrow$ x, u, q, $\mathcal{L}_\mathrm{rel}$;
2 Solve CIA problem for $q \rightarrow p$;
3 Solve MINLP with $b := p$ fixed ($\mathrm{NLP_{bin}}$) $\rightarrow$ x, u, $\mathcal{L}_\mathrm{bin}$;
4 **return**: $(x^*, u^*, b^*, \mathcal{L}^*) = (x, u, p, \mathcal{L}_\mathrm{bin})$;

Algorithm 1: CIA decomposition algorithm.

is a Non-Linear Program (NLP) further referred to as $\mathrm{NLP_{rel}}$. Afterwards, a binary approximation $p_k \in \{0,1\}^{n_b}$ of q_k, $k = 0,\dots,N-1$ is obtained by solving a so-called CIA problem. Afterwards, the MINLP is solved again with binary controls $b_k := p_k$ fixed, which is again an NLP, for optimization of continuous controls and states considering the obtained binary solution.

This approach is justified by a theorem which states that the optimal MIOCP solution can be arbitrarily well approximated in the absence of combinatorial constraints with the CIA decomposition by refining the discretization grid (Sager et al., 2012). For further details on the algorithm and its generalizations we refer to Hahn et al. (2019).

Within this work, the focus lies on the solution of the CIA problem in the second step of Alg. 1, which is described in the following in more detail.

2.3 The CIA problem

After solving $\mathrm{NLP_{rel}}$ in Alg. 1, the idea is to find binary values $p_k \in \{0,1\}^{n_b}$, $k = 0,\dots,N-1$ that minimize the accumulated difference to the relaxed values q in the $\|\cdot\|_\infty$-norm for all N discrete time intervals and n_b binary controls. This is expressed by the following MILP referred to as the CIA problem:

$$\min_{p,\theta} \quad \theta \tag{2a}$$

$$\text{s.t.} \quad \text{for } k = 0,\dots,N-1: \tag{2b}$$

$$\theta \ge \pm\sum\nolimits_{j=0}^{k} \big(q_{j,i} - p_{j,i}\big)\Delta_j, \ i = 1,\dots,n_b, \tag{2c}$$

$$1 = \sum\nolimits_{i=1}^{n_b} p_{k,i}, \tag{2d}$$

$$p_k \in \{0,1\}^{n_b}, \tag{2e}$$

$$\text{(optional combinatorial constraints)}, \tag{2f}$$

where θ denotes an introduced auxiliary variable and Δ_j is the length of the j^th discretization interval. We specify optional combinatorial constraints that are added to (2) in Section 3.3.

3. THE PYCOMBINA SOFTWARE TOOL

An application of the decomposition algorithm requires the availability of efficient methods and implementations for solution of the CIA problem, as the problem size can be huge (e. g., in Partial Differential Equation (PDE)-constrained optimal control as in Manns and Kirches (2019)) and the solving process can be time-critical (e. g., in an MPC setting). At the same time, consideration

of combinatorial constraints is often required for realistic modeling within the MIOCPs. In the following, we describe the available solution methods and constraint definition facilities included in the open-source software tool *pycombina* (Bürger et al., 2019b,c).

3.1 Focus and design of the software tool

As powerful frameworks and solvers for formulation and solution of NLPs exist, e. g., CasADi (Andersson et al., 2019) and IPOPT (Wächter and Biegler, 2006), the focus of pycombina lies directly on the formualtion and solution of the CIA problem (2) arising in the second stage of the decomposition algorithm. The tool accepts time grids and the relaxed binary solutions as numerical inputs and provides the approximated binary trajectories resulting from solution of (2) in the same form. This facilitates flexible use of the tool in combination with different modeling frameworks and solvers used within the first and third stage of the decomposition algorithm.

While major parts of the tool, including the user interface, are developed in Python to simplify use and integration of the methods in existing projects, performance-critical parts are either implemented in C++ and interfaced using pybind11 (Jakob et al., 2017) or rely on state-of-the-art numerical solvers.

3.2 Available options for solving the CIA problem

Sum Up Rounding (SUR): SUR has been established as a widely-used algorithm to generate approximate binary solutions of (2) due to its simplicity and its well approximating binary control trajectories. The rounding scheme reads recursively for $k = 0, \dots, N-2,\ i = 1, \dots, n_b$ as

$$p_{k,i} := \begin{cases} 1, \text{ if } i = \underset{i=1,\dots,n_b}{\arg\max} \sum_{j=0}^{k+1} q_{j,i}\Delta_j - \sum_{j=0}^{k} p_{j,i}\Delta_j. \\ 0, \text{ else.} \end{cases}$$

The usage of SUR is recommended for fast generation of (approximate) solutions of (2) if additional (combinatorial) constraints do not need to be considered.

General MILP solver: An exact and globally optimal solution of (2) can be obtained through application of a general MILP solver. For this purpose, pycombina provides the possibility for automated set up of CIA problems for solution by Gurobi (Gurobi Optimization, LLC, 2019) [2]. The automated setup of the corresponding MILP, with or without additional combinatorial constraints, is realized via the Python interface of Gurobi. This option is especially useful within rapid prototyping and for addition and testing of new constraint types for a CIA problem. For real-time applications, however, it can be favorable to apply tailored solution algorithms, as presented in the following.

Tailored BnB solver: For solution of (2) a tailored BnB algorithm can be used, cf. Sager et al. (2011) and Jung (2013), which branches forward in time and exploits that an evaluation of the objective function up to the current grid interval yields a valid lower bound due to the maximization operator over all intermediate intervals in the objective function. For further details we refer to Jung (2013). The BnB scheme has been shown to have run time advantages for solution of (2) in comparison to general MILP solvers (Jung, 2013; Bürger et al., 2019b). Within pycombina, we equip the method with options to limit the maximum number of iterations and to choose from different node selection strategies, e.g., depth- or best-first-search.

[2] For using this functionality, a separate installation of the Gurobi solver and an adequate Gurobi license are required.

3.3 Available optional combinatorial constraints

In the following, we present important types of additional constraints of the CIA problem for which the tool provides automatic setup facilities.

Maximum switching constraints: Such constraints allow us to limit the maximum number of mode switches on the defined time horizon. This is imposed by

$$\sum\nolimits_{k=0}^{N-2} \left| p_{k+1,i} - p_{k,i} \right| \leq \sigma_{i,\max}, \quad i = 1, \dots, n_b, \tag{3}$$

where $\sigma_{i,\max} \in \mathbb{N}^+$ denotes the maximum number of allowed switches per mode.

Minimum dwell time constraints: Minimum time spans $t_{i,\min} \in \mathbb{R}_0^+,\ i = 1, \dots, n_b$ can be set for which a mode i must remain active (inactive) once it has been switched on (off). These requirements would read for $j = 0, \dots, N-2,\ k \in \{l \mid t_l \in T_N \cap [t_j, t_j + t_{i,\min})\}$ as

$$p_{k,i} \geq p_{j+1,i} - p_{j,i}, \quad i = 1, \dots, n_b, \tag{4}$$

$$1 - p_{k,i} \geq p_{j,i} - p_{j+1,i}, \quad i = 1, \dots, n_b, \tag{5}$$

Note that the j^{th} interval is given by $[t_j, t_{j+1})$. The constraint class (4) is often referred to as min-up time constraints and the constraint class (5) as min-down time constraints. It is possible with pycombina to specify different time spans $t_{i,\min}$ for min-up and min-down times of each mode.

Total maximum uptime constraints: We limit the total activation time $t_{i,\max} \in \mathbb{R}_0^+,\ i = 1, \dots, n_b$ of a mode i over the defined time horizon as in

$$t_{i,\max} \geq \sum\nolimits_{k=0}^{N-1} p_{k,i}\Delta_k, \quad i = 1, \dots, n_b. \tag{6}$$

Allowed mode transition constraints: We define which mode can be followed and/or preceded by another mode, e. g., for modeling the sequence of a gear shift, cf. Robuschi et al. (2019). Let $\mathcal{P}_i \subset \{1, \dots, n_b\}$ denote the allowed modes that can be activated directly after mode i has been active. Then, this constraint can be expressed for $k = 1, \dots, N-1$ as

$$1 \geq p_{k-1,i} + \sum\nolimits_{m \notin \mathcal{P}_i} p_{k,m}, \quad i = 1, \dots, n_b. \tag{7}$$

Restrict allowed modes for defined time periods: This permits us to prohibit activation of certain modes during defined time periods. Mode i may be trivially excluded from activation on a specified interval $[\tau_1, \tau_2] \subset [t_0, t_\mathrm{f}]$ via

$$p_{k,i} = 0, k \in \{l = 0, \dots, N-1 \mid [t_l, t_{l+1}] \cap [\tau_1, \tau_2] \neq \emptyset\}. \tag{8}$$

3.4 Usage of the Python interface

The basic usage of the Python interface of pycombina is exemplified in Fig. 1. A CIA problem in pycombina is

```
import pycombina

T = [0.0, 1.0, 2.5, 4.0, 6.0, 8.0]
q = [[0.1, 0.9, 0.9, 0.4, 0.2],
     [0.9, 0.1, 0.1, 0.6, 0.8]]

ba = pycombina.BinApprox(T, q)
ba.set_min_up_times([3, 4])

bnb = pycombina.CombinaBnB(ba)
opts = {"max_cpu_time": 30.0}
bnb.solve(**opts)
```

Fig. 1. Sample use of the pycombina Python interface. The set up of a `BinApprox` object requires a time grid, here T, and a matrix of the relaxed control values q, here with $N = 5$ and $n_b = 2$. We require min-up times of 3, respectively 4, time units for the available modes by using the function `set_min_up_times` so that we alter the CIA problem by modifying the `BinApprox` object. Finally, we solve the problem by applying the BnB solver with a time limit of 30 seconds.

defined using the `BinApprox` class, which is instantiated by passing at least numerical data for the time grid T_N and relaxed binary solution q. Afterwards, optional combinatorial constraints of the problem and other settings can be specified. The problem can then be passed to one of the solver classes `CombinaSUR`, `CombinaBnB`, or `CombinaMILP`, which can then be used to solve the specified problem so that the desired binary control values p are constructed.

4. CASE STUDY

In this section, we demonstrate and evaluate selected methods and features of pycombina for relaxed binary solutions coming from the real-life MPC operation of a Solar Thermal Climate System (STCS) for building climate control located at Karlsruhe University of Applied Sciences. Core element of this system shown in Fig. 2 is an Adsorption Cooling Machine (ACM) that can operate in two different modes: in Adsorption Cooling (AC) mode ($b_{\text{ac}} = 1$), it utilizes hot water from a high temperature storage to cool down a low temperature storage; in Free Cooling (FC) mode ($b_{\text{fc}} = 1$), it utilizes the recooling tower on the roof of the building to directly cool down the low temperature storage at the ambient. For all intervals, at most one of these modes can be active, i. e.,

$$b_{k,\text{ac}} + b_{k,\text{fc}} \leq 1, \quad k = 0, \dots, N-1. \tag{9}$$

Frequent switching of the modes should be avoided due to hardware restrictions and unmodeled ramping phases. A detailed description of the STCS, the employed system model and MPC strategy (which is a parallelized variant of Alg. 1), and the utilized methods and software is given in Bürger et al. (2019a) and Bürger et al. (2019b).

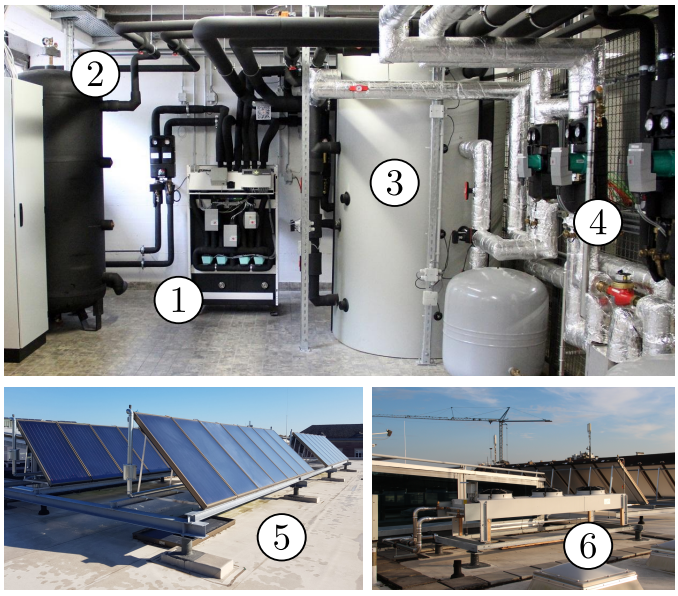

Fig. 2. Depiction of the STCS: (1) ACM, (2) low temperature storage, (3) high temperature storage, (4) pumps, (5) solar collector arrays, (6) recooling tower.

In the following, we compare the performance of the implemented solution methods for several variants of the basic CIA problem for different relaxed binary solutions. Also, we evaluate the applicability of the obtained binary trajectories within an MPC setting and provide usage advice for the pycombina software tool.

All computations are conducted on a Fujitsu P920 Desktop PC with an Intel Core i5-4570 3.20 GHz CPU and 16 GB RAM running Debian 9, using Python 3.6, gcc 6.3, Gurobi 8.1.1, and pycombina 0.3.1.

4.1 Solution of the basic CIA problem

The data set shown in Fig. 3 is a relaxed binary solution q for the STCS that originated from the solution of a relaxed problem NLP_{rel} on a morning in September 2019. The data is given on a non-equidistant time grid with a total of $N = 91$ discrete time intervals on a horizon of 24 h. The relaxed solution indicates that activation of the AC mode is mainly requested between 08:00 and 18:00 and activation of the FC mode mainly between 20:00 and 08:00 of the next day.

Since pycombina expects that $\sum_{i=1}^{n_b} q_{k,i} = 1$, $k = 0, \dots, N-1$, an additional binary control b_{off} must be introduced with $f_{\text{off}}(\cdot) = 0$ whose relaxed solution takes the values

$$q_{k,\text{off}} = 1 - (q_{k,\text{ac}} + q_{k,\text{fc}}), \quad k = 0, \dots, N-1. \tag{10}$$

In the remainder of this section, however, the values of this trivial control are not explicitly shown.

First, the performance of the available solution methods is compared for the solution of a CIA problem for the data

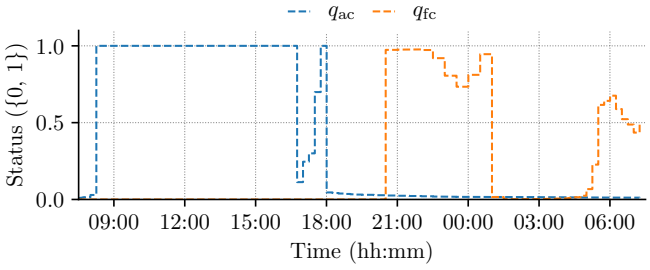

Fig. 3. Depiction of the relaxed solution.

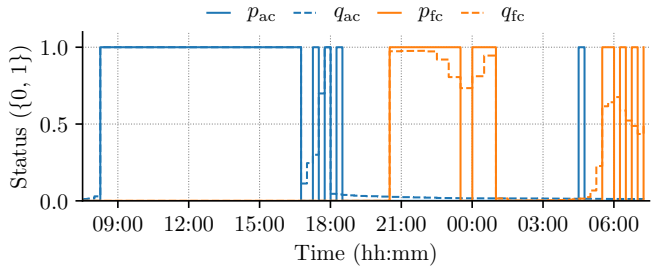

Fig. 4. Depiction of the CIA solution using SUR.

Table 1. Runtimes and objective values without additional constraints.

	Runtime (s)	Objective of (2)
SUR	$1.04 \cdot 10^{-3}$	693.54
BnB	$8.96 \cdot 10^{1}$	693.54
MILP	$8.45 \cdot 10^{-1}$	693.54

shown in Fig. 3 in the absence of additional combinatorial constraints. Table 1 shows that the objective values obtained by the methods are identical, which is the global optimal objective of the problem as the solutions obtained via the BnB and MILP solver are globally optimal. This shows that in this particular case, SUR was able to generate the global optimal solution as well, however, it should be noted that SUR only constructs a guaranteed optimal solution in the absence of additional constraints and for $n_b = 2$ (Sager et al., 2011).

Fig. 4 shows the solution p of the CIA problem obtained using SUR, Fig. 5 and Fig. 6 show the binary solution constructed by the BnB solver and MILP solver, respectively. It can be observed that the trajectories obtained by the different methods differ, which is due to the fact that the global optimal solution of a CIA problem might not be unique, cf. Jung (2013). A comparison of the solution times given in Table 1 shows that especially the BnB method does not perform well in this setting. Since SUR is able to generate a global optimal solution without the need to solve an actual optimization problem, SUR can be a good choice for this basic setting. It may appear, however, that the relaxed solution exhibits a special structure that results in underperforming of the SUR binary solutions, cf. Jung (2013).

All obtained solutions show frequent switching for both modes, which should be avoided for the considered application. As soon as more complex settings such as maximum switching or dwell time constraints need to be included,

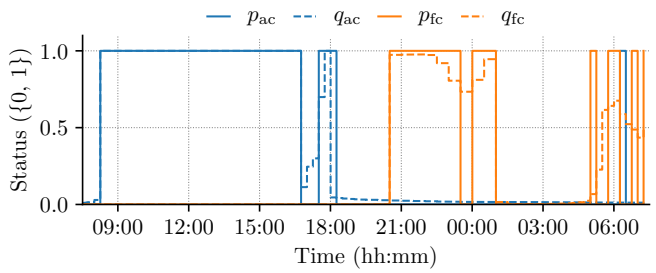

Fig. 5. Depiction of the CIA solution using BnB solver.

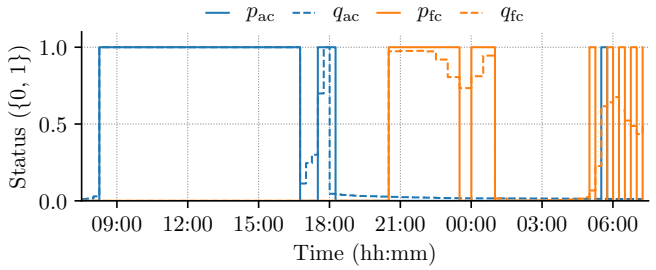

Fig. 6. Depiction of the CIA solution using MILP solver.

Table 2. Runtimes and objective values considering maximum switching constraints.

	Runtime (s)	Objective of (2)
BnB	$5.71 \cdot 10^{-2}$	1495.61
MILP	$7.98 \cdot 10^{-1}$	1495.61

however, SUR becomes inapplicable as the method cannot consider such constraints. In such cases, the BnB or MILP solver must be utilized, as shown in the upcoming section.

4.2 Consideration of maximum switching constraints

One possibility to avoid frequent switching of the ACM is to utilize the maximum switching constraint (3) introduced in Section 3.3. Fig. 7 shows the binary solution of CIA with $\sigma_{\mathrm{ac,max}} = 2$ and $\sigma_{\mathrm{fc,max}} = 4$ using the BnB solver, Table 2 shows the achieved runtimes and objective values using BnB and the MILP solver.

While the optimal objective values increase due to the reduced degrees of freedom for the solvers, frequent switching is successfully avoided. At the same time, the solution time for the BnB solver decreases. This is due to the fact that this solver can actively consider such constraints during the solution process and take according "shortcuts" once the maximum amount of switches for construction of a solution candidate has been spent, cf. Jung (2013).

4.3 Consideration of minimum dwell time constraints

Another way to reduce frequent switching is to ban activation times that are too short to meet the requirements of a controlled component or machinery. For this, min-up time constraints for b_{ac} and b_{fc} as in (4) can be introduced, so that once activated, a mode must remain active at least for a defined duration. Fig. 8 shows the binary solution

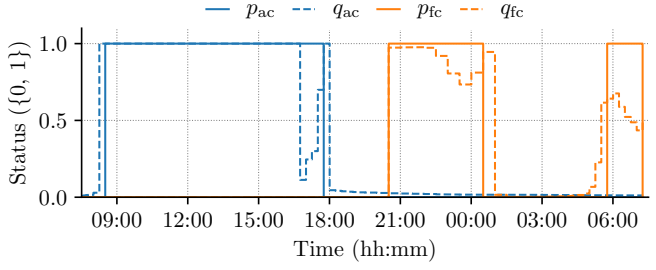

Fig. 7. Depiction of the CIA solution using BnB solver and considering maximum switching constraints.

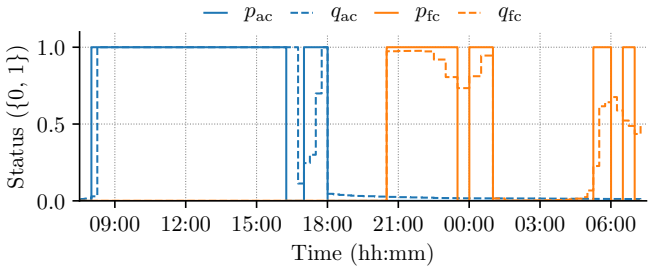

Fig. 8. Depiction of the CIA solution using BnB solver and considering dwell time constraints.

Table 3. Runtimes and objective values considering dwell time constraints.

	Runtime (s)	Objective of (2)
BnB	$1.0 \cdot 10^{-1}$	1051.8
MILP	$8.6 \cdot 10^{-1}$	1051.8

of CIA with $t_{\text{ac,min-up}} = 1\,\text{h}$ and $t_{\text{fc,min-up}} = 0.5\,\text{h}$ using the BnB solver, Table 3 shows the achieved runtimes and objective values using BnB and the MILP solver.

Compared to the results given in Table 2, the objective values have decreased. While short activation times of machinery are successfully avoided, it can be seen in Fig. 8 that the introduced constraints led to more frequent switching with short down times. If desired, such effects could now be counteracted by introduction of additional min-down time constraints as in (5) and/or by a combination of the introduced min-up time constraints with maximum switching constraints.

4.4 Solver performances under multiple constraints

Especially in the presence of multiple additional constraints, their active consideration within the tailored BnB algorithm can facilitate reduced solution times for the corresponding CIA problem. To illustrate this, we compare the runtimes of the BnB solver r_{BnB} and the MILP solver r_{MILP} for the solution of a set of CIA problems $\mathcal{S}_{\text{CIA}}$ that consists of 500 CIA problems under simultaneous consideration of maximum switching constraints $\sigma_{\text{ac,max}} = \sigma_{\text{fc,max}} = 4$, min-up time constraints $t_{\text{ac,min-up}} = 1\,\text{h}$ and $t_{\text{fc,min-up}} = 0.25\,\text{h}$, and min-down time constraints $t_{\text{ac,min-down}} = 0.5\,\text{h}$ and $t_{\text{fc,min-down}} = 0.25\,\text{h}$ based on 500 arbitrary relaxed binary solutions which occured during MPC operation of the STCS in September 2019, including the set shown in Fig. 3. Each problem from $\mathcal{S}_{\text{CIA}}$ is solved individually and solvers are not warm-started.

Fig. 9 shows two box plots that illustrate the runtimes for solution of $\mathcal{S}_{\text{CIA}}$ achieved by the BnB and MILP solver. Each problem was solved to global optimality with a maximum runtime of approx. 18 s for the BnB and approx. 32.5 s for the MILP solver. We observe from the plot that the runtime of the BnB solver is typically lower than the runtime of the MILP method.

This aspect is further investigated in Fig. 10, which illustrates the accumulated occurrences of the differences Δ_r between the runtime of the BnB solver r_{BnB} and the MILP solver r_{MILP} as in

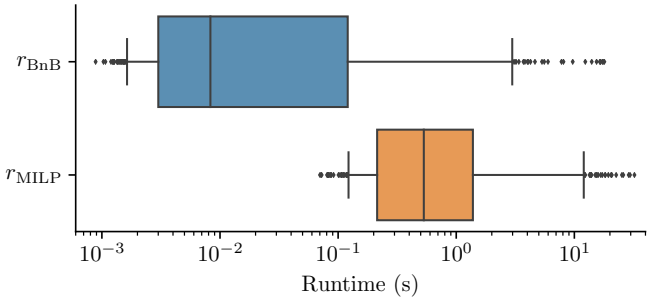

Fig. 9. BnB and MILP solver runtimes for solution of $\mathcal{S}_{\text{CIA}}$. The boxes range from the 25 % to the 75 % quartile, the whiskers mark the 5 % and 95 % percentile.

$$\Delta_r \leq r_{\text{BnB}} - r_{\text{MILP}} \tag{11}$$

for each instance of $\mathcal{S}_{\text{CIA}}$. We represent these occurrences as fraction of all instances. The plot shows that, while the BnB solver was able to solve one CIA instance more than 30 s faster than the MILP solver, it was at most less than 8 s slower for another problem instance. Overall, the BnB solver runtime was lower than the MILP solver runtime for more than 96 % of the problem instances.

This comparison illustrates how the presence of combinations of additional contraints can lead to improved solution speed of the BnB solver, which can render the solver favorable for this case especially within time-critical applications, such as MPC. We notice, however, that we used the default settings of Gurobi for the benchmark computations, whereas a customization of the solver parameters could possibly lead to a considerable runtime improvement for specific instances.

4.5 Restriction of allowed modes for defined time periods

In case min-up time constraints become relevant in an MPC setting, it must be ensured that such restrictions are considered also in between subsequent MPC iterations, i. e., that a machine or mode activated in a previous step must remain active for a minimum amount of time, possibly even independent from the obtained relaxed binary solution.

For this, constraints according to (8) can be introduced to restrict the allowed modes during the corresponding time span accordingly. Fig. 11 shows the binary solution of CIA with $\sigma_{\text{ac,max}} = 4$ and $\sigma_{\text{fc,max}} = 4$ and with

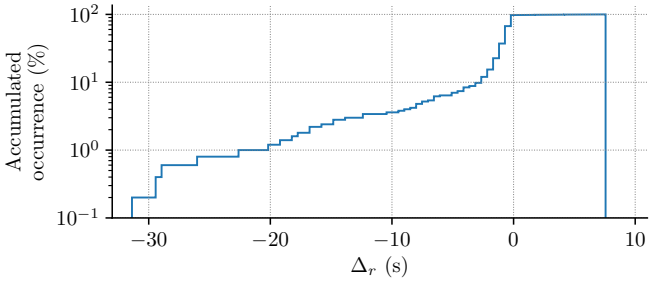

Fig. 10. Accumulated occurrences (in %) of the runtime differences Δ_r between BnB to MILP solver according to (11) for each problem instance of $\mathcal{S}_{\text{CIA}}$.

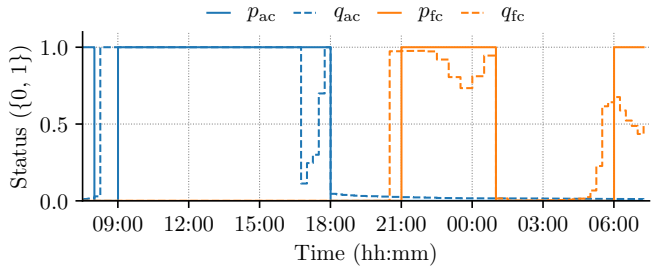

Fig. 11. Depiction of the CIA solution using BnB solver, maximum switching constraints and restricted mode activation.

$$p_{\mathrm{ac},k} = 1, \quad \text{for } k \in \{l \mid t_l \in [0, 0.5\,\mathrm{h}]\}, \tag{12}$$

constructed by the BnB solver. It can be observed that the AC mode of the machine is now forced to stay active for the first 0.5 h of the time horizon. Accordingly, the machine is deactivated after that period and re-activated on a later time point compared with the previous setting.

5. CONCLUSIONS AND FUTURE WORK

In this work, we provided a comprehensive introduction to the open-source software tool pycombina for formulation and solution of CIA problems. We exemplified the performance of the implemented methods on data sets from the real-life operation of a thermal energy supply system and gave best practice recommendations.

Future work should focus on the implementation of possibilities to include information from $\mathrm{NLP}_{\mathrm{rel}}$ in the CIA step, such as information on the influence of binary control decisions on state constraint compliance. Further, migration of sophisticated rounding schemes for combinatorial constraints as in Sager and Zeile (2019) should be considered.

REFERENCES

Andersson, J.A.E., Gillis, J., Horn, G., Rawlings, J.B., and Diehl, M. (2019). CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1), 1–36. doi:10.1007/s12532-018-0139-4.

Bock, H. and Plitt, K. (1984). A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proceedings Volumes*, 17(2), 1603–1608. doi:10.1016/S1474-6670(17)61205-9.

Bürger, A., Bohlayer, M., Hoffmann, S., Altmann-Dieses, A., Braun, M., and Diehl, M. (2020). A whole-year simulation study on nonlinear mixed-integer model predictive control for a thermal energy supply system with multi-use components. *Applied Energy*, 258, 114064. doi:10.1016/j.apenergy.2019.114064.

Bürger, A., Bull, D., Sawant, P., Bohlayer, M., Klotz, A., Beschütz, B., Altmann-Dieses, A., Braun, M., and Diehl, M. (2019a). Experimental operation of a solar-driven climate system with thermal energy storages using mixed-integer nonlinear MPC. URL `http://www.optimization-online.org/DB_HTML/2019/10/7422.html`. Accessed Oct. 11, 2019.

Bürger, A., Zeile, C., Altmann-Dieses, A., Sager, S., and Diehl, M. (2019b). Design, implementation and simulation of an MPC algorithm for switched nonlinear systems under combinatorial constraints. *Journal of Process Control*, 81, 15–30. doi:10.1016/j.jprocont.2019.05.016.

Bürger, A., Zeile, C., Hahn, M., Altmann-Dieses, A., Sager, S., and Diehl, M. (2019c). pycombina - Solving binary approximation problems in Python. `https://github.com/adbuerger/pycombina`. Accessed Oct. 29, 2019.

Göttlich, S., Potschka, A., and Ziegler, U. (2017). Partial outer convexification for traffic light optimization in road networks. *SIAM Journal on Scientific Computing*, 39(1), B53–B75. doi:10.1137/15M1048197.

Gurobi Optimization, LLC (2019). Gurobi optimizer reference manual. URL `http://www.gurobi.com`. Accessed Oct. 29, 2019.

Hahn, M., Kirches, C., Manns, P., Sager, S., and Zeile, C. (2019). Decomposition and approximation for PDE-constrained mixed-integer optimal control. In M.H. et al. (ed.), *SPP1962 Special Issue*. Birkhäuser. (accepted).

Jakob, W., Rhinelander, J., and Moldovan, D. (2017). pybind11 – seamless operability between C++11 and Python. URL `https://github.com/pybind/pybind11`. Accessed Oct. 29, 2019.

Jung, M. (2013). *Relaxations and Approximations for Mixed-Integer Optimal Control*. Ph.D. dissertation, Interdisciplinary Center for Scientific Computing, Heidelberg University. doi:10.11588/heidok.00016036.

Kirches, C. (2011). *Fast Numerical Methods for Mixed-Integer Nonlinear Model-Predictive Control*. Vieweg+Teubner Verlag. doi:10.1007/978-3-8348-8202-8.

Manns, P. and Kirches, C. (2019). Multi-dimensional sum-up rounding using hilbert curve iterates. *PAMM*, 19(1), e201900065. doi:10.1002/pamm.201900065.

Robuschi, N., Zeile, C., Sager, S., Braghin, F., and Cheli, F. (2019). Multiphase mixed-integer nonlinear optimal control of hybrid electric vehicles. URL `http://www.optimization-online.org/DB\_HTML/2019/05/7223.html`. Accessed Oct. 1, 2019.

Sager, S., Bock, H., and Diehl, M. (2012). The Integer Approximation Error in Mixed-Integer Optimal Control. *Mathematical Programming A*, 133(1–2), 1–23. doi:10.1007/s10107-010-0405-3.

Sager, S., Jung, M., and Kirches, C. (2011). Combinatorial Integral Approximation. *Mathematical Methods of Operations Research*, 73(3), 363–380. doi:10.1007/s00186-011-0355-4.

Sager, S. and Zeile, C. (2019). On mixed-integer optimal control with constrained total variation of the integer control. URL `http://www.optimization-online.org/DB_HTML/2019/10/7432.html`. Accessed Oct. 16, 2019.

Tsang, T.H., Himmelblau, D.M., and Edgar, T.F. (1975). Optimal control via collocation and non-linear programming. *International Journal of Control*, 21(5), 763–768. doi:10.1080/00207177508922030.

Wächter, A. and Biegler, L.T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1), 25–57. doi:10.1007/s10107-004-0559-y.