2024 European Control Conference (ECC)
June 25-28, 2024. Stockholm, Sweden

A Numerical Study on the Parallelization of Dual Decomposition-based Distributed Mixed-Integer Programming

Mario Klostermeier[1], Vassilios Yfantis[1], Achim Wagner[2] and Martin Ruskowski[1,2]

***Abstract*— The shift from centralized to decentralized systems is increasing the complexity of many problems in control and optimization. However, it also presents the opportunity to exploit parallelized computational schemes. This paper shows how the solution process of mixed-integer problems, which often arise in areas like production scheduling or logistics, can be supported by employing parallel computations. To this end, dual variables are introduced that enable the decomposition of these complex problems into subproblems that can then be solved in parallel. The presented dual decomposition-based approach provides a lower bound for the optimal solution of the original problem, which can support the overall solution process. The focus of this paper is on the parallelizability of the computation of this lower bound. The bounds from three different dual decomposition-based distributed optimization algorithms are compared to the lower bounds provided by several commercial solvers within their branch-&-cut framework.**

I. Introduction

The digital landscape continues to evolve from central to decentralized structures. As a result, the relevance of multi-agent systems continues to increase. Such systems consist of several agents that pursue a common goal, achievable only through the cooperative and coordinated efforts of the agents. At the same time, each agent also pursues its own goal. The global goal of the system is broken down into subproblems, which are then assigned to an agent. The autonomy of the agents is crucial here, further improving the overall system's flexibility and resilience. Potential applications of these systems span areas such as manufacturing [1], network systems [2], the process industry [3], and aerospace [4].

Many practical applications in these areas consist of both continuous and discrete components. The resulting mixed-integer problems can be challenging due to their combinatorial nature and inherent non-convexity. Often, these problems are solved using the branch-&-cut (B&C) algorithm. In practice, this approach incurs significant computational and memory overheads, as an optimization problem must be solved at each node. With increasingly complex problems and the ever-growing volume of data [5], this effort intensifies. Hence, efficient methods are crucial for solving such problems. Enhancing the efficiency of B&C can be achieved by finding better problem bounds. Particularly in cases of weak relaxations, searching for a lower bound becomes a bottleneck in the solution process. This paper provides a numerical study on the parallelizability of the computation of lower bounds for constraint-coupled mixed-integer programs via dual decomposition-based distributed optimization. The remainder of this paper is structured as follows: Sec. II introduces dual decomposition for Mixed-Integer Programming (MIP) as well as the examined distributed optimization algorithms. Sec. III presents a case study in the form of a quadratic order slotting problem. Numerical results are presented in Sec. IV and the paper is concluded in Sec. V.

[1]The author is with the Chair of Machine Tools and Control Systems, Department of Mechanical and Process Engineering, University of Kaiserslautern-Landau, Kaiserslautern D-67663, Germany (e-mail: mario.klostermeier@rptu.de)

[2] The author is with the German Research Center for Artificial Intelligence (DFKI), Kaiserslautern D-67663, Germany

II. Dual Decomposition for MIP

This paper focuses on problems of the form

$$\min \sum_{i\in\mathcal{I}} f_i(\mathbf{x}_i,\mathbf{z}_i) \tag{1a}$$

$$\text{s. t. } \sum_{i\in\mathcal{I}} \mathbf{C}_i\mathbf{x}_i + \mathbf{D}_i\mathbf{z}_i \leq \mathbf{b} \tag{1b}$$

$$\mathbf{x}_i \in \mathcal{X}_i,\ \mathbf{z}_i \in \mathcal{Z}_i\ \forall i \in \mathcal{I}, \tag{1c}$$

which are composed of a set of subproblems, denoted by $\mathcal{I} = \{1,\dots,N_S\}$. Each individual subproblem has its own objective function

$$f_i(\mathbf{x}_i,\mathbf{z}_i) = \frac{1}{2}\mathbf{x}_i^T\mathbf{Q}_{\mathbf{x},i}\mathbf{x}_i + \mathbf{q}_{\mathbf{x},i}^T\mathbf{x}_i + \frac{1}{2}\mathbf{z}_i^T\mathbf{Q}_{\mathbf{z},i}\mathbf{z}_i + \mathbf{q}_{\mathbf{z},i}^T\mathbf{z}_i \tag{2}$$

and its own variables $\mathbf{x}_i \in \mathbb{R}^{n_{\mathbf{x}_i}}$ and $\mathbf{z}_i \in \mathbb{Z}^{n_{\mathbf{z}_i}}$. The individual subproblems are coupled by global constraints (1b). These can be interpreted as the consumption and production of shared limited resources. $\mathbf{C}_i\mathbf{x}_i + \mathbf{D}_i\mathbf{z}_i$, with $\mathbf{C}_i \in \mathbb{R}^{n_\mathbf{b}\times n_{\mathbf{x}_i}}$ and $\mathbf{D}_i \in \mathbb{R}^{n_\mathbf{b}\times n_{\mathbf{z}_i}}$, reflects the consumption or production of resources by the subsystem i as a function of $\mathbf{x}_i$ and $\mathbf{z}_i$. The global availability of resources is represented by $\mathbf{b} \in \mathbb{R}^{n_\mathbf{b}}$. An example of such a resource would be energy which must be shared between several machines in a production environment. The subproblems are also subject to local constraints $\mathbf{x}_i \in \mathcal{X}_i$ and $\mathbf{z}_i \in \mathcal{Z}_i$, where $\mathcal{X}_i$ and $\mathcal{Z}_i$ are compact polyhedral sets. The objective of the entire system results from the sum of all objectives of the subproblems. One way to decompose such a problem is the dual decomposition method. For this, the dual variables $\boldsymbol{\lambda} \in \mathbb{R}^{n_\mathbf{b}}$ are introduced and the Lagrange function for the problem is formulated as

$$\mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\lambda}) := \sum_{i\in\mathcal{I}} f_i(\mathbf{x}_i,\mathbf{z}_i) + \boldsymbol{\lambda}^T \sum_{i\in\mathcal{I}} (\mathbf{C}_i\mathbf{x}_i + \mathbf{D}_i\mathbf{z}_i) - \boldsymbol{\lambda}^T\mathbf{b}. \tag{3}$$

Through this, the global constraints are relaxed and weighted. Based on the Lagrange function, the dual function

$$d(\boldsymbol{\lambda}) := \inf_{\mathbf{x},\mathbf{z}} \mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\lambda}) \tag{4a}$$

$$\text{s.t. } \mathbf{x}_i \in \mathcal{X}_i,\ \mathbf{z}_i \in \mathcal{Z}_i,\ \forall i \in \mathcal{I} \tag{4b}$$

can then be defined. A key property of the dual function, utilized in this work, is its provision of a lower bound to the objective function of the system-wide problem (1) [6]. Thus, the dual function provides a bound for the optimal solution, $\mathbf{x}^*, \mathbf{z}^*$. To find the optimal lower bound for problem (1), the dual optimization problem

$$\max_{\boldsymbol{\lambda} \in \mathbb{R}^{n_b}} d(\boldsymbol{\lambda}) \tag{5a}$$

$$\text{s. t. } \boldsymbol{\lambda} \geq 0 \tag{5b}$$

has to be solved. The original problem (1) is referred to as the primal problem. The solution of the dual problem is amenable to distributed computations since the dual function can be evaluated by solving the optimization problems

$$\min f_i(\mathbf{x}_i, \mathbf{z}_i) + \boldsymbol{\lambda}^T(\mathbf{C}_i\mathbf{x}_i + \mathbf{D}_i\mathbf{z}_i) \tag{6a}$$

$$\text{s. t. } \mathbf{x}_i \in \mathcal{X}_i, \mathbf{z}_i \in \mathcal{Z}_i \tag{6b}$$

in a distributed manner. Note that the term $\boldsymbol{\lambda}^T\mathbf{b}$ is known only to the coordinator. Depending on the dual variables, certain local constraints may become active or inactive. If the set of active constraints changes, nondifferentiabilities in the dual function arise [7]. The following sections present different algorithms for the soultuon of the dual problem (5).

A. Subgradient Method

A subgradient is a generalization of the gradient for non-smooth functions. For the dual function a subgradient can be computed when solving problem (5) by evaluating the coupling constraints (1b). In the k-th iteration, this depends on the optimal solutions $\mathbf{x}_i^*(\boldsymbol{\lambda}^{(k)}), \mathbf{z}_i^*(\boldsymbol{\lambda}^{(k)})$ of the subproblems,

$$\mathbf{g}(\boldsymbol{\lambda}^{(k)}) = \sum_{i \in \mathcal{I}} (\mathbf{C}_i\mathbf{x}_i^*(\boldsymbol{\lambda}^{(k)}) + \mathbf{D}_i\mathbf{z}_i^*(\boldsymbol{\lambda}^{(k)})) - \mathbf{b} \tag{7}$$

The primal and dual variables are then updated in each iteration k according to

$$\boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\lambda}^{(k)} + \alpha^{(k)}\mathbf{g}(\boldsymbol{\lambda}^{(k)}). \tag{8}$$

The step size in the direction of the subgradient is denoted by $\alpha^{(k)}$. A clear advantage of this method is its simplicity, as it is easy to implement and requires a small amount of computing capacity and memory.

B. Bundle Trust Method

Another approach is the Bundle Trust Method (BTM), which belongs to the bundle methods. In this group of algorithms, the dual function is approximated by a piece-wise linear function, which is also called a cutting plane model. The basis of this model is the information from the previous iterations which is stored in a bundle

$$\mathcal{B}^{(k)} = \{(\boldsymbol{\lambda}^{(j)}, d(\boldsymbol{\lambda}^{(j)}), \mathbf{g}(\boldsymbol{\lambda}^{(j)})) \in \mathbb{R}^{n_b} \times \mathbb{R} \times \mathbb{R}^{n_b} | \; 1 \leq j \leq k\} \tag{9}$$

The cutting plane model

$$\hat{d}^{(k)}(\boldsymbol{\lambda}) = \min_{j \in \mathcal{J}^{(k)}} d(\boldsymbol{\lambda}^{(k)}) + \mathbf{g}^T(\boldsymbol{\lambda}^{(j)})(\boldsymbol{\lambda} - \boldsymbol{\lambda}^{(k)}) - \beta^{(j,k)}, \tag{10}$$

with the linearization error

$$\beta^{(j,k)} = d(\boldsymbol{\lambda}^{(k)}) - d(\boldsymbol{\lambda}^{(j)}) - \mathbf{g}^T(\boldsymbol{\lambda}^{(j)})(\boldsymbol{\lambda}^{(k)} - \boldsymbol{\lambda}^{(j)}) \tag{11}$$

is based on the set $\mathcal{J}^{(k)} \subset \{1, \ldots, k\}$.
The aim of the BTM is to find a search direction $\mathbf{s} \in \mathbb{R}^{n_b}$ in each iteration.
Given its piece-wise linear characteristics, the optimization problem is convertible into a smooth quadratically constrained direction-finding problem [8]:

$$\max v \tag{12a}$$

$$\text{s. t. } \|\mathbf{s}\|_2^2 \leq \alpha^{(k)} \tag{12b}$$

$$\mathbf{g}^T(\boldsymbol{\lambda}^{(j)})\mathbf{s} - \beta^{(j,k)} \geq v, \; \forall j \in \mathcal{J}^{(k)} \tag{12c}$$

$$\boldsymbol{\lambda}^{(k)} + \mathbf{s} \geq \mathbf{0}. \tag{12d}$$

C. Quasi-Newton Dual Ascent

Another approach that approximates the dual function is the Quasi-Newton Dual Ascent (QNDA) method. The dual function is approximated at the current iterate $\boldsymbol{\lambda}^{(k)}$ as

$$\tilde{d}^{(k)}(\boldsymbol{\lambda}) = \frac{1}{2}(\boldsymbol{\lambda} - \boldsymbol{\lambda}^{(k)})^T\mathbf{B}^{(k)}(\boldsymbol{\lambda} - \boldsymbol{\lambda}^{(k)}) + \mathbf{g}^T(\boldsymbol{\lambda}^{(k)})(\boldsymbol{\lambda} - \boldsymbol{\lambda}^{(k)}) + d(\boldsymbol{\lambda}^{(k)}). \tag{13}$$

To determine the matrix $\mathbf{B}$, the established Broyden–Fletcher–Goldfarb–Shanno (BFGS) method is used [9]. In each iteration k, the matrix $\mathbf{B}^{(k)}$ is updated with

$$\mathbf{B}^{(k)} = \mathbf{B}^{(k-1)} + \frac{\mathbf{y}^{(k)}\mathbf{y}^{(k),T}}{\mathbf{y}^{(k),T}\mathbf{s}^{(k)}} - \frac{\mathbf{B}^{(k-1)}\mathbf{s}^{(k)}\mathbf{s}^{(k),T}\mathbf{B}^{(k-1),T}}{\mathbf{s}^{(k),T}\mathbf{B}^{(k)}\mathbf{s}^{(k)}}, \tag{14}$$

Where $\mathbf{y}$ describes the change in the subgradient and $\mathbf{s}$ the change in the dual variables.

$$\mathbf{y}^k = \mathbf{g}(\boldsymbol{\lambda}^{(k)}) - \mathbf{g}(\boldsymbol{\lambda}^{(k-1)}) \tag{15a}$$

$$\mathbf{s}^k = \boldsymbol{\lambda}^{(k)} - \boldsymbol{\lambda}^{(k-1)} \tag{15b}$$

The resulting function from equation (13) is then a smooth approximation of the dual function. Based on this, the update problem for the dual variables $\boldsymbol{\lambda}^{(k)}$ can be formulated as

$$\max_{\boldsymbol{\lambda} \in \mathbb{R}^{n_b}} \tilde{d}^{(k)}(\boldsymbol{\lambda}), \tag{16a}$$

$$\text{s. t. } \tilde{d}^{(k)}(\boldsymbol{\lambda}) \leq d(\boldsymbol{\lambda}^{(j)}) + \mathbf{g}^T(\boldsymbol{\lambda}^{(j)})(\boldsymbol{\lambda} - \boldsymbol{\lambda}^{(j)}), \quad \forall j \in \mathcal{J}^{(k)}, \tag{16b}$$

$$\|\boldsymbol{\lambda} - \boldsymbol{\lambda}^{(k)}\|_2^2 \leq \alpha^{(k)} \tag{16c}$$

$$\boldsymbol{\lambda}^{(k)} \geq \mathbf{0}. \tag{16d}$$

Constraints (16b) are refered to as bundle cuts and they are used to take the non-smoothess of the dual function into account while employing a smooth approximation [7].

III. Quadratic Order-slotting Problem

In the following section, the parallelization capabilities of the algorithms presented earlier will be assessed through a case study. For this purpose, the mathematical formulation of the problem instances is presented first. Then, the strategy for generating different benchmark problem instances is outlined.

A. Mathematical Model

The case study deals with the distribution of orders within a modern production network, inspired by the concept of shared production detailed in [10]. In this model, the network is comprised of a multitude of producers, each acting as an autonomous agent within a centralized coordination framework tasked with order allocation. In this illustrative example, each agent represents an individual factory within the network. The collective aim is to efficiently allocate a set of orders, denoted by $\mathcal{P}$, across these factories, $\mathcal{F}$. Each factory is equipped with a specific number of production slots, $\mathcal{S}$, to which orders can be assigned. The primary decision variable in this model is the discrete quantity of goods produced from order p in slot s of factory f, represented by $z_{fsp} \in \mathbb{N}_0$. This variable captures the essence of the allocation and production process within the network. The production of a unit of order p in factory f necessitates a specific amount of production capacity, denoted by Cap_{fp}. This capacity requirement varies based on the production processes and machinery employed within each factory, reflecting the unique operational characteristics of each production site. The total utilized production capacity of slot s in factory f is denoted by the continuous variable $x_{fs} \in \mathbb{R}_0^+$ and is modeled as

$$x_{fs} = \sum_{p \in \mathcal{P}} z_{fsp} \cdot Cap_{fp}, \forall f \in \mathcal{F}, s \in \mathcal{S}. \tag{17}$$

Each individual agent then utilizes a capacity for each of their slots s. This means that only a limited amount of goods can be produced in each slot, depending on the maximum available capacity:

$$x_{fs} \leq Cap_{fp}^{\max}, \forall f \in \mathcal{F}, s \in \mathcal{S}, p \in \mathcal{P}. \tag{18}$$

The objective of each agent consists of several objectives which are presented in the following. The first objective is to maximize turnover, which depends on the order. A turnover T_p is achieved through the production of a unit of each product. Since the entire problem is to be formulated as a minimization problem, a minus sign is applied to the term. At the same time, the costs for the production of each order are to be minimized. These costs $Cost_{fp}$ depend on the type of product and also differ between factories.

$$o_{1,f} = -\sum_{s \in \mathcal{S}} \sum_{p \in \mathcal{P}} z_{fsp} \cdot (T_p - Cost_{fp}). \forall f \in \mathcal{F} \tag{19}$$

The previous targets maximized profit for the entire production network. The quality of the products should also be improved. From the past, it is already known which producer f produces which product p with which quality level $Qual_{fp}$. Additionally, a target quality level $Qual_p^{\text{target}}$ is determined for each order. A quadratic objective can then be used to minimize the deviation from the target quality,

$$o_{2,f} = \sum_{s \in \mathcal{S}} \sum_{p \in \mathcal{P}} (z_{fsp} \cdot (Qual_{fp} - Qual_p^{\text{target}})^2), \forall f \in \mathcal{F}. \tag{20}$$

To avoid overproduction, a constraint is defined for each order, depending on the demand Dem_p of the respective product:

$$\sum_{f \in \mathcal{F}} \sum_{s \in \mathcal{S}} z_{fsp} \leq Dem_p, \forall p \in \mathcal{P}. \tag{21}$$

This is the coupling constraint that connects the systems with each other.

B. Data generation

The quantity of factories, denoted as $|\mathcal{F}|$, and consequently the number of subproblems, were varied to suit different experimental setups. Each factory was allocated $|\mathcal{S}| = 3$ time slots, and a total of $|\mathcal{P}| = 100$ orders were required to be fulfilled across the network. To introduce variability and complexity into the problem, all other parameters were generated using random distributions: continuous uniform distributions, denoted by $\mathcal{U}_c[a, b]$, and discrete uniform distributions, represented as $\mathcal{U}_d\{a, ..., b\}$. The specific distributions employed for each parameter are detailed in Table I.

TABLE I: Distributions for the randomly generated Mixed-Integer Quadratic Programming (MIQP) parameters.

Name	Parameter	Distribution
Capacity	Cap_{fp}	$\mathcal{U}_c[5, 10]$
Max Capacity	$Cap_{fp}^{\max}$	$\mathcal{U}_c[10, 20]$
Turnover	T_p	$\mathcal{U}_c[50, 100]$
Cost	$Cost_{fp}$	$\mathcal{U}_c[1, 10]$
Quality level	$Qual_{fp}$	$\mathcal{U}_d\{1, ..., 10\}$
Quality target	$Qual_p^{\text{target}}$	$\mathcal{U}_d\{1, ..., 10\}$
Demand	Dem_p	$\mathcal{U}_d\{20, ..., 50\}$

IV. Numerical Results

In the following, the algorithms presented in Sec. II are evaluated using the model presented in Sec. III. To evaluate the decomposition-based approach, a lower bound was first determined by a MIP solver during the solution process of the original problem (1). Subsequently, the problem was decomposed and then a lower bound was determined by the decomposition. For better comparability, the respective subproblems (6a) were also solved with the same solver. The commercial solvers Gurobi V.10 [11], CPLEX V.22.1.1 [12] and Mosek V.10 [13] were used as MIP solvers in this experiment. The default parameters of the MIP solvers were always used. These solvers employ a B&C algorithm and thus also compute a lower bound from the QP relaxations. The goal is to compare the lower bound obtained within the B&C algorithm to the bounds computed by the dual decomposition-based distributed optimization algorithms and assess the impact of parallelization on the latter. The algorithms were implemented in the Julia programming language [14] and all involved optimization problems were modelled with the JuMP [15] package. An Intel 13900K with 128GB@5600MHz RAM with 24 physical cores was used for the calculation. To illustrate the potential of the previously presented method regarding its parallelizability, experiments with a different number of workers were conducted to assess the benefits of parallelization for the presented algorithms. The available cores were equally distributed among the workers which solve the subproblems (6) in

parallel. All algorithms were initialized with $\boldsymbol{\lambda}^{(0)} = \mathbf{0}$ and an initial step size/trust region of $\alpha^{(0)} = 0.5$, which was then varied according to

$$\alpha^{(k)} = \alpha^{(0)}/k. \tag{22}$$

The maximum size of the bundle for the BTM and QNDA was set to 25 and the approximated Hessian of QNDA was initialized with the negative identity matrix.

A. Impact of the problem size

The size of the problems is decisive for their complexity and also complicates the solvability of many problems. The decomposition method shown above is intended to help with problems that are difficult to solve due to their sheer size. To demonstrate this, 15 different problems were created with different numbers of subproblems $|\mathcal{F}| \in \{150, 200, 250\}$. Fig. 1 illustrates the mean difference between the bound found by the MIP solvers in the B&C algorithm while solving the original problem (1) and the bound found by the three dual decomposition-based algorithms by solving the dual problem (5) after 100 *sec.* For this experiment, 24 workers were selected so that each had exactly one core to solve the subproblems. The percentage difference between the bounds was determined as follows:

$$b_{Diff} = (1 - \frac{b_C}{b_{DD}}) \times 100. \tag{23}$$

Here, b_{DD} denotes the best bound which was found with the decomposition-based method and b_C the best bound which was found with the respective MIP solver. Note that for the problems with $|\mathcal{F}| = 250$ factories (subproblems), Gurobi could not find a bound as the root node could not be solved within the given time limit. The subgradient method and the QNDA algorithm provide similar bounds which are consistently better than the bounds provided by the solvers. Furthermore, the relative difference between the bounds increases as the number of factories increases. In the case of BTM the quality of the bound decreases compared to the relaxation-based one as the subproblems increase. More subproblems also increase the number of dual variables, which could indicate that the BTM is more suitable for problems with few variables.

Fig. 2 shows the average time needed to find a better bound for the different algorithms. Again, the values for Gurobi are missing for $|\mathcal{F}| = 250$ subproblems because no comparison was possible. Note that the dual decomposition-based algorithms can provide a bound in these cases. The values for the BTM algorithm are also missing here if the Mosek solver was used to solve the subproblems because no better bound could be found within the 100 *sec.* Similarly, no better bound could be found at $|\mathcal{F}| = 250$ with the BTM algorithm when CPLEX was used. It can be seen that the time increases with the number of factories as more subproblems have to be solved in each step. It is noticeable that when using the Gurobi and CPLEX solvers, the time increases less when more subproblems have to be solved than with the Mosek solver. As can be seen in Sec. IV-D, the

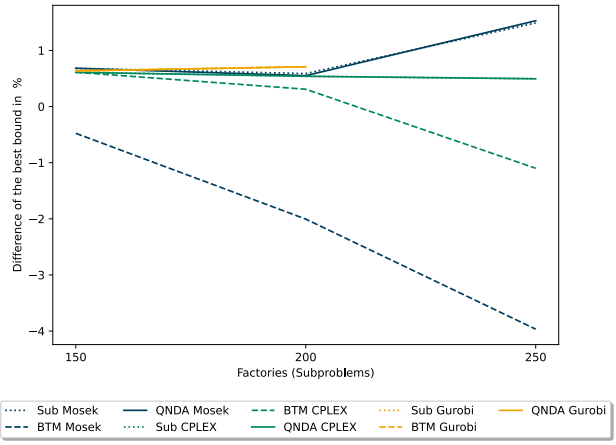

Fig. 1: Difference in bound between solving the original problem and the three algorithms for each of the three solvers.

Mosek solver is least suitable for solving the subproblems. This could be because the Mosek solver requires more CPU cores to solve the problems. However, this is only the case with the subgradient method and QNDA. These two also took less time to solve the subproblems. It should be noted that the solvers also need more time to find the bound of the original problem due to the higher complexity of the problem and the resulting harder problems in the root node and the nodes of the B&C tree.

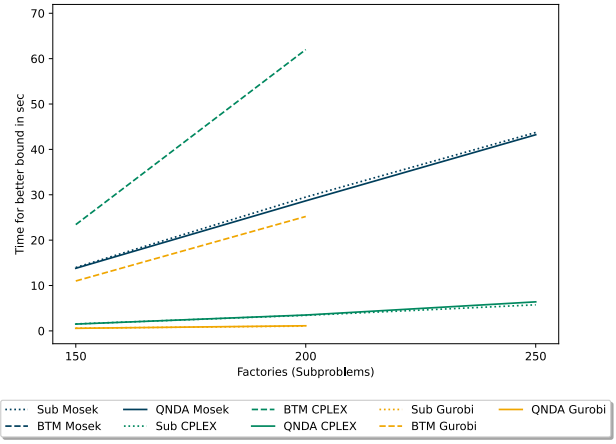

Fig. 2: Time needed to find a better lower bound for each algorithm depending on the number of subproblems

B. Parallelizability of the subproblems

An advantage of the dual decomposition approach is that the solution of the subproblems can be parallelized, which facilitates their distribution among different computational units. To investigate this behavior, the lower bound was again computed for in a second experiment for 15 problems with

$|\mathcal{F}| = 200$. A lower bound was first computed by the three commercial MIP solvers which had the 24 threads of the CPU at their disposal. Note that state-of-the-art MIP solvers exploit parallelism in several ways, e.g., by parallelizing the solution of the node problems. The subproblems were then solved in parallel with the Julia distributed computing package. The number of workers and the number of threads per worker were varied. The threads were distributed equally among all workers. Thus, the number of threads per worker decreases when more problems are solved in parallel. The

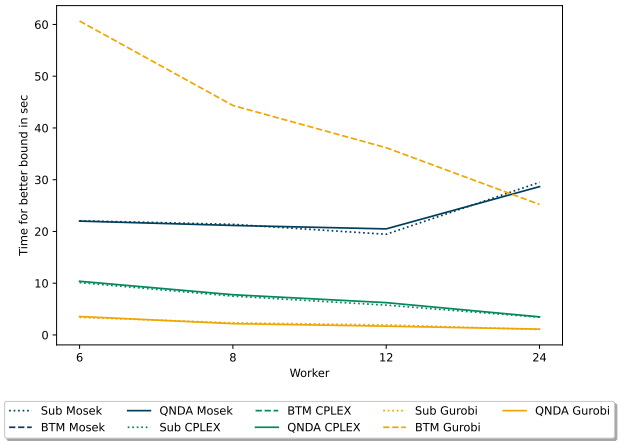

Fig. 3: Time needed to find a better lower bound for each algorithm depending on the number of workers

QNDA algorithm and the subgradient method also showed their advantages in this experiment. Fig. 3 shows the average time needed to find a better lower bound. For the Gurobi and the CPLEX solver it can be seen that with more workers a better bound can be found faster. This indicates that the subproblems can be solved quickly and with less computational effort. This means that they can be parallelized well. This is not the case for Mosek with 24 workers, which indicates the benefit of parallelization is offset by the increased solution times of the subproblems when only a single core is used per worker. Another advantage is that the QNDA algorithm and the subgradient method deliver more consistent results compared to the BTM, as shown in Fig. 4. The figure indicates the difference between the best lower bound found with dual decomposition and the lower bound found with the solver when solving the original problem. When using the BTM, the result can be improved by using more workers with CPLEX and Gurobi. Mosek initially improves by increasing the number of workers, but with 24 workers with one thread each, the result deteriorates again.

C. Solving subproblems

The difference between the solvers can also be seen in the number of subproblems they solve per second. By solving the subproblems faster, more iterations of the three algorithms can be executed and a better bound can be found faster.

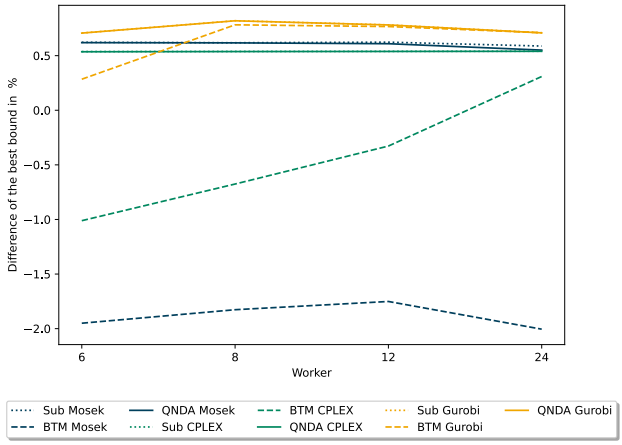

Fig. 4: Comparison of the bound depending on the number of workers.

Likewise, a better bound can also be found overall. The results are shown in Fig. 5, where the number of workers was also varied. It was shown that the Gurobi solver could solve the most subproblems in this case, regardless of the number of workers. The number of subproblems solved per second increases with the number of workers. The CPLEX solver shows the same behavior. However, it solves fewer problems than the Gurobi solver. This also applies to the Mosek solver. This also stands out because it solves fewer subproblems with more workers.

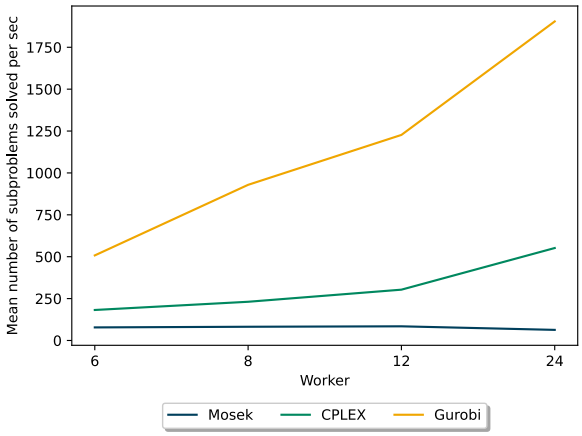

Fig. 5: Comparison of the solvers regarding their ability to solve the subproblems as fast as possible.

D. Communication rounds

To compare the decomposition-based algorithms, an important property is how many iterations they require to find a good lower bound. The more iterations are needed, the more subproblems have to be solved, which leads to a higher computational effort and thus resource consumption. If the

subproblems are distributed to physically separated computing units, communication between them is also necessary in each iteration. Therefore, an efficient algorithm should need as few iterations as possible to find a better bound. To investigate this, problems were generated with $|\mathcal{F}| = 150$ subproblems and then again a bound was first sought with the MIP solvers. Then the number of iterations required by the algorithms to find a better bound was measured. For this, 24 workers were used. The results for this can be seen in Tab. II. Since the BTM algorithm with the Mosek solver could not find a better solution, the value is missing here. It can be seen that the QNDA algorithm requires the fewest iterations. In comparison, the subgradient method requires the most iterations. Fig. 1 and 2 are also interesting, as they show that the two methods provided similar results in terms of the quality of the bound and the time required to find a better bound.

TABLE II: Required number of iterations to find a better lower bound.

	Sub	BTM	QNDA
CPLEX	382	272	249
Gurobi	1198	862	446
Mosek	47	–	41

V. Conclusion and Outlook

In this paper, lower bounds for MIP problems consisting of multiple subproblems were computed using dual decomposition-based distributed optimization algorithms. It was described how such systems can be decomposed by introducing dual variables and how the resulting non-smooth optimization problem can be solved by the subgradient method, BTM or the QNDA algorithm. This approach was examined on a problem from the field of production planning, where different production orders are to be allocated to production slots in different factories. The results were compared with the solution of three commercial MIP solvers (CPLEX, Gurobi, Mosek). Here, the subgradient method and the QNDA algorithm show that a better bound for such problems can be found more quickly than with the MIP solvers. The BTM algorithm showed the worst performance of the three algorithms. This could be due to, among other things, the case study under consideration, as the BTM algorithm approximates the dual function by a piecewise linear function. However, since the case study involves a quadratic problem, this can result in a high approximation error. At the same time, an optimization problem must be solved in each iteration which means that each iteration requires significantly more time than with the subgradient method. Additionally, the algorithms were able to find a better lower bound than the three solvers within the same computation time. It was also investigated which of these solvers is best suited for the application of the dual decomposition method. A decisive characteristic for this is the ability of the solver to solve the subproblems quickly, as well as good scalability when solving them. In a real application, it would make sense to solve the subproblems on a computing cluster. In such a system, the communication between the computing nodes would be crucial. To minimize this, an algorithm should be selected which can find a good bound with the fewest iterations. It was shown that the QNDA algorithm is best suited for this, as it requires the least iterations on average. In the future, it will be investigated how such a bound can improve the solution process of integer problems. The use of a custom lower bound in the solution process of a solver can lead to a less efficient search within the tree in the B&C procedure. However, the lower bound can still be used without affecting the B&C search by including a custom termination criterion based on the achieved duality gap. This provides a worst-case distance to the global optimum, similar to the integrality gap used by MIP solvers. Furthermore, dual decomposition could be used as a heuristic within specific nodes of the search tree to quickly compute bounds and potentially prune the branch if the obtained bound is tighter than the bound provided by the relaxation and larger than the incumbent.

References

[1] J. Popper, V. Yfantis, and M. Ruskowski, "Simultaneous Production and AGV Scheduling using Multi-Agent Deep Reinforcement Learning," *Procedia CIRP*, vol. 104, pp. 1523–1528, 2021.

[2] F. Bullo, *Lectures on Network Systems*. Kindle Direct Publishing Seattle, DC, USA, 2020, vol. 1.

[3] M. Baldea, T. F. Edgar, B. L. Stanley, and A. A. Kiss, "Modular manufacturing processes: Status, challenges, and opportunities," *AIChE Journal*, vol. 63, no. 10, pp. 4262–4272, 2017.

[4] T. Lei, Z. Min, Q. Gao, L. Song, X. Zhang, and X. Zhang, "The Architecture Optimization and Energy Management Technology of Aircraft Power Systems: A Review and Future Trends," *Energies*, vol. 15, no. 11, p. 4109, 2022.

[5] M. Niño, F. Sáenz, J. M. Blanco, and A. Illarramendi, "Requirements for a big data capturing and integration architecture in a distributed manufacturing scenario," in *14th International Conference on Industrial Informatics (INDIN)*, Jul. 2016, pp. 1326–1329.

[6] J. Nocedal and S. Wright, *Numerical Optimization*, ser. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.

[7] V. Yfantis, S. Wenzel, A. Wagner, M. Ruskowski, and S. Engell, "Hierarchical distributed optimization of constraint-coupled convex and mixed-integer programs using approximations of the dual function," *EURO Journal on Computational Optimization*, vol. 11, p. 100058, 2023.

[8] M. Mäkelä, "Survey of Bundle Methods for Nonsmooth Optimization," *Optimization Methods & Software - OPTIM METHOD SOFTW*, vol. 17, pp. 1–29, 2002.

[9] D. Bertsekas, *Nonlinear Programming*, 2003.

[10] M. Simon, J. Hermann, S. Jungbluth, A. Witton, M. Volkmann, A. Belyaev, C. Urban, C. Diedrich, P. Rübel, and M. Ruskowski, "Realisierung einer Shared Production: Integration von Plattform Industrie 4.0 und Gaia-X-Konzepten," *atp magazin*, vol. 65, no. 6-7, pp. 99–109, 2023.

[11] "Gurobi Optimizer Reference Manual." Gurobi Optimizer LCC, 2023.

[12] "IBM ILOG CPLEX Reference Manual," IBM Corporation, 2023.

[13] "Mosek Reference Manual," MOSEK ApS, 2023.

[14] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A Fresh Approach to Numerical Computing," Jul. 2015.

[15] M. Lubin, O. Dowson, J. D. Garcia, J. Huchette, B. Legat, and J. P. Vielma, "JuMP 1.0: Recent improvements to a modeling language for mathematical optimization," *Mathematical Programming Computation*, vol. 15, no. 3, pp. 581–589, Sep. 2023.