A column generation algorithm for solving energy system planning problems

Pavlo Muts¹ · Stefan Bruche² · Ivo Nowak¹ · Ouyang Wu¹ · Eligius M. T. Hendrix³ · George Tsatsaronis²

Received: 19 November 2020 / Revised: 26 June 2021 / Accepted: 27 August 2021 /

Published online: 3 October 2021

© The Author(s) 2021

Abstract

Energy system optimization models are typically large models which combine sub-models which range from linear to very nonlinear. Column generation (CG) is a classical tool to generate feasible solutions of sub-models, defining columns of global master problems, which are used to steer the search for a global solution. In this paper, we present a new inner approximation method for solving energy system MINLP models. The approach is based on combining CG and the Frank Wolfe algorithm for generating an inner approximation of a convex relaxation and a primal heuristic for computing solution candidates. The features of this approach are: (i) no global branch-and-bound tree is used, (ii) sub-problems can be solved in parallel to generate columns, which do not have to be optimal, nor become available at the same time to synchronize the solution, (iii) an arbitrary solver can be used to solve sub-models, (iv) the approach (and the implementation) is generic and can be used to solve other nonconvex MINLP models. We perform experiments with decentralized energy supply system models with more than 3000 variables. The numerical results show that the new decomposition method is able to compute high-quality solutions and has the potential to outperform state-of-the-art MINLP solvers.

Keywords Decomposition method · Parallel computing · Column generation · Nonconvex optimization · Global optimization · Mixed-integer nonlinear programming

✉ Ivo Nowak
ivo.nowak@haw-hamburg.de

Extended author information available on the last page of the article

1 Introduction

1.1 Energy system optimization

Energy systems involve the conversion, distribution, and storage of energy. They represent complex structures of interconnected subsystems, including different technologies, actors, and markets, depending on the problem under consideration and the defined system boundary (Möst et al. 2009). Energy system studies range from technology comparisons for domestic applications (Ashouri et al. 2013) to the design of long-term transnational transformation strategies (Brown et al. 2018). When mathematical optimization methods are applied to the analysis, it is the modeler's responsibility to select an appropriate level of detail for the investigated problem to ensure the computability of the model and the significance of the obtained results (Brooks and Tobias 1996). Hence, the modeler has to address the trade-off between model accuracy and the required solution effort. In the optimization-based process synthesis, Chen and Grossmann distinguish between aggregated models, shortcut models, and rigorous models (Chen and Grossmann 2017).

Aggregated models typically represent the underlying physical relationships in a simplified way. The problems under investigation are often embedded in large energy systems, and usually, the plant operation is considered in more detail. The resulting models are typically linear (LP) or mixed-integer linear (MILP), and commercial and non-commercial software is available to solve them effectively, e.g., (Gurobi Optimization 2020), (CPLEX IBM 2020), (CBC Forrest et al. 2020) or (SCIP Gamrath et al. 2020). Examples include design concepts for central plants to supply district heating networks (Bruche and Tsatsaronis 2019), industrial parks with electricity, heating, and cooling demands (Voll et al. 2013), and decentralized urban areas with consideration of heat network expansion (Rieder et al. 2014). Rigorous models, in contrast, feature a significantly higher level of detail of the modeled technical devices and their fundamental principles. They represent the plant behavior at the component level and take into account, for example, chemical equilibria or heat transfer phenomena. The models are typically developed using special simulation software, and for optimization, derivative-free black-box methods such as evolutionary algorithms are frequently employed. These often time-consuming techniques enable investigation of complex models but have the drawback that they do not provide quality proof of the results and may produce only local optima. Examples are the design optimization for supercritical steam power plants (Wang et al. 2014) and natural gas-fired oxy-fuel plants (Teichgraeber et al. 2017).

The shortcut models are a compromise between those mentioned above. They can either be large and thus closer to the aggregated models, but take into account, for instance, more advanced (nonlinear) functions for describing the component performance or costs, e.g., (Elsido et al. 2017; Goderbauer et al. 2016). Alternatively, the models can be more comparable to rigorous models with reduced complexity to enable deterministic solving methods to handle

them. Model simplification can be achieved by considering only a few characteristic load points to limit the number of variables, and introduce model equations with lower mathematical complexity, e.g., to describe the thermodynamic properties (Ahadi-Oskui et al. 2010; Jüdes et al. 2009). In both cases, the results are commonly nonlinear mathematical models (NLP). Most of them are also non-convex and have additional integer variables to state, for example, if a component is present or absent in a design. In this case, the problem class is mixed-integer nonlinear (MINLP). There are several deterministic methods available for solving nonconvex MINLPs. They can be subdivided into (i) branch-and-bound (Burer and Letchford 2012; Sahinidis 2020), (ii) MIP approximation Goderbauer et al. (2016), and (iii) decomposition methods (Nowak et al. 2019; Muts et al. 2020b). Most of the available commercial and open-source solvers, e.g., (BARON Sahinidis 2020) or (SCIP Gamrath et al. 2020) apply the branch-and-bound method. However, finding (high quality) solutions for large-scale MINLP problems with branch-and-bound-based solvers is often difficult with limited time and memory resources, as the size of the branch and bound tree to be managed may grow with the problem dimension.

Hence, a second approach in dealing with the challenging nonlinearities in optimization models is introducing piecewise-linear MIP approximations for the nonlinear functions. The MIP approximations are usually applied manually during the model preprocessing. The variety of applications includes combined cooling, heat, and power plants (Bischi et al. 2014), energy systems with heat demands at different temperature levels (Yokoyama et al. 2002), and electrochemical conversion technologies (Gabrielli et al. 2018). However, a MIP approximation solution is not an exact solution of the nonlinear model and might even be infeasible in the original problem. Goderbauer et al. addressed this challenge with a problem-tailored adaptive algorithm for the synthesis of decentralized energy supply systems (DESS) (Goderbauer et al. 2016). Their iterative two-step process combines the identification of approximate solutions with discretized variables and the solving of NLPs with fixed selected variables. Between two iterations, the discretization grid is reduced and shifted in the direction of the previously calculated solution. With this approach, Goderbauer et al. generated high-quality MINLP solutions in short computation time and outperformed the commercial solver BARON. However, we should take into account that unlike BARON, the method is not a general purpose MINLP solver, but it has been tuned to solve specifically DESS problems.

A third deterministic method for solving MINLPs is decomposition. The authors have gained very good experience with problem decomposition via Column Generation (CG) in solving huge network problems (Borndörfer et al. 2013; Nowak 2014). Motivated by CG's excellent performance, we present a new MINLP CG method. It is based on combining CG and the Frank-Wolfe method (FW) for generating an inner approximation (IA) of a convex relaxation, and a primal heuristic for computing solution candidates. The characteristics of this approach are:

- (i) it does not require a global branch-and-bound tree,
- (ii) sub-problems can be solved in parallel to generate columns, which do not have to be optimal, nor become available at the same time to synchronize the solution,
- (iii) an arbitrary solver can be used to solve sub-models.

The new method can be applied to general nonconvex or convex sparse real-world MINLPs, e.g. instances of the MINLPLib (Vigerske 2018). These models can be reformulated as block-separable problems, defined by low-dimensional sub-problems coupled by a moderate number of global linear constraints. In this work, we focus on the characteristics of the superstructure-based synthesis of decentralized energy supply system presented in Goderbauer et al. (2016).

1.2 Outline

First, Sect. 2 describes a practical application from the area of energy supply system modeling. Section 3 describes the decomposition view of the problem and sketches the resource-constrained approach. Section 4 presents various algorithms based on IA to solve the problem. The numerical evaluation in Sect. 5 shows the potential of the new decomposition-based approximation approach. We conclude in Sect. 6 outlining steps for future investigation.

2 Modeling of a practical application

2.1 Decentralized energy supply systems

A DESS is a complex, integrated system consisting of several energy conversion units and energy supply and demand forms. Usually, the starting point for DESS optimization is a superstructure containing all possible components and their interconnections. The optimization goal involves identifying a plant design and a suitable plant schedule simultaneously to minimize or maximize an objective function value. The objective function is usually of economic nature (e.g., maximizing the net present value of an investment). Binary variables (valid values one or zero) can be used to model the facility selection (existence/non-existence) and the operation states of selected units (on/off). Continuous non-negative variables represent the model component input and output streams and the component costs. Both input-output relationships and component costs can be expressed by nonlinear functions. In this case, the underlying model belongs to the class of mixed-integer-nonlinear-problems (MINLP).

Fig. 1 Superstructure of the investigated DESS with one unit of each component type

2.2 General MINLP model formulation

The DESS model used in this paper was initially presented in Voll et al. (2013) and is described in detail in Goderbauer et al. (2016). For the following brief model description, the notation is adopted from Goderbauer et al. (2016). The superstructure of the model is sketched in Fig. 1.

Every superstructure S is assembled from the components s , namely boilers (B), combined heat and power engines (C), electrically powered compression chillers (referred to as “turbo chillers”, T), and heat-utilizing absorption chillers (A). In the base case, each component can only be installed once in a superstructure (see Sect. 2.3 for more details). The DESS is required to meet the heating ($\dot{E}_\ell^{\text{heat}}$), cooling ($\dot{E}_\ell^{\text{cool}}$), and electricity demands ($\dot{E}_\ell^{\text{el}}$) for every load case ℓ . The latter are specified by a set of load cases L . Gas-fired boilers and combined heat and power engines are available to provide thermal energy (gas price: $p^{\text{gas, buy}}$). The electricity produced in the engines is used to satisfy the electricity demand. Missing or excess electricity can be balanced with the connected electricity grid. Costs of $p^{\text{el, buy}}$ are incurred for purchased electricity, and the electricity sale is reimbursed with $p^{\text{el, sell}}$. Electrically powered turbo-chillers and heat-utilizing absorption chillers are available to meet the cooling demand. The maximization of the net present value is selected as the objective function. The annual costs are discounted to the investment time using the interest rate i and the cash flow time γ^{CF} . The factor m_s considers the annual maintenance costs as a percentage of the investment costs I_s .

The following variable values are determined during the optimization process:

- existence/non-existence y_s of all units in the superstructure; $y_s \in \{0, 1\}$
- operation status $\delta_{s\ell}$ (on/off) of all units in all load cases; $\delta_{s\ell} \in \{0, 1\}$
- input $\dot{U}_{s\ell}$ and output energy flow $\dot{V}_{s\ell}$ of all units in all load cases (main output of engines: heat; electricity output $\dot{V}_{s\ell}^{\text{el}}$ is derived); $\dot{U}_{s\ell}, \dot{V}_{s\ell} \geq 0$
- nominal size $\dot{V}_s^N$ of all units (engines: related to heat output); investment costs I_s are derived from the nominal size; $\dot{V}_s^N \geq 0$

- electricity import $\dot{U}_\ell^{\text{el, buy}}$ and export $\dot{V}_\ell^{\text{el, sell}}$ from and to the grid in all load cases; $\dot{U}_\ell^{\text{el, buy}}, \dot{V}_\ell^{\text{el, sell}} \geq 0$

The objective function of the optimization problem is presented in Eq. (1). The net present value to be maximized is calculated from the discounted annual cash flows minus the overall investment costs. The annual cash flow is derived from the revenues for electricity sales minus the expenses for electricity import, purchased fuel, and maintenance costs.

$$\begin{aligned} \max \quad & \frac{(i+1)^{\gamma^{\text{CF}}} - 1}{i \cdot (i+1)^{\gamma^{\text{CF}}}} \cdot \left[\sum_{l \in L} \Delta_l \cdot \left(p^{\text{el, sell}} \cdot \dot{V}_l^{\text{el, sell}} - p^{\text{el, buy}} \cdot \dot{U}_l^{\text{el, buy}} \right. \right. \\ & \left. \left. - p^{\text{gas, buy}} \cdot \sum_{s \in B \cup C} \delta_{sl} \cdot \dot{U}_s(\dot{V}_{sl}, \dot{V}_s^{\text{N}}) \right) - \sum_{s \in S} m_s \cdot I_s(\dot{V}_s^{\text{N}}) \cdot y_s \right] \\ & - \sum_{s \in S} I_s(\dot{V}_s^{\text{N}}) \cdot y_s \end{aligned} \quad (1)$$

Equations (2)–(4) represent the balance constraints for the three energy demands heating, cooling, and electricity. They need to be met with equality in all load cases l .

$$\dot{E}_\ell^{\text{heat}} = \sum_{s \in B \cup C} \dot{V}_{s\ell} - \sum_{s \in A} \delta_{s\ell} \cdot \dot{U}_s(\dot{V}_{s\ell}, \dot{V}_s^{\text{N}}), \quad \forall \ell \in L \quad (2)$$

$$\dot{E}_\ell^{\text{cool}} = \sum_{s \in A \cup T} \dot{V}_{s\ell}, \quad \forall \ell \in L \quad (3)$$

$$\begin{aligned} \dot{E}_\ell^{\text{el}} = & \dot{U}_\ell^{\text{el, buy}} + \sum_{s \in C} \delta_{s\ell} \cdot \dot{V}_s^{\text{el}}(\dot{V}_{s\ell}, \dot{V}_s^{\text{N}}) \\ & - \sum_{s \in T} \delta_{s\ell} \cdot \dot{U}_s(\dot{V}_{s\ell}, \dot{V}_s^{\text{N}}) - \dot{V}_\ell^{\text{el, sell}}, \quad \forall \ell \in L \end{aligned} \quad (4)$$

All components are available in continuous nominal sizes. Equation (5) bounds their size to the range between a lower and upper limit.

$$\dot{V}_s^{\text{N, min}} \leq \dot{V}_s^{\text{N}} \leq \dot{V}_s^{\text{N, max}}, \quad \forall s \in S \quad (5)$$

Equations (6)–(8) are responsible to set the output energy flow to zero if the binary operation variable is zero. Otherwise the output flow is limited to values between the minimum part-load ($\alpha_s^{\text{min}} \cdot \dot{V}_s^{\text{N}}$) and the nominal size. Eq. (9) assures that every unit of a superstructure can only be operated if it exists.

$$\dot{V}_{s\ell} \leq \delta_{s\ell} \cdot \dot{V}_s^{\text{N, max}}, \quad \forall s \in S, \ell \in L \quad (6)$$

$$\dot{V}_{s\ell} \leq \dot{V}_s^{\text{N}}, \quad \forall s \in S, \ell \in L \quad (7)$$

$$\dot{V}_{s\ell} \geq \alpha_s^{\min} \cdot \dot{V}_s^N - (1 - \delta_{s\ell}) \cdot \alpha_s^{\min} \cdot \dot{V}_s^{N,\max}, \quad \forall \quad s \in S, \ell \in L \quad (8)$$

$$y_s \geq \delta_{s\ell}, \quad \forall \quad s \in S, \ell \in L \quad (9)$$

All parameter values, the nonlinear part-load performance curves $\dot{U}_s(\dot{V}_{s\ell}, \dot{V}_s^N)$, and investment cost functions $I_s(\dot{V}_s^N)$ are specified in the Appendix. As stated by Goderbauer et al. (2016) the MINLP model is nonconvex due to the investment cost functions and the nonlinear equality constraints (2) and (4).

2.3 DESSLib model instances

The DESS model studied here is published as library DESSLib (Bahl et al. 2016) including multiple model instances of varying complexity. The instances differ in three properties:

- number of similar units of all component types (S4, S8, S12, S16)
- number of load cases (L1, L2, L4, L6, L8, L12, L16, L24)
- energy demands values (ten sampled variants)

For example, instance S8L4-0 consists of maximal two units for each of the four component types (boiler, engine, turbo-chiller, absorption chiller). Both units can have different nominal sizes and can be operated independently. Furthermore, instance S8L4-0 includes four load cases (L4), and uses the data variation with the index zero. The DESSLib is used in Goderbauer et al. (2016) to evaluate their problem-tailored method AdaptDiscAlgo (see explanation in Sect. 1) and to benchmark it with state-of-the-art solvers. The results are also accessible for download at Bahl et al. (2016). Thus, high-quality, near-optimal solutions for the applied MINLP problems are available for all model instances.

2.4 MINLP block model formulation

Our model has been developed with the Python-based algebraic modeling language Pyomo. Pyomo provides the component `Block`, which serves as a container for variables, constraints, and more (Hart et al. 2017). For all energy conversion units of the DESS model, i.e., for each boiler, engine, turbo-chiller, and absorption chiller, individual Pyomo blocks are created. These main blocks contain the component variables for existence, nominal size, and investment costs, and the associated nonlinear investment cost functions. Additionally, one or multiple sub-blocks are created within the main component block to store the variables and constraints with load case dependency. These include operation status variables, input and output flow variables, plus the nonlinear performance constraints, and Eq. (6)–(9). Each sub-block can contain one or more load cases. Thus, the size of the sub-blocks is adjustable and depends on the decision of

Fig. 2 Implemented modeling concept with the application of Pyomo Block components. In this example, each load case ℓ has an individual sub-block in the unit s

the model user. A schematic representation of the modeling concept is shown in Fig. 2. Modeling with Pyomo blocks is necessary to support the automatic model decomposition. Further details are provided in Sect. 3.1.

3 MINLP formulation and approximation

3.1 Block-separable formulation

In order to apply the new decomposition method, the DESS model is formulated as a *block-separable* (or *quasi-separable*) MINLP problem, where the vector of variables $x \in \mathbb{R}^n$ is partitioned into $|K|$ blocks of dimension n_k such that $n = \sum_{k \in K} n_k$ and $x_k \in \mathbb{R}^{n_k}$ denotes the variables of block k . The model is described by

$$\min c^T x \quad \text{s.t.} \quad x \in P, \quad x_k \in X_k, \quad k \in K \quad (10)$$

with global (coupling) linear constraints

$$P := \{x \in \mathbb{R}^n : a_i^T x \leq b_i, i \in M_1, a_i^T x = b_i, i \in M_2\}, \quad (11)$$

and local (block) constraints

$$X_k := G_k \cap Y_k, \quad Y_k := \mathbb{R}^{n_{k1}} \times \mathbb{Z}^{n_{k2}}, \quad (12)$$

with $n_k = n_{k1} + n_{k2}$ and

$$G_k := \{y \in [x_k, \bar{x}_k] \subset \mathbb{R}^{n_k} : g_{kj}(y) \leq 0, j \in J_k\}, \quad (13)$$

Fig. 3 Example of the nonlinear investment cost function of the absorption chillers (see Eq. (A.7) in the Appendix)

and $|M_1| + |M_2| = m$. The vectors $\underline{x}, \bar{x} \in \mathbb{R}^n$ denote lower and upper bounds on the variables.

The local (block) set X_k is defined by linear and nonlinear local constraint functions, $g_{kj} : \mathbb{R}^{n_k} \rightarrow \mathbb{R}$, which are assumed to be bounded and continuously differentiable within the set $[\underline{x}_k, \bar{x}_k]$. An example of a nonlinear constraint function is depicted in Fig. 3. The linear global constraints in P are defined by $a_i^T x := \sum_{k \in K} a_{ki}^T x_k$, $a_{ki} \in \mathbb{R}^{n_k}$, $b_i \in \mathbb{R}$, $i \in [m]$. The linear objective function is defined by $c^T x := \sum_{k \in K} c_k^T x_k$, $c_k \in \mathbb{R}^{n_k}$.

Blocks in set K can be detected in an automatic way or can be defined manually. One idea of automatic block detection is based on connected components of the so-called ‘sparsity graph’, defined by nonzero entries of the Hessian of constraint functions of the original MINLP (Nowak 2005). In this paper, we consider the MINLP problems (DESS models) with manually defined blocks, see Sect. 2.4 for implementation details. Some of these blocks are defined by continuous variables and by linear local constraints. We merge these blocks into one single linear block which is represented by the following polytope

$$X_1 := \{y \in [\underline{x}_1, \bar{x}_1] \subset \mathbb{R}^{n_1} : g_{1j}(y) := a_{1j}^T y - b_{1j} \leq 0, j \in J_1\}. \quad (14)$$

Further details how we deal with linear block (14) are given in Sect. 4.1.1.

3.2 Resource-constrained reformulation

In order to define an inner approximation, the MINLP (10) is reformulated as a *resource-constrained optimization problem* by an affine transformation of the original variables x_k , $k \in K$, into resource variables (Muts et al. 2020a)

$$w_k := A_k x_k, \quad A_{ki} = \begin{cases} c_k^T & : i = 0, \\ a_{ki}^T & : i \in [m]. \end{cases} \quad (15)$$

The variables $w_k, k \in K$, describe how the objective value and the constraint values $a_i^T x$ are distributed over blocks. Considering the transformed local feasible set

$$W_k := \{A_k x_k : x_k \in X_k\} \subset \mathbb{R}^{m+1}, \quad (16)$$

the *resource-constrained formulation* of (10) is defined as

$$\min F(w) \quad \text{s.t.} \quad w \in H, \quad w_k \in W_k, \quad k \in K, \quad (17)$$

with the transformed objective function

$$F(w) := \sum_{k \in K} w_{k0},$$

and the transformed global feasible set

$$H := \left\{ w \in \mathbb{R}^{(m+1)|K|} : \sum_{k \in K} w_{ki} \leq b_i, \quad i \in M_1, \quad \sum_{k \in K} w_{ki} = b_i, \quad i \in M_2 \right\}. \quad (18)$$

Notice that the resource constraint view considers a space with $(m+1)|K|$ variables instead of the original n variables. For sparse MINLPs with sparse matrices A_k the number of resources variables can be greatly reduced, since only a resource variable w_{ki} is needed, if $A_{ki} \neq 0$. For instance, let $m = 1499, |K| = 50$ and number of nonzero rows of $A_k, k \in K$ to be 100. Then the overall number of resource variables is 75000. This number is reduced to 5000, if we consider only the nonzero rows of matrix A_k .

3.3 Convex hull relaxation and inner approximation

A resource-constrained *convex hull relaxation* of (17) is defined by

$$\min F(w) \quad \text{s.t.} \quad w \in H, \quad w_k \in \text{conv}(W_k), \quad k \in K. \quad (19)$$

The quality of relaxation (19) depends strongly on the *duality gap*, defined by

$$\text{gap} := \text{val}(10) - \text{val}(19). \quad (20)$$

Let $S_k := \{y_{kj}\}_{j \in [S_k]} \subset X_k$ be a finite set of locally feasible points, called *inner approximation* of X_k , and define the corresponding resources by $R_k := \{A_k y : y \in S_k\}$, called *columns*. An inner approximation of (19), called LP-IA, is defined by

$$\min F(w) \quad \text{s.t.} \quad w \in H, \quad w_k \in \text{conv}(R_k), \quad k \in K. \quad (21)$$

We use the notation $[R] := \{1, \dots, |R|\}$ for the index set of the finite set R , i.e. $R = \{r_j : j \in [R]\}$. An LP formulation of (21) is given by

$$\min F(w(z)) \quad \text{s.t.} \quad w(z) \in H, \quad z_k \in \Delta_{|R_k|}, \quad k \in K, \quad (22)$$

where

$$w(z) := w_k(z_k)_{k \in K}, \quad w_k(z_k) := \sum_{j \in [R_k]} z_{kj} r_{kj}, \quad z_k \in \Delta_{|R_k|}. \quad (23)$$

We call z_{kj} the *weight* of column $r_{kj} \in R, k \in K$ and denote by $\Delta_{|R_k|} \subset \mathbb{R}^{|R_k|}$ the standard simplex with $|R_k|$ vertices in the following way

$$\Delta_{|R_k|} = \{z \in \mathbb{R}^{|R_k|} : \sum_{j \in [R_k]} z_j = 1, z_j \geq 0, j \in [R_k]\}. \quad (24)$$

4 A MINLP column generation algorithm

In this section we describe a new MINLP CG method. The method, called MINLPCG, solves MINLP sub-problems to generate partial feasible solutions $x_k \in X_k$ and $w_k \in W_k$, which in the end are combined into a global solution which also fulfills the global constraints. The direction for looking for better partial solutions are given by master problems in the resource space. The basic steps of MINLPCG are:

1. Initialize column sets $R_k, k \in K$.
2. Solve (21) for computing an approximate solution $\tilde{w}$ of (19).
3. Project $\tilde{w}$ onto the feasible set finding a solution candidate w^* .
4. Add new columns to $R_k, k \in K$.
5. Repeat from step 2 until stopping criterion.

Since the sub-models of (10) represent difficult MINLP problems, it is not easy to quickly generate many columns. Therefore, we developed several strategies for accelerating the column generation (CG).

4.1 Traditional generation of columns

4.1.1 Dealing with linear block (sub-problem), adapting inner approximation

In the DESS models the dimension n_1 of the linear block (14) is much higher than the dimension n_k of other blocks. Preliminary numerical experiments showed that, if we distinguish the linear block instead of running traditional CG over all blocks, we obtain a running time reduction in order of magnitude from 2 days towards 1-2 hours. Mathematically this can be explained by the many vertices that the polytope of a linear block has which should be generated as columns if treated as nonlinear blocks. Therefore, we

use the following modified LP-IA master problem (25), for which the linear constraints of X_1 in (14) are directly integrated in the LP-IA:

$$\begin{aligned} & \min F(w(z, x_1)) \\ & \text{s.t.} \quad \sum_{k \in K} w_{ki}(z_k, x_1) \leq b_i, \quad i \in M_1, \\ & \quad \sum_{k \in K} w_{ki}(z_k, x_1) = b_i, \quad i \in M_2, \\ & \quad z_k \in \Delta_{|R_k|}, \quad k \in K \setminus \{1\}, \quad x_1 \in X_1, \end{aligned} \quad (25)$$

where $w_k(z_k, x_1)$ is defined similarly as in (23) by

$$w_k(z_k, x_1) := \begin{cases} w_k(z_k) & : k \in K \setminus \{1\} \\ A_1 x_1 & : k = 1. \end{cases} \quad (26)$$

So we distinguish $K \setminus \{1\}$ as the set of nonlinear blocks whereas $x_1 \in \mathbb{R}^{n_1}$ are the variables of the linear block, and X_1 is defined by only local linear constraints as in (14).

Lemma 1 *Let $R_1 = \text{vert}(W_1)$ be the set of extreme points of W_1 . Then problems (25) and (21) are equivalent.*

Proof By definition (14), X_1 defines a polytope. Hence, W_1 is a polytope defined by a linear transformation of X_1 , and $W_1 = \text{conv}(R_1)$. Therefore, $A_1 x_1$ in (26) can be replaced by $w_1(z_1)$, which proves the statement. $\square$

The procedure `SOLVEINNERLP(R)` solves (25). It returns a primal solution $(x_k)_{k \in K}$, where x_1 is the solution of the linear block and $x_k = x_k(z_k)$ with

$$x_k(z_k) := \sum_{j \in [S_k]} z_{kj} y_{kj}, \quad z_k \in \Delta_{|S_k|}, \quad k \in K \setminus \{1\}, \quad (27)$$

and S_k is the set of generated feasible points of X_k related to R_k , i.e. $r_{kj} = A_k y_{kj}$. Moreover, the procedure returns the dual values μ for the global resource constraints.

4.1.2 Generation of columns solving sub-problems

The MINLP CG-algorithm generates columns by solving the following MINLP sub-problems

$$y_k \in \argmin d^T A_k x_k, \quad x_k \in X_k \quad (28)$$

regarding a search direction $d \in \mathbb{R}^{m+1}$, where d is typically defined by a dual solution $\mu \in \mathbb{R}^m$ of the LP-IA (21), i.e. $d = (1, \mu^T)$. Notice that the result y_k corresponds

to an extreme point of X_k as well as W_k and is a so-called supported Pareto point in the resource space, see Muts et al. (2020b).

The procedure `SOLVEMINLP SUBPROBL( $d$ )` solves (28) and is used in procedure `ADD COL( $d, R_k$ )`, described in Algorithm 1, to add a column $A_k y_k$ to R_k . Moreover, the procedure computes the *reduced cost*

$$\delta_k := \max\{0, \min\{d^T r_k - d^T A_k y_k : r_k \in R_k\}\}, \quad (29)$$

which is used later to measure the impact of the procedure. If $\delta_k < 0$ for some $k \in K$, then column $A_k y_k$ may improve the objective value of (25). In the other case, if $\delta = 0$, the objective value of (25) cannot be changed (Nowak 2005) and the column generation algorithm can be stopped.

Algorithm 1 Generation of columns

```

1: function ADDCOL( $d, R_k$ )
2:    $y_k \leftarrow \text{SOLVEMINLP SUBPROBL}(d)$ 
3:    $\delta_k \leftarrow \max\{0, \min\{d^T r_k - d^T A_k y_k : r_k \in R_k\}\}$ 
4:   if  $A_k y_k \notin R_k$  then
5:      $R_k \leftarrow R_k \cup \{A_k y_k\}$  # add column
6:   return  $(\delta_k, R_k)$ 

```

Algorithm 2 shows a method for generating columns by alternately solving an LP-IA master problem and generating columns using Algorithm 1. The set $\hat{K} \subseteq K$ is subset of the block set K . Note that columns can be generated in parallel.

Algorithm 2 Column generation

```

1: function COLGEN( $\hat{K}, R$ )
2:   repeat
3:      $(x, \mu) \leftarrow \text{SOLVEINNERLP}(R)$ 
4:     for  $k \in \hat{K}$  do # col generation
5:        $(\delta_k, R_k) \leftarrow \text{ADD COL}((1, \mu^T), R_k)$ 
6:   until  $\delta = 0$  or stopping criterion
7:   return  $R$ 

```

4.1.3 Initializing columns

In order to initialize column set R_k , we perform a *subgradient method* using Algorithm 3 for maximizing the dual of (10) regarding the global constraints:

$$L(\mu) := \sum_{k \in K} \left(\min_{y_k \in X_k} (1, \mu^T) A_k y_k \right) - \mu^T b. \quad (30)$$

We compute the step length α^p by comparing the values of the function $L(\mu)$ defined in (30) at different iterations p of Algorithm 3, similarly as in Shor (1985):

$$\alpha^{p+1} = \begin{cases} 0.5\alpha^p & : L(\mu^p) \leq L(\mu^{p-1}), \\ 2\alpha^p & : \text{otherwise.} \end{cases} \quad (31)$$

Note that procedure ADDCOL in Algorithm 3 can be performed in parallel.

Algorithm 3 IA initialization

```

1: function IAINIT
2:    $R \leftarrow \emptyset, \quad p \leftarrow 0, \quad \mu^p \leftarrow 0, \quad \alpha^p = 1$ 
3:   repeat # subgradient steps
4:      $\hat{x}_1 \leftarrow \operatorname{argmin}_{x_1 \in X_1} (1, \mu^p)^T x_1, \quad \hat{r}_1 \leftarrow A_1 \hat{x}_1$ 
5:     for  $k \in K \setminus \{1\}$  do
6:        $(\delta_k, R_k) \leftarrow \text{ADDCOL}((1, \mu^p), R_k)$ 
7:        $\hat{r}_k \leftarrow r_{k, |R_k|}$  # new column
8:        $p \leftarrow p + 1, \mu_i^p \leftarrow \mu_i^{p-1} + \alpha^p (\sum_{k \in K} \hat{r}_{ki} - b_i), i = 1, \dots, m$ 
9:   until  $p = p_{\max}$ 
10:  return  $R$ 

```

4.2 Accelerating column generation

In order to accelerate CG without calling an MINLP routine, we developed two methods. The first method generates columns by performing NLP local search from good starting points of an IA-LP. The second method is a Frank-Wolfe algorithm for quickly solving the convex hull relaxation (19).

4.2.1 Fast generation of columns using NLP local search and rounding

The master problem used to provide starting points for the column generation is LP-IA problem (21). Since in the beginning only few columns are available, often LP-IA master problem (21) is infeasible, i.e. $H \cap \prod_{k \in K} \operatorname{conv}(R_k) = \emptyset$. Therefore, we use an LP-IA master problem (32), that includes slacks and a penalty term $\Psi(\theta, s)$, similar as in du Merle et al. (1999):

$$\begin{aligned}
 & \min F(w(z, x_1)) + \Psi(\theta, s) \\
 \text{s.t.} \quad & \sum_{k \in K} w_{ki}(z_k, x_1) \leq b_i + s_i, \quad i \in M_1, \\
 & \sum_{k \in K} w_{ki}(z_k, x_1) = b_i + s_{i1} - s_{i2}, \quad i \in M_2, \\
 & z_k \in \Delta_{|R_k|}, \quad k \in K \setminus \{1\}, \quad x_1 \in X_1, \\
 & s_i \geq 0, \quad i \in [m],
 \end{aligned} \quad (32)$$

where the penalty term for slack variables is

$$\Psi(\theta, s) := \sum_{i \in M_1} \theta_i s_i + \sum_{i \in M_2} \theta_i (s_{i1} + s_{i2}), \quad (33)$$

and penalty weights $\theta > 0$ are sufficiently large. We define $w_k(z_k, x_1)$ similarly to (23) as follows

$$w_k(z_k, x_1) := \begin{cases} w_k(z_k) & : k \in K \setminus \{1\}, \\ A_1 x_1 & : k = 1, \end{cases}$$

and $x_1 \in \mathbb{R}^{n_1}$ are the variables of the linear block, defined in (14). The procedure `SOLVESLACKINNERLP(R)` solves (32) and returns solution point x , dual solution μ and slack values s . If the slack variables are nonzero, i.e. $s \neq 0$, in order to eliminate nonzero slack variables, the method `SLACKDIRECTIONS` computes a new search direction $d \in \mathbb{R}^m$

$$d := \sum_{\substack{s_i > 0.1 \max(s), \\ i \in M_1}} e_i + \sum_{\substack{s_{i1} > 0.1 \max(s), \\ i \in M_2}} e_i - \sum_{\substack{s_{i2} > 0.1 \max(s), \\ i \in M_2}} e_i,$$

with $e_i \in \mathbb{R}^m$ the coordinate i unit vector.

Algorithm 4 Approximate sub-problem solving

```

1: function APPROXSOLVEMINLPSUBPROBL( $x_k, d$ )
2:    $\tilde{y}_k \leftarrow \text{SOLVENLPSUBPROBLEM}(x_k, d)$ 
3:    $\hat{x}_k \leftarrow \text{ROUND}(\tilde{y}_k)$ 
4:    $\tilde{x}_k \leftarrow \text{SOLVEFIXEDNLPSUBPROBLEM}(\hat{x}_k)$ 
5:   return  $\tilde{x}_k$ 

```

Since, for the CG algorithm, it is sufficient to compute high-quality local feasible solutions, we present a local search procedure `APPROXSOLVEMINLPSUBPROBL` in Algorithm 4 based on rounding of locally feasible point. The goal of this procedure is to avoid usage of a MINLP solver for solving sub-problems and, therefore, reduce the time for sub-problem solving. The inputs of local search procedure `APPROXSOLVEMINLPSUBPROBL` are the block solution x_k as starting point and the direction d or $(1, \mu)$ as search direction. It starts by running procedure `SOLVENLPSUBPROBLEM` which computes a local minimizer of the integer relaxed sub-problem

$$\begin{aligned} \tilde{y}_k &:= \operatorname{argmin} d^T A_k x \\ \text{s.t. } &x \in G_k \end{aligned} \quad (34)$$

starting from the primal solution x_k of the LP-IA. Then the procedure `ROUND` rounds integer variables of block k in $\tilde{y}_k$ to obtain $\hat{x}_k$. Finally, procedure `SOLVEFIXEDNLPSUBPROBLEM` solves again an NLP problem, fixing the rounded integer variables of $\hat{x}_k$:

$$\tilde{x}_k := \operatorname{argmin} c_k^T x_k \quad \text{s.t.} \quad x_k \in G_k, \quad x_{ki} = \hat{x}_{ki}, \quad i \in [n_{k2}], \quad (35)$$

and using the continuous variable values of $\tilde{x}_k$ as starting point. The complete column generation procedure is depicted in Algorithm 5. Note that procedure APPROX-SOLVEMINLP SUBPROBL in Algorithm 5 can be performed in parallel.

Algorithm 5 Approximate column generation

```

1: function APPROXCOLGEN( $R$ )
2:   repeat
3:      $(x, \mu, s) \leftarrow \text{SOLVESLACKINNERLP}(R)$ 
4:     for  $k \in K \setminus \{1\}$  do                                     # col generation
5:        $y_k \leftarrow \text{APPROXSOLVEMINLP SUBPROBL}(x_k, (1, \mu^T))$ 
6:       if  $A_k y_k \notin R_k$  and  $y_k \in X_k$  then    $R_k \leftarrow R_k \cup \{A_k y_k\}$    # add column
7:        $(x, \mu, s) \leftarrow \text{SOLVESLACKINNERLP}(R)$ 
8:       if  $\|s\|_\infty > 0$  then                                     # reduce slacks
9:          $d \leftarrow \text{SLACKDIRECTIONS}(s)$ 
10:        for  $k \in K \setminus \{1\}$  do                               # col generation
11:           $y_k \leftarrow \text{APPROXSOLVEMINLP SUBPROBL}(x_k, d)$ 
12:          if  $A_k y_k \notin R_k$  and  $y_k \in X_k$  then    $R_k \leftarrow R_k \cup \{A_k y_k\}$    # add column
13:      until stopping criterion
14:  return  $R$ 

```

4.2.2 CG by solving the convex hull relaxation using a Frank-Wolfe algorithm

In this section, we present a Frank-Wolfe algorithm which is an alternative way to generate columns. It is based on solving convex hull relaxation (19) by a quadratic penalty function approach:

$$Q(w, \sigma) := F(w) + \sum_{i \in [m]} \sigma_i \left(\sum_{k \in K} w_{ki} - b_i \right)^2 \quad (36)$$

where $\sigma \in \mathbb{R}_+^m$ is a vector of penalty weights. Consider the convex optimization problem

$$\min Q(w, \sigma) \quad \text{s.t.} \quad w_k \in \text{conv}(W_k), \quad k \in K. \quad (37)$$

Let μ^* be an optimal dual solution of (19) regarding the global constraints $w \in H$ and set the penalty weights $\sigma_i = 0$, if $\mu_i^* = 0$, and $\sigma_i \geq |\mu_i^*|$, else, for $i \in [m]$. Then it can be shown that (36) is an exact penalty function and (37) is a reformulation of the convex relaxation (19), i.e. (19) is equivalent to (37).

Algorithm 6 presents a Frank-Wolfe (FW) algorithm for approximately solving the convex penalty problem (37). For acceleration, we use the Nesterov direction update rule (Nesterov 1983), line 17. We set the penalty weight $\sigma = |\mu|$, where μ is a dual solution of LP-IA (32). One step of the FW-algorithm is performed by approximately solving the problem with a linearized objective

$$\min \nabla_w Q(\tilde{w}, \sigma)^T w \quad \text{s.t.} \quad w_k \in \text{conv}(W_k), \quad k \in K, \quad (38)$$

which is equivalent to solving the sub-problems

$$\min \nabla_{w_k} Q(\tilde{w}, \sigma)^T w_k \quad \text{s.t.} \quad w_k \in \text{conv}(W_k). \quad (39)$$

The sub-problem (39) is solved with APPROXSOLVEMINLP SUBPROBL, depicted in Algorithm 4, in order to compute quickly new columns. The columns can be computed in parallel. Note that the gradient $\nabla_{w_k} Q(\tilde{w}, \sigma)$ is defined by

$$\frac{\partial}{\partial w_{k0}} Q(w, \sigma) = 1, \quad \frac{\partial}{\partial w_{ki}} Q(w, \sigma) = 2\sigma_i \left(\sum_{\ell \in K} w_{\ell i} - b_i \right) =: \eta_i(w, \sigma).$$

Hence, $\nabla_{w_k} Q(\tilde{w}, \sigma) = (1, \eta(\tilde{w}, \sigma)^T)$ for all $k \in K$. The quadratic line search problem

$$\theta = \text{argmin} Q(\tilde{w} + t(r - \tilde{w}), \sigma)$$

in step 14 of Algorithm 6 can be easily solved.

Algorithm 6 Fast column generation using a Frank-Wolfe method

```

1: function FWCOLGEN( $R$ )
2:   repeat
3:      $(x, \mu) \leftarrow \text{SOLVEINNERLP}(R)$ 
4:     for  $i \in [m]$  do  $\sigma_i \leftarrow |\mu_i|$ 
5:     for  $k \in K \setminus \{1\}$  do  $\tilde{w}_k \leftarrow \text{argmin}_{r_k \in R_k} (1, \mu^T) r_k$ 
6:      $\tilde{w}_1 \leftarrow A_1 x_1$ 
7:      $\gamma^+ \leftarrow 1, v^+ \leftarrow \tilde{w}, \quad d \leftarrow (1, \mu^T)$ 
8:     repeat # Frank Wolfe iteration
9:        $\tilde{x}_1 \leftarrow \text{argmin}_{x_1 \in X_1} d^T x_1, \quad \tilde{r}_1 \leftarrow A_1 \tilde{x}_1$ 
10:      for  $k \in K \setminus \{1\}$  do
11:         $\tilde{x}_k \leftarrow \text{APPROXSOLVEMINLP SUBPROBL}(x_k, d), \quad \tilde{r}_k \leftarrow A_k \tilde{x}_k$ 
12:        if  $\tilde{x}_k \in X_k$  (feasible) and  $\tilde{r}_k \notin R_k$  then  $R_k \leftarrow R_k \cup \{\tilde{r}_k\}$ 
13:       $(x, \mu) \leftarrow \text{SOLVEINNERLP}(R)$ 
14:       $\theta \leftarrow \text{argmin}_t Q(\tilde{w} + t(\tilde{r} - \tilde{w}), \sigma)$ 
15:       $v \leftarrow v^+, \quad v^+ \leftarrow \tilde{w} + \theta(\tilde{r} - \tilde{w})$ 
16:       $\gamma \leftarrow \gamma^+, \quad \gamma^+ \leftarrow 0.5(1 + \sqrt{4\gamma^2 + 1})$ 
17:       $\tilde{w} \leftarrow v^+ + \frac{\gamma-1}{\gamma^+}(v^+ - v)$ 
18:       $d \leftarrow (1, \eta(\tilde{w}, \sigma)^T)$ 
19:    until stopping criterion
20:  until stopping criterion
21:  return  $R$ 
```

4.3 A primal heuristic for finding solution candidates

In this section, we present two heuristic procedures for computing solution candidates. The first one computes a feasible solution from the slack LP-IA problem (32) solution. The second one computes high-quality solution candidates.

Algorithm 7 presents the initial primal heuristic, which aims at eliminating slacks in LP-IA master problem (32). It starts with running procedure

NLPRESOURCEPROJECT, which performs an NLP local search solution of the following integer relaxed *resource-projection* NLP master problem

$$\begin{aligned} \min \quad & \sum_{k \in K} \|A_k x_k - A_k \tilde{x}_k\|^2, \\ \text{s.t.} \quad & x \in P, \quad x_k \in G_k, \quad k \in K, \end{aligned} \quad (40)$$

where $\tilde{x}$ is the solution of the LP-IA master problem (32).

Algorithm 7 Initial primal heuristic

```

1: function FINDSOLUTIONINIT( $X^*, R$ )
2:    $(\tilde{x}, \mu, s) \leftarrow \text{SOLVESLACKINNERLP}(R)$ 
3:    $\tilde{y} \leftarrow \text{NLPRESOURCEPROJECT}(\tilde{x})$ 
4:    $\hat{x} \leftarrow \text{MIPPROJECT}(\tilde{y})$ 
5:    $\tilde{x} \leftarrow \text{SOLVEFIXEDNLP}(\hat{x})$ 
6:   if  $\tilde{x}$  feasible then
7:      $X^* \leftarrow X^* \cup \{\tilde{x}\}$ 
8:     for  $k \in K \setminus \{1\}$  do  $R_k \leftarrow R_k \cup \{A_k \tilde{x}_k\}$ 
9:   return  $(X^*, R)$ 
```

Using the potentially fractional solution $\tilde{y}$ of (40), the algorithm computes an integer globally feasible solution $\hat{y}$ by calling the procedure $\text{MIPPROJECT}(\tilde{y})$ which solves *MIP-projection* master problem

$$\begin{aligned} \hat{y} = \operatorname{argmin} \quad & \sum_{k \in K} \|x_k - \tilde{y}_k\|_\infty \\ \text{s.t.} \quad & x \in P, \quad x_k \in Y_k, \quad k \in K. \end{aligned} \quad (41)$$

The integer globally feasible solution $\hat{y}$ is then used to perform an NLP local search, where integer variables are fixed starting from $\hat{y}$ by procedure $\text{SOLVEFIXEDNIP}(\hat{y})$:

$$\begin{aligned} x^* = \operatorname{argmin} \quad & c^T x + \sum_{i \in M_1} \theta_i s_i + \sum_{i \in M_2} \theta_i (s_{i1} + s_{i2}) \\ \text{s.t.} \quad & \sum_{k \in K} A_k x_k \leq b_i + s_i, \quad i \in M_1, \\ & \sum_{k \in K} A_k x_k = b_i + s_{i1} - s_{i2}, \quad i \in M_2, \\ & x_k \in G_k, x_{ki} = \hat{y}_{ki}, \quad i \in [n_{k2}], k \in K. \end{aligned} \quad (42)$$

Algorithm 8 presents a primal heuristic for computing a high-quality solution candidate of MINLP problem (10). The procedure is very similar to Algorithm 7, but it does not use the NLP resource-projection problem (40). Instead, the solution of LP-IA (25) is used directly in MIP-projection problem (41). There is no guarantee that the optimal solution of (41) provides the best primal bound. Moreover, it may be infeasible for the original problem (10). Therefore, we generate a pool $\hat{Y}$ of feasible solutions of problem (41) provided by the MIP solver. Solution pool $\hat{Y}$ provides

good starting points for an NLP local search over the global space and increases the possibility of improving the quality of solution candidate.

Similarly to Algorithm 7, Algorithm 8 starts with computing an inner LP solution $\tilde{x}$ of problem (25) by calling procedure SOLVEINNERLP. Point $\tilde{x}$ is used in a procedure SOLPOOLMIPPROJECT($\tilde{x}, N$) to generate solution set (pool) $\hat{Y}$ of (41) of size N , which also includes the optimal solution. Like in Algorithm 7, those alternative solutions are used to perform an NLP local search over the global space, defined in (42) by fixing the integer valued variables. In order to find better solution candidates, these steps are repeated iteratively with an updated point $\tilde{x}$. In each iteration, the point $\tilde{x}$ is shifted towards the point x^* which corresponds to the best current primal bound of the original problem (10). This is a typical heuristic local search procedure, which aims to generate a different solution pool $\hat{Y}$ in each iteration of the algorithm. Algorithm 8 terminates when the maximum number of iterations is reached or the best primal bound in the current iteration does not improve the best primal bound in the previous iteration.

Algorithm 8 Primal heuristic

```

1: function FINDSOLUTION( $X^*, R$ )
2:    $(\tilde{x}, \mu) \leftarrow \text{SOLVEINNERLP}(R)$ 
3:   repeat
4:      $\hat{Y} \leftarrow \text{SOLPOOLMIPPROJECT}(\tilde{x}, N)$ 
5:     for  $\hat{y} \in \hat{Y}$  do
6:        $\tilde{x} \leftarrow \text{SOLVEFIXEDNLP}(\hat{y})$ 
7:       if  $\tilde{x}$  feasible then
8:          $X^* \leftarrow X^* \cup \{\tilde{x}\}$ 
9:         for  $k \in K \setminus \{1\}$  do  $R_k \leftarrow R_k \cup \{A_k \tilde{x}_k\}$ 
10:       $x^* \leftarrow \operatorname{argmin}_{x \in X^*} c^T x$ 
11:       $\tilde{x} \leftarrow \tau \tilde{x} + (1 - \tau)x^*$  #  $\tau \in (0, 1)$ 
12:    until stopping criterion
13:  return ( $X^*, R$ )

```

4.4 Main algorithm

Algorithm 9 describes a MINLP-CG method for computing a solution candidate of (10). The algorithm starts with the initialization of IA with the procedure IAINIT (Algorithm 3). Since the problem (32) might have nonzero slack values, the algorithm tries to eliminate them by computing a first primal solution. This is done by alternately calling the procedures APPROXCOLGEN (Algorithm 5) and FINDSOLUTIONINIT (Algorithm 7). For quickly improving the convex relaxation (19), the algorithm calls FW-based column generation procedure FWCOLGEN (Algorithm 6).

In the main loop, the algorithm alternately performs COLGEN (Algorithm 2) and heuristic procedure FINDSOLUTION (Algorithm 8) for computing solution candidates. The procedure COLGEN is performed for a subset of blocks $\hat{K} \subseteq K \setminus \{1\}$, in order to keep the number of solved MINLP sub-problems low. Moreover, focusing on a subset of blocks helps avoiding computing already existing columns. The blocks can

be excluded for a while by looking at the value of the reduced cost $\delta_k, k \in K \setminus \{1\}$, which is computed in line 14, as defined in (29). The reduced block set $\hat{K}$ contains the blocks where the reduced cost is negative, i.e. $\delta_k < 0, k \in K \setminus \{1\}$, and is updated at each main iteration by solving the sub-problems for the full set K . Note that if the reduced cost δ_k is nonnegative for all blocks, i.e. $\delta_k \geq 0, k \in K \setminus \{1\}$, the Column Generation has converged and the algorithm terminates.

Algorithm 9 MINLP CG Algorithm

```

1: function MINLPCG
2:    $R \leftarrow \text{IAINIT}, \quad X^* \leftarrow \emptyset$                                 # init columns
3:   repeat                                                                # slack elimination
4:      $R \leftarrow \text{APPROXCOLGEN}(R)$ 
5:      $(X^*, R) \leftarrow \text{FINDSOLUTIONINIT}(X^*, R)$ 
6:   until  $X^* \neq \emptyset$ 
7:    $R \leftarrow \text{FWCOLGEN}(R)$                                            # fast sol. of conv. hull relaxation
8:    $\delta \leftarrow -\infty$ 
9:   repeat                                                                # main iteration
10:     $\hat{K} \leftarrow \{k \in K \setminus \{1\} : \delta_k < 0\}$                       # sub-problem reduction
11:     $R \leftarrow \text{COLGEN}(\hat{K}, R)$                                        # col. gen. for reduced block set
12:     $(x, \mu) \leftarrow \text{SOLVEINNERLP}(R)$ 
13:    for  $k \in K \setminus \{1\}$  do                                           # col. gen. for all blocks
14:       $(\delta_k, R_k) \leftarrow \text{ADDCOL}((1, \mu^T), R_k)$ 
15:       $(X^*, R) \leftarrow \text{FINDSOLUTION}(X^*, R)$                          # primal heuristic
16:    until  $\delta \geq 0$  or  $\text{TIMELIMIT}$ 
17:  return  $\min_{x \in X^*} c^T x$ 

```

4.5 Convergence analysis

Column Generation and Frank-Wolfe algorithms are well-known approaches and their convergence has already been proven. In this section, we discuss the convergence of Column Generation in Algorithm 9 and the Frank-Wolfe method in Algorithm 6.

4.5.1 Convergence of Algorithm 9

The convergence proof of the Column Generation algorithm is due to its equivalence to the dual cutting-plane algorithm, see Lemma 4.10 in Nowak (2005). Note that the proof is not based on the computation of reduced cost $\delta_k, k \in K$, defined in (29). However, it can be used for measuring the impact of the new columns and as a criterion for algorithm termination, see Nowak (2005). For the convergence proof, we assume that all LP master problems (21) and MINLP sub-problems (28) are solved to optimality. Since Algorithm 2 is performed regarding the subset of the blocks $\hat{K}$, we ensure that main Algorithm 9 converges by performing a standard CG step in line 14 regarding all blocks. Note that the direct integration of the linear block

(14) into LP-IA master problem (25) is equivalent to performing the CG algorithm regarding that block until convergence, as shown in Lemma 1.

Proposition 1 *Let x^p be the solution of LP-IA (21) at the p -th iteration of Algorithm 9 at line 12 and v^* be the optimal value of convex hull relaxation (19). Then $\lim_{p \rightarrow \infty} c^T x^p = v^*$.*

Proof The proof is equivalent to the proof of Proposition 4.11 in Nowak (2005). $\square$

4.5.2 Convergence of Algorithm 6

Algorithm 6 combines the original Frank-Wolfe algorithm (see Algorithm 1 in Jaggi (2013)) with Nesterov update rule (Nesterov 1983). The approach proposed by Nesterov in Nesterov (1983) has a convergence rate $O(1/p^2)$, whereas the original Frank-Wolfe algorithm has a slower convergence rate of $O(1/p)$, see Theorem 1 in Jaggi (2013). In order to prove the convergence of Algorithm 6, we assume that all sub-problems (39) are solved to global optimality. Next, we state that Algorithm 6 has a convergence rate $O(1/p^2)$.

Proposition 2 *Let $v^p := Q(\tilde{w}^p, \sigma)$ be the value of the quadratic penalty function (36) at p -th iteration of Algorithm 6 and v^* be the optimal value of convex hull relaxation (19). Assume that $\sigma_i \geq |\mu_i^*|$, $i \in [m]$, where μ^* defines an optimal dual solution of (19). Then there exist a constant C such that $\forall p \geq 0$*

$$v^p - v^* \leq \frac{C}{(p+2)^2}.$$

Proof The proof is equivalent to the proof of Theorem 1 in Mouatasim and Farhaoui (2019). $\square$

5 Numerical results

In this section, we evaluate the performance of Algorithm 9 solving several DESS model instances, described in Sect. 2. Algorithm 9 was implemented with Pyomo (Hart et al. 2017), an algebraic modeling language in Python, as part of the parallel MINLP-solver (Decogo Nowak et al. 2018). In the current implementation of Decogo, the sub-problems are not solved in parallel. The solver utilizes SCIP 7.0.0 (Gamrath et al. 2020) for solving MINLP sub-problems, GUROBI 9.0.1 (Gurobi Optimization 2020) for solving MIP/LP master-problems and IPOPT 3.12.13 (Wächter and Lorenz 2006) for performing an NLP local search in master and sub-problems. All computational experiments were performed using a computer with AMD Ryzen Threadripper 1950X 16-Core 3.4 GHz CPU and 128 GB RAM. For

the experiments with DESS instances, the blocks K were defined manually using Pyomo `Block` component (Hart et al. 2017), for more details see Sect. 2.4.

For the experiments with Algorithm 9, we use the following stopping criteria, unless another is mentioned:

- Time and iteration limit in Algorithm 9 is set to 12 hrs and 20 iterations, respectively;
- The iteration limit of Algorithm 2 is set to 5;
- The iteration limit of the outer and inner loop of Algorithm 6 is set to 5 and 10, respectively;
- In Algorithm 8, the iteration limit is set to 5, the pool size to $N = 100$ and $\tau = 0.5$. To generate a solution pool with Gurobi, we used a parameter value of `POOLSEARCHMODE=1`. In this approach, the solver computes a set of N high-quality solutions including an optimal one, see Gurobi Optimization (2020) for more details;
- For SCIP we set the maximum number of processed nodes after the last improvement of the primal bound to 1000, since, for CG, it is sufficient to compute good feasible solutions, for more details see Muts et al. (2020a).

Our main research question is how to exploit the decomposable structure of DESS models to generate a new approach which may be an alternative to state-of-the-art software like BARON. During the investigation, we developed several elements to speed up the computation. To evaluate their effect, first in Sect. 5.1 we measure the effect of distinguishing the linear block and adding the fast FW approach for generating promising columns. The idea to use a pool of high-quality MIP solutions is numerically investigated in Sect. 5.2. The pool generates a group of starting points to look for feasible solutions. That does not only affect the quality of the reached best solution, but also provides alternatives for the design problem that may be analyzed by the decision maker. Finally in Sect. 5.3, we compare the outcomes and the solution process of Algorithm 9 to that of the state-of-the-art solver BARON and to the problem-tailored MIP approximation algorithm presented in Goderbauer et al. (2016).

5.1 Effect of linear block integration into LP-IA and fast CG with the Frank-Wolfe approach

To illustrate the impact of direct integration of one linear block into the LP IA master problem, as defined in (25), we compare two runs of Algorithm 9 on the same instance: The first run uses the LP-IA formulation given by (25) and the second uses the formulation given by (22). The comparison was done for instance S16L16-1, see Table 6 for more details.

For this particular instance, approximately 37% of all variables belong to the linear block. Whereas the CG algorithm for formulation (25) relies on generating linear block polytope vertices as columns, the second formulation (22) may save this effort. The difference can be observed in Fig. 4; the running time of Algorithm 9 with the

Fig. 4 Convergence of the IA objective of Algorithm 9 with and without linear block integration for instance S16L16-1

Fig. 5 Convergence of the IA objective value of Algorithm 9 for S4L4-1 with and without the fast FW column generation

direct linear block integration is drastically reduced from two days to approximately two hours. It also shows a significant improvement of the convergence speed of the IA objective value and a significant improvement of its final value.

The next question focuses on the convergence impact of including procedure FWCOLGEN (Algorithm 6) into Algorithm 9. To measure the effect, we perform two equivalent runs of Algorithm 9; one of those does not use procedure FWCOLGEN. We only focus on the convergence of the IA and don't perform procedure FINDSOLUTION (Algorithm 8). The test instance is S4L4-1, see Table 6 for more details.

Figure 5 shows that the IA objective value (25) converges faster with procedure FWCOLGEN at the initial stage of Algorithm 9. Moreover, it appears that in the beginning of the algorithm, the FWCOLGEN IA objective value is very close to the final IA objective value, i.e. after 50 seconds, the IA objective value with procedure FWCOLGEN is approximately 0.5 % worse than the final IA objective value.

Fig. 6 Solution pool for S16L16-1

Table 1 Nominal equipment sizes for all component types and unit numbers for the best solution computed with algorithm 9 for instance S16L16-1

Component type	Size fo each unit in MW				Σ in MW
	#1	#2	#3	#4	
Absorption chiller	3.09	3.09	3.09	0.87	10.14
Turbo chiller	1.47	0.94	0.94	0.94	4.29
Boiler	11.59	0.10	0.10	0.10	11.89
CHP engine	3.20	1.92	0.50	0.50	6.12

Figure 5 indicates that Algorithm 9 with procedure FWCOLGEN converges slower to the final IA objective value than without procedure FWCOLGEN. The reason for that is that FWCOLGEN fails to generate high-quality columns in the later stages of the algorithm. This is due to the fact that we have no guarantee that penalty weights σ , defined in (36), are sufficiently large. Another reason is that the procedure generates columns by solving NLP sub-problems heuristically. However, it has an advantage in speed when comparing column generation by solving MINLP sub-problems.

5.2 Impact of using the solution pool in Algorithm 8

This section focuses on using the solution pool in the heuristic procedure FINDSOLUTION (Algorithm 8). Note that using a single integer feasible point, we perform local search with the fixed NLP (42). This may imply that the result is not a feasible solution of the problem. Using a pool of starting points, may alleviate this effect.

Figure 6 shows the objective function values of all pool solutions, generated for instance S16L16-1. The solution with the best primal bound of approximately -5.15×10^7 is identified in the seventh iteration of Algorithm 9. The component configuration of this solution is specified in Table 1. In this solution, all four available units of all component types are selected. From Fig. 6, one can notice that other high-quality solutions could be computed in the earlier iterations of the algorithm. Note that the MIP solver was not always able to provide a solution pool of prescribed size of $N = 100$. Instead it computed a solution pool of smaller size. Therefore, in Fig. 6 one can observe that the number of generated feasible solutions varies over the iterations. Moreover, Fig. 6 indicates that a solution pool may contain many feasible points with a very similar objective function value. These points are distinct, since GUROBI adds only a feasible point to the solution pool if the value of

Fig. 7 DESS configurations for all solutions of the solution pool of instance S16L16-1 within a range of 2% from the solution with the best primal bound

its integer variables is different for at least one integer variable Gurobi Optimization (2020).

One advantage of the solution method is that a variety of feasible solutions are generated during the process. These can be stored in the solution pool. Often near-optimal solutions are valuable for the user to gain a more profound knowledge of the optimization problem Voll et al. (2015). In addition, if they better satisfy requirements not included in the mathematical model (e.g., safety, maintainability, and operability) near-optimal solutions might be finally selected in the design of energy systems instead of an optimal one Bejan et al. (1996). Figure 7 shows the DESS configurations of all solutions of the solution pool of instance S16L16-1 with a primal bound differing at most 2% from the best solution. In the case of instance S16L16-1, this range comprises 95 solutions. In Fig. 7 they are sorted by their primal bound values. Thus the solution from Table 1 is shown on the very left. First, it is apparent that all four available units of each component type are selected in almost all cases. In further investigations, it could be analyzed whether this indeed provides advantages regarding the objective function value or if this behavior results from the the solution procedure. For the boiler and the CHP engine, the total installed capacity is mainly focused on one or two units, and the other units are installed with a smaller or minimal size. For the cooling equipment, the total capacity tends to be more evenly distributed among all units. This may be explained by their distinct part-load behavior. The configuration of most solutions is relatively similar. In the considerably different solution numbers 10 and 12, it is indicated that a shift of the cold supply from heat-driven absorption chillers to a higher proportion of electric turbo chillers is possible with only small reductions in the objective function value. This simultaneously reduces the required boiler capacity and increases the capacity of the CHP engines to cover the increased internal electricity demand.

5.3 Comparison to other approaches

In this section, we compare the performance and solution quality of Algorithm 9 to other solvers on 12 selected instances from the DESSLib library (Bahl et al. 2016). The test instances were selected varying the number of unit components and number of load cases, as described in Sect. 2.3. Instance characteristics are reported in Table 6. The results of Algorithm 9 are compared to results obtained by the state-of-the-art MINLP solver BARON 20.4.14 (Sahinidis 2020). We also compare results provided by adaptive discretization MINLP algorithm (AdaptDiscAlgo) reported in Goderbauer et al. (2016). We do not compare the performance of Algorithm 9 to AdaptDiscAlgo, since the implementation of AdaptDiscAlgo is not publicly available. Therefore, we analyze the results by looking at the primal bounds of AdaptDiscAlgo, reported in Bahl et al. (2016). Note that AdaptDiscAlgo does not provide a valid lower bound and cannot be applied to general MINLP problems.

In this analysis, we use a 12 hour time limit for BARON and Algorithm 9. For the quality evaluation of a feasible solution, we define the gap to a reference (base) objective function value b as

Table 2 Solution quality comparison of Algorithm 9 solution v_{CG} to the primal BARON solution v_B and best known solution v^* . Note that negative value means that the primal bound has been improved. All values are given as percentage

Instances	Gap defined by (43), %					
	$(v_{CG}, \underline{v}_{CG})$	$(v_B, \underline{v}_B)$	(v_{CG}, v_B)	(v_{CG}, v^*)	After 2 main iter.	
					(v_{CG}, v_B)	(v_{CG}, v^*)
S4L4-1	7.52	0.00	0.00	0.00	0.00	0.00
S4L8-1	9.03	0.00	0.00	0.00	0.00	0.00
S4L16-1	11.29	0.00	0.34	0.34	2.32	2.32
S4L24-1	10.53	3.52	0.10	0.10	6.47	6.47
S12L4-1	28.17	37.09	1.99	2.72	2.30	3.03
S12L8-1	26.36	58.51	− 32.76	0.64	− 31.69	2.19
S12L16-1	26.14	41.05	− 1.20	− 1.20	0.65	0.65
S12L24-1	28.26	42.42	− 2.73	1.88	− 2.73	1.88
S16L4-1	32.42	44.77	0.55	0.81	0.55	0.81
S16L8-1	31.49	48.71	− 2.12	1.29	1.86	5.17
S16L16-1	33.05	49.23	− 0.20	2.95	− 0.20	2.95
S16L24-1	34.38	49.75	0.57	2.51	2.07	3.98

For numbers in bold, algorithm 9 performed best

$$\text{gap}(a, b) = -100 \frac{a - b}{\max\{|a|, |b|\} + 10^{-7}}, \quad (43)$$

where a is an objective value of the feasible solution point. Note that a “−” is used before the term in (43), since we are considering a maximization problem. In case of minimization, it is omitted.

Table 2 compares the primal solution quality of decomposition Algorithm 9 versus that of BARON and AdaptDiscAlgo using gap value (43). Let v_{CG} and v_B be the primal bound of CG Algorithm 9 and BARON, respectively. For the sake of simplicity, let v^* denote the best known primal bound among BARON and AdaptDiscAlgo. Table 2 also presents the duality gap of Algorithm 9 and BARON. Let $\underline{v}_{CG}$ denote the dual bound of Algorithm 9 defined by the objective value of the last solution of (25) and $\underline{v}_B$ denote the dual bound of BARON given by the lower bound provided by BARON.

Table 2 shows that for five instances, Algorithm 9 improves the primal bound of BARON. Even though AdaptDiscAlgo is specially tailored to DESS, the Algorithm 9 result differs at most 3 % from the best known bound and for one instance improves the solution. We also evaluated the solution after performing two main iterations with Algorithm 9. The solution quality is relatively good. This means that with a limited solution time, relatively good solutions can be generated after two iterations. For the instances with more than 500 variables, the duality gap of Algorithm 9 is smaller than that of BARON. One can also observe that the duality gap value

Table 3 Performance comparison of Column Generation Algorithm 9 and BARON

Instances	N_{iter}	Time, s	
		Alg. 9	BARON
S4L4-1	5	247.05	2.71
S4L8-1	6	571.57	34.41
S4L16-1	11	1768.54	21.83
S4L24-1	11	2084.46	43202.13
S12L4-1	10	5013.30	43203.63
S12L8-1	18	31062.51	43201.73
S12L16-1	11	43202.12	43203.14
S12L24-1	5	43200.31	43203.20
S16L4-1	6	5241.29	43202.53
S16L8-1	13	26365.72	43201.97
S16L16-1	6	43200.37	43201.73
S16L24-1	3	43200.42	43203.20

Fig. 8 Algorithm 9 computing time versus problem size of all instances in Table 3

of Algorithm 9 depends mostly on the number of unit components (parameter S) and is less sensitive to the number of load cases (parameter L).

Table 3 gives the number of iterations N_{iter} performed by Algorithm 9 and the running time of both algorithms, Algorithm 9 to BARON.

Despite the fact that Algorithm 9 does not solve the sub-problems in parallel and it is implemented with Python, Table 3 shows that for several larger instances it requires less time than BARON. Moreover, we can see that for some instances Algorithm 9 did not reach the maximum number of iterations and the time limit. This indicates that apparently the Column Generation has converged.

We are interested in the computing time with increasing size of the problem instance without varying the number of iterations. The estimated running time of Algorithm 9 after two main iterations is sketched in Fig. 8. Moreover, it sketches the actual time for initializing the Inner Approximation (all procedures before entering into main iterations of Algorithm 9). Note that the total time for finishing two iterations includes the actual time for the CG initialization. The idea of time estimation for two main iterations is based on the assumption that, for each instance, in each call of `FINDSOLUTION` (Algorithm 8), we obtain a solution pool by a MIP solver of size 500 ($500 = 5 \cdot 100$, where 5 is the prescribed number of iterations of Algorithm 8 and 100 is a prescribed size of the solution pool). In fact, the solution pool provided by a MIP solver can be smaller than the prescribed number, see Fig. 6. Therefore, the estimated time indicates how long it would take if we would need to call the NLP solver 500 times to solve fixed NLP problem (42) in the first two main iterations of Algorithm 9. This time could be also useful for estimating the time complexity of similar instances with a much larger size. From Fig. 8, one can notice that the time for initialization of Column Generation is relatively small in comparison to the estimated time after performing two main iterations of Algorithm 8. Figure 5 illustrates that the algorithm quickly computes a good CG bound in the initial stage of the algorithm. Figure 8 indicates that `FINDSOLUTION` (Algorithm 8) has a big influence on the running time of the entire algorithm.

6 Conclusions

Energy system optimization models combine potentially difficult sub-models into a global system. Due to their dimension, such systems may be difficult to solve by generic solvers based on a single branch-and-bound tree like BARON. Decomposition appears to be an appropriate concept to apply to such models due to the structure of sub-models and global constraints. Our research question is how to do this in an efficient way.

In our investigation, we looked into the potential of using Column Generation (CG). The idea is to generate feasible solutions of sub-models, defining columns of a global master problem, which is used to steer the search for a global solution. The master problem is based on an easy to solve inner approximation (IA) of a convex hull relaxation. One of our findings is, that it is more efficient to deal with blocks that have a linear structure in a separate way and include them directly in the master problem instead of generating columns for them. For speeding up column generation, we developed a fast Frank-Wolfe (FW) algorithm to generate feasible solutions of sub-problems. Solution candidates are computed by solving NLP problems with fixed integer variables regarding a solution pool of a MIP projection master problem.

Typical features of this approach for solving energy system MINLP models are: (i) no global branch-and-bound tree is used, (ii) sub-problems can be solved in parallel to generate columns, which do not have to be optimal, nor become available at the same time to synchronize the solution, (iii) an arbitrary solver can be used to generate solutions of sub-models, (iv) the approach (and the implementation) is generic and can be used to solve other nonconvex MINLP models, (v) the process

generates feasible, not necessarily optimal, solutions during the algorithm run, which may be inspected by the decision maker.

Notice that this way of working provides a generic procedure where the sub-problems may have a black-box structure. In extremis, one may modify the optimization model during the solution process. The generated columns and solutions can also be used for performing a warm-start if the model has been changed slightly, e.g. in a dynamic optimization model with a moving horizon.

Experiments with instances of several hundreds and thousands of variables show that standard software like BARON is faster for smaller problems and reaches good solutions. We should keep in mind that, in the current Python-based implementation, we do not solve the sub-problems in parallel. The interesting notice is that for the largest DESS models we obtained significantly better dual bounds and slightly better primal bounds than BARON. Hence, for larger models the presented decomposition approach is indeed able to reach better solutions for problems with thousands of variables the energy system models usually consist of.

Appendix A Model parameters and equipment constraints

For the sake of completeness, the equations of the employed technical equipment models are given below. These include the non-linear performance maps in Equations (44)–(45) and the investment cost functions in Equations (46)–(52). The applied technical parameters are given in Table 4. Moreover, Table 5 contains the economic parameters of the DESS model. All parameter values and equations are obtained from Goderbauer et al. (2016).

Part-load performance equations: (44)–(48)

Boiler ($\forall s \in B$):

Table 4 Parameter values for all types of technical equipment (minimal and maximal nominal size $\dot{V}_s^N$, annual maintenance cost share m and minimum part-load factor α)

	$\dot{V}_s^{N,\min}$	$\dot{V}_s^{N,\max}$	m_s	$\alpha_s^{\min}$
Boilers ($\forall s \in B$)	0.10 MW	14.0 MW	1.5%	0.2
CHP engines ($\forall s \in C$)	0.50 MW	3.2 MW	10.0%	0.5
Absorption chillers ($\forall s \in A$)	0.05 MW	6.5 MW	1.0%	0.2
Turbo chillers ($\forall s \in T$)	0.40 MW	10.0 MW	4.0%	0.2

Table 5 Economic parameters of the DESS model

$p^{\text{el,buy}}$	$p^{\text{el,sell}}$	$p^{\text{gas,buy}}$	i	γ^{CF}
0.16 ct/kWh	0.10 ct/kWh	0.06 ct/kWh	0.08	10 a

$$\dot{U}_s(\dot{V}_{s\ell}, \dot{V}_s^N) = \frac{1}{\eta^{N,B}} \cdot \left(c_1^B \cdot \frac{\dot{V}_{s\ell}^2}{\dot{V}_s^N} + c_2^B \cdot \dot{V}_{s\ell} + c_3^B \cdot \dot{V}_s^N \right), \quad (44)$$

$$\eta^{N,B} = 0.9, \quad c_1^B = 0.1021, \quad c_2^B = 0.8355, \quad c_3^B = 0.0666$$

Absorption chiller ($\forall s \in A$):

$$\dot{U}_s(\dot{V}_{s\ell}, \dot{V}_s^N) = \frac{1}{\text{COP}^{N,A}} \cdot \left(c_1^A \cdot \frac{\dot{V}_{s\ell}^2}{\dot{V}_s^N} + c_2^A \cdot \dot{V}_{s\ell} + c_3^A \cdot \dot{V}_s^N \right), \quad (45)$$

$$\text{COP}^{N,A} = 0.67, \quad c_1^A = 0.8333, \quad c_2^A = -0.0833, \quad c_3^A = 0.25$$

Turbo chiller ($\forall s \in T$):

$$\dot{U}_s(\dot{V}_{s\ell}, \dot{V}_s^N) = \frac{1}{\text{COP}^{N,T}} \cdot \left(c_1^T \cdot \frac{\dot{V}_{s\ell}^2}{\dot{V}_s^N} + c_2^T \cdot \dot{V}_{s\ell} + c_3^T \cdot \dot{V}_s^N \right), \quad (46)$$

$$\text{COP}^{N,T} = 5.54, \quad c_1^T = 0.8119, \quad c_2^T = -0.1688, \quad c_3^T = 0.3392$$

CHP engine ($\forall s \in C$):

$$\dot{U}_s(\dot{V}_{s\ell}, \dot{V}_s^N) = c_1^C + c_2^C \cdot \frac{\dot{V}_{s\ell}}{\dot{V}_s^N} + c_3^C \cdot \dot{V}_s^N + c_4^C \cdot \left(\frac{\dot{V}_{s\ell}}{\dot{V}_s^N} \right)^2 + c_5^C \cdot \dot{V}_{s\ell} + c_6^C \cdot (\dot{V}_s^N)^2, \quad (47)$$

$$c_1^C = 550.3, \quad c_2^C = -1328, \quad c_3^C = -0.4537,$$

$$c_4^C = 668.3, \quad c_5^C = 2.649, \quad c_6^C = 9.571 \cdot 10^{-5}$$

$$\dot{V}_s^{\text{el}}(\dot{V}_{s\ell}, \dot{V}_s^N) = c_7^C + c_8^C \cdot \frac{\dot{V}_{s\ell}}{\dot{V}_s^N} + c_9^C \cdot \dot{V}_s^N + c_{10}^C \cdot \left(\frac{\dot{V}_{s\ell}}{\dot{V}_s^N} \right)^2 + c_{11}^C \cdot \dot{V}_{s\ell} + c_{12}^C \cdot (\dot{V}_s^N)^2, \quad (48)$$

$$c_7^C = 518.8, \quad c_8^C = -1203, \quad c_9^C = -0.5361,$$

$$c_{10}^C = 579.3, \quad c_{11}^C = 1.464, \quad c_{12}^C = 7.728 \cdot 10^{-5}$$

Investment cost equations: (49)–(52)

Boiler ($\forall s \in B$):

$$I(\dot{V}_s^N) = 1.85484 \cdot \left[(11418.6 + 64.115 \cdot \dot{V}_s^{N 0.7978}) \cdot 1.046 \right. \\ \left. \cdot (1.0917 - 1.1921 \cdot 10^{-6} \cdot \dot{V}_s^N) \right] \quad (49)$$

Absorption chiller ($\forall s \in A$):

$$I(\dot{V}_s^N) = 0.50401 \cdot 17554.18 \cdot \dot{V}_s^{N 0.4345} \quad (50)$$

Turbo chiller ($\forall s \in T$):

$$I(\dot{V}_s^N) = 0.8102 \cdot V_s^N \cdot (179.63 + 4991.3436 \cdot V_s^N^{-0.6794}) \quad (51)$$

CHP engine ($\forall s \in C$):

$$I(\dot{V}_s^N) = 9332.6 \cdot \left(V_s^N \cdot \frac{\eta_s^{N,el}(\dot{V}_s^N)}{\eta_s^{N,th}(\dot{V}_s^N)} \right)^{0.539}, \quad (52)$$

$$\eta_s^{N,th}(\dot{V}_s^N) = 0.498 - 3.55 \cdot 10^{-5} \cdot \dot{V}_s^N, \quad \eta_s^{N,el}(\dot{V}_s^N) = 0.87 - \eta_s^{N,th}(\dot{V}_s^N)$$

Appendix B Instances used for numerical experiments

The following table contains characteristics of instances that were used for comparison of Algorithm 9 to BARON (Sahinidis 2020) and to the MIP approximation algorithm (Goderbauer et al. 2016) in Sect. 5.3.

Table 6 Characteristics of selected test instances. Note that $|K|$ denotes number of blocks. The number of units applies to all four component types

Instances	Units	Load cases	Number of variables			$ K $
			All	Binary	Continuous	
S4L4-1	1	4	204	28	176	9
S4L8-1	1	8	332	44	288	13
S4L16-1	1	16	588	76	512	17
S4L24-1	1	24	844	108	736	25
S12L4-1	3	4	570	84	486	25
S12L8-1	3	8	926	132	794	37
S12L16-1	3	16	1638	228	1410	49
S12L24-1	3	24	2350	324	2026	73
S16L4-1	4	4	744	112	632	33
S16L8-1	4	8	1208	176	1032	49
S16L16-1	4	16	2136	304	1832	65
S16L24-1	4	24	3064	432	2632	97

Acknowledgements This work has been funded by Grants 03ET4053A and 03ET4053B of the German Federal Ministry for Economic Affairs and Energy, Grant 01IS19079 of the Federal Ministry of Education and Research and Grant RTI2018-095993-B-I00 from the Spanish Ministry in part financed by the European Regional Development Fund (ERDF). The funding is gratefully acknowledged.

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long

as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Ahadi-Oskui T, Vigerske S, Nowak I, Tsatsaronis G (2010) Optimizing the design of complex energy conversion systems by Branch and Cut. *Comput Chem Eng* 34(8):1226–1236. <https://doi.org/10.1016/j.compchemeng.2010.03.007>
- Ashouri A, Fux SS, Benz MJ, Guzzella L (2013) Optimal design and operation of building services using mixed-integer linear programming techniques. *Energy* 59:365–376. <https://doi.org/10.1016/j.energy.2013.06.053>
- Bahl B, Goderbauer S, Arnold F, Voll P, Lübbecke M, Bardow A, Koster AM (2016) DESSLib – Benchmark instances for optimization of decentralized energy supply systems. Tech. rep., RWTH Aachen University. <http://www.math2.rwth-aachen.de/DESSLib/>
- Bejan A, Tsatsaronis G, Moran M (1996) Thermal design and optimization. Wiley, New York
- Bischi A, Taccari L, Martelli E, Amaldi E, Manzolini G, Silva P, Campanari S, Macchi E (2014) A detailed MILP optimization model for combined cooling, heat and power system operation planning. *Energy* 74:12–26. <https://doi.org/10.1016/j.energy.2014.02.042>
- Borndörfer R, Löbel A, Reuther M, Schlechte T, Weider S (2013) Rapid branching. *Public Transp* 5:3–23
- Brooks RJ, Tobias AM (1996) Choosing the best model: level of detail, complexity, and model performance. *Math Comput Model* 24(4):1–14. [https://doi.org/10.1016/0895-7177\(96\)00103-3](https://doi.org/10.1016/0895-7177(96)00103-3)
- Brown T, Schlachtberger D, Kies A, Schramm S, Greiner M (2018) Synergies of sector coupling and transmission reinforcement in a cost-optimised, highly renewable European energy system. *Energy* 160:720–739. <https://doi.org/10.1016/j.energy.2018.06.222>
- Bruche S, Tsatsaronis G (2019) A multi-stage optimization approach for energy supply systems with discrete design decisions. In: Volume 6: energy. American Society of Mechanical Engineers. <https://doi.org/10.1115/IMECE2019-11519>
- Burer S, Letchford A (2012) Non-convex mixed-integer nonlinear programming: a survey. *Surv Oper Res Manag Sci* 17(2):97–106
- Chen Q, Grossmann IE (2017) Recent developments and challenges in optimization-based process synthesis. *Annu Rev Chem Biomol Eng* 8:249–283. <https://doi.org/10.1146/annurev-chembioeng-080615-033546>
- du Merle O, Villeneuve D, Desrosiers J, Hansen P (1999) Stabilized column generation. *Discret Math* 194(1):229–237. [https://doi.org/10.1016/S0012-365X\(98\)00213-1](https://doi.org/10.1016/S0012-365X(98)00213-1)
- Elsido C, Bischi A, Silva P, Martelli E (2017) Two-stage MINLP algorithm for the optimal synthesis and design of networks of CHP units. *Energy* 121:403–426. <https://doi.org/10.1016/j.energy.2017.01.014>
- Forrest J, Vigerske S, Santos HG, Ralphs T, Hafer L, Kristjansson B, Straver E, Lubin M, Gassmann HI, Saltzman M (2020) coin-or/Cbc: Version 2.10.5. <https://doi.org/10.5281/zenodo.3700700>. <https://github.com/coin-or/Cbc>
- Gabrielli P, Gazzani M, Mazzotti M (2018) Electrochemical conversion technologies for optimal design of decentralized multi-energy systems: Modeling framework and technology assessment. *Appl Energy* 221:557–575. <https://doi.org/10.1016/j.apenergy.2018.03.149>
- Gamrath G, Anderson D, Bestuzheva K, Chen WK, Eifler L, Gasse M, Gemander P, Gleixner A, Gottwald L, Halbig K, Hendel G, Hojny C, Koch T, L Bodic, P, Maher SJ, Matter F, Miltenberger M, Mühmer E, Müller B, Pfetsch ME, Schöller F, Serrano F, Shinano Y, Tawfik C, Vigerske S, Wegscheider F, Weninger D, Witzig J (2020) The SCIP optimization suite 7.0. Technical report, Optimization Online. http://www.optimization-online.org/DB_HTML/2020/03/7705.html
- Goderbauer S, Bahl B, Voll P, Lübbecke ME, Bardow A, Koster AM (2016) An adaptive discretization MINLP algorithm for optimal synthesis of decentralized energy supply systems. *Comput Chem Eng* 95:38–48. <https://doi.org/10.1016/j.compchemeng.2016.09.008>

- Gurobi Optimization L (2020) Gurobi optimizer reference manual. <http://www.gurobi.com>
- Hart WE, Laird CD, Watson JP, Woodruff DL, Hackebeil GA, Nicholson BL, Sirola, JD (2017) Pyomo—optimization modeling in Python, vol. 67, second edn. Springer Science & Business Media, Berlin
- IBM: IBM ILOG CPLEX optimization studio (2020). <https://www.ibm.com/>
- Jaggi M (2013) Revisiting frank-wolfe: projection-free sparse convex optimization. In: Dasgupta S, McAllester D (eds) Proceedings of machine learning research, vol 28, pp 427–435. PMLR, Atlanta, Georgia, USA. <http://proceedings.mlr.press/v28/jaggi13.html>
- Jüdes M, Vigerske S, Tsatsaronis G (2009) Optimization of the design and partial-load operation of power plants using mixed-integer nonlinear programming. In: Kallrath J, Pardalos PM, Rebennack S, Scheidt M (eds) Optimization in the energy industry, energy systems. Springer, Berlin Heidelberg, Berlin, Heidelberg, pp 193–220. https://doi.org/10.1007/978-3-540-88965-6_9
- Möst D, Fichtner W, Grunwald A, (eds.) (2009) Energiesystemanalyse: Tagungsband des Workshops, “Energiesystemanalyse” vom 27. (2008) am KIT Zentrum Energie. Universitätsverlag Karlsruhe, Karlsruhe
- Mouatasim AE, Farhaoui Y (2019) Nesterov step reduced gradient algorithm for convex programming problems. Big Data and Networks Technologies. Springer International Publishing, pp 140–148. https://doi.org/10.1007/978-3-030-23672-4_11
- Muts P, Nowak I, Hendrix EMT (2020) On decomposition and multiobjective-based column and disjunctive cut generation for MINLP. Optim Eng. <https://doi.org/10.1007/s11081-020-09576-x>
- Muts P, Nowak I, Hendrix EMT (2020) A resource constraint approach for one global constraint MINLP. In: Computational science and its applications – ICCSA 2020, pp 590–605. Springer International Publishing. https://doi.org/10.1007/978-3-030-58808-3_43
- Nesterov YE (1983) A method for solving the convex programming problem with convergence rate $O(1/k^2)$. Dokl Akad Nauk SSSR 269:543–547
- Nowak I (2005) Relaxation and decomposition methods for mixed integer nonlinear programming. Birkhäuser, Basel
- Nowak I (2014) Outer-point generation - a decomposition algorithm for solving MINLPs. scip.zib.de/workshop/scip_nowak.pdf. SCIP Workshop, Berlin
- Nowak I, Breittfeld N, Hendrix EMT, Njacheun-Njanzoua G (2018) Decomposition-based inner- and outer-refinement algorithms for global optimization. J Global Optim 72(2):305–321
- Nowak I, Muts P, Hendrix EMT (2019) Multi-tree decomposition methods for large-scale mixed integer nonlinear optimization. Springer Optimization and Its Applications. Springer International Publishing, pp 27–58
- Rieder A, Christidis A, Tsatsaronis G (2014) Multi criteria dynamic design optimization of a small scale distributed energy system. Energy 74:230–239. <https://doi.org/10.1016/j.energy.2014.06.007>
- Sahinidis NV (2020) BARON 20.4.14: global optimization of mixed-integer nonlinear programs. User's Manual. <http://www.minlp.com/>
- Shor N (1985) Minimization methods for non-differentiable functions. Springer, Berlin
- Teichgraber H, Brodrick PG, Brandt AR (2017) Optimal design and operations of a flexible oxyfuel natural gas plant. Energy 141:506–518. <https://doi.org/10.1016/j.energy.2017.09.087>
- Vigerske S (2018) MINLPlib. <http://minlpilib.org/index.html>
- Voll P, Jennings M, Hennen M, Shah N, Bardow A (2015) The optimum is not enough: a near-optimal solution paradigm for energy systems synthesis. Energy 82:446–456. <https://doi.org/10.1016/j.energy.2015.01.055>
- Voll P, Klaffke C, Hennen M, Bardow A (2013) Automated superstructure-based synthesis and optimization of distributed energy supply systems. Energy 50:374–388. <https://doi.org/10.1016/j.energy.2012.10.045>
- Wächter A, Lorenz BT (2006) On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math Program 106(1):25–57
- Wang L, Yang Y, Dong C, Morosuk T, Tsatsaronis G (2014) Parametric optimization of supercritical coal-fired power plants by MINLP and differential evolution. Energy Convers Manage 85:828–838. <https://doi.org/10.1016/j.enconman.2014.01.006>
- Yokoyama R, Hasegawa Y, Ito K (2002) A MILP decomposition approach to large scale optimization in structural design of energy supply systems. Energy Convers Manage 43(6):771–790. [https://doi.org/10.1016/S0196-8904\(01\)00075-9](https://doi.org/10.1016/S0196-8904(01)00075-9)

Authors and Affiliations

Pavlo Muts¹ · Stefan Bruche² · Ivo Nowak¹ · Ouyang Wu¹ · Eligius M. T. Hendrix³ · George Tsatsaronis²

Pavlo Muts
pavlo.muts@haw-hamburg.de

Stefan Bruche
stefan.bruche@tu-berlin.de

Ouyang Wu
ouyang.wu@haw-hamburg.de

Eligius M. T. Hendrix
eligius@uma.es

George Tsatsaronis
georgios.tsatsaronis@tu-berlin.de

¹ Hamburg University of Applied Sciences, Hamburg, Germany

² Technische Universität Berlin, Berlin, Germany

³ Universidad de Málaga, Málaga, Spain