Journal of Global Optimization (2020) 77:75–96
https://doi.org/10.1007/s10898-020-00888-x

The decomposition-based outer approximation algorithm for convex mixed-integer nonlinear programming

Pavlo Muts[1] · Ivo Nowak[1] · Eligius M. T. Hendrix[2]

Received: 15 November 2018 / Accepted: 4 February 2020 / Published online: 20 February 2020

Abstract
This paper presents a new two-phase method for solving convex mixed-integer nonlinear programming (MINLP) problems, called Decomposition-based Outer Approximation Algorithm (DECOA). In the first phase, a sequence of linear integer relaxed sub-problems (LP phase) is solved in order to rapidly generate a good linear relaxation of the original MINLP problem. In the second phase, the algorithm solves a sequence of mixed integer linear programming sub-problems (MIP phase). In both phases the outer approximation is improved iteratively by adding new supporting hyperplanes by solving many easier sub-problems in parallel. DECOA is implemented as a part of Decogo (Decomposition-based Global Optimizer), a parallel decomposition-based MINLP solver implemented in Python and Pyomo. Preliminary numerical results based on 70 convex MINLP instances up to 2700 variables show that due to the generated cuts in the LP phase, on average only 2–3 MIP problems have to be solved in the MIP phase.

Keywords Global optimization · Decomposition method · Convex MINLP · Outer approximation · DECOA

1 Introduction

Many optimization problems arising in engineering and science contain both combinatorial and nonlinear relations. Such optimization problems are modeled by mixed-integer nonlinear programming (MINLP), which combines capabilities of mixed-integer linear programming (MILP) and nonlinear programming (NLP). The ability to accurately model real-world problems has made MINLP an active research area with a large number of industrial applications.

✉ Pavlo Muts
pavlo.muts@haw-hamburg.de

Ivo Nowak
ivo.nowak@haw-hamburg.de

Eligius M. T. Hendrix
eligius@uma.es

1 Hamburg University of Applied Sciences, Hamburg, Germany

2 University of Málaga, Málaga, Spain

A large collection of real-world MINLP problems can be found in MINLPLib [33]. In this paper, we consider a subclass of MINLP problems where the feasible set is defined by integrality restrictions and convex nonlinear functions.

1.1 Known solution methods

1.1.1 Convex MINLP-methods

There are several well-known methods for solving convex MINLP problems, *e.g.*, generalized Benders decomposition [15], outer approximation (OA) [10], branch-and-bound [7], extended cutting plane (ECP) [36] and extended supporting hyperplane (ESH) [20].

Most of the current MINLP deterministic solvers are based on the branch-and-bound (BB) algorithm [4,6], in particular branch-and-cut, like ANTIGONE [26], BARON [31], Couenne [1], Lindo API [23] and SCIP [32]. Other methods based on BB are branch-cut-and-price [9], branch-decompose-and-cut [30] and branch-and-refine [22]. Although these methods have found a lot of applications, they can be computationally very demanding, due to a rapidly growing global search tree, which may prevent the method to find an optimal solution in a reasonable time.

In contrast to BB, successive approximation methods solve an optimization problem without using a single global search tree. The outer approximation (OA) method [10,12], the extended cutting plane (ECP) algorithm [36] and extended supporting hyperplane (ESH) algorithm [20] solve convex MINLPs by successive linearization of nonlinear constraints. A comparison of several solvers for convex MINLP [19] reveals that the SHOT (ESH-based) solver [20] and the AOA (OA-based) solver [18] have the best performance. Improvement of polyhedral outer approximations using extended formulations significantly reduces the number of OA iterations [25]. Generalized Benders Decomposition (GBD) [13,15] solves a convex MINLP by iteratively solving NLP and MIP sub-problems. The adaptive MIP OA-method is based on the refinement of MIP relaxations by projecting infeasible points onto a feasible set, see [5,8].

1.1.2 Decomposition methods

Decomposition is a general method that can be applied to convex optimization, as well as nonconvex optimization. The idea of decomposition is based on dividing an original problem into smaller and easier sub-problems. Mainly the approach of these methods consists in solving small sub-problems and then feeding the result to a global master problem. In this case, the sub-problems can be solved simultaneously, which makes decomposition methods very attractive in terms of computational demand. Decomposition can be applied along a number of dimensions, like time-windows, resources or system components. Most decomposition methods are based on solving a Lagrangian relaxation of the decomposed problem [11,14,21], e.g. Column Generation (CG) [24]. Rapid Branching is an efficient CG-based heuristic for solving large-scale transport planning problems [3,28].

1.2 The new solution approach

This paper describes a decomposition-based successive outer approximation algorithm (DECOA) for convex MINLP problems. Like the OA method, the ESH algorithm, the ECP algorithm, and the adaptive MIP algorithm, DECOA constructs MIP outer approximations by linearization of nonlinear functions. The key difference to these well-known approaches is that DECOA uses a decomposition-based cut generation, i.e. supporting hyperplanes are constructed only by solving small sub-problems in parallel.

DECOA uses projection as a basic type of cut generation, *i.e.* infeasible points are projected onto the feasible set by solving small sub-problems. The algorithm also uses a line search procedure (like ESH) in order to generate additional supporting hyperplanes. A detailed description of DECOA is given in Sect. 3. Note that in Algorithm 3 of [29], a variant of DECOA has been presented, which, in contrast to DECOA, solves non-convex MINLPs by adapting break-points without using projection steps.

DECOA is implemented as a part of the MINLP solver Decogo (Decomposition-based Global Optimizer). Preliminary results of the implementation are presented.

1.3 Outline of the paper

This paper is structured as follows. In Sect. 2, the definition of block-separable MINLP and the notation are given. Section 3 presents the new decomposition-based outer approximation (DECOA) algorithm. A proof of convergence is given in Sect. 4. In Sect. 5, the implementation of DECOA is briefly described. Preliminary results of DECOA on convex MINLPs of the MINLPLib are presented in Sect. 6. We summarize findings and discuss possible next steps in Sect. 7.

2 Block-separable reformulation of MINLP

DECOA solves convex *block-separable* (or *quasi-separable*) MINLP problems of the form

$$\min c^T x \quad \text{s.t.} \quad x \in P,\ x_k \in X_k,\ k \in K \tag{1}$$

with

$$\begin{aligned} P &:= \{x \in [\underline{x}, \overline{x}] :\ a_j^T x \le b_j,\ j \in J\} \\ X_k &:= G_k \cap P_k \cap Y_k, \end{aligned} \tag{2}$$

where

$$\begin{aligned} G_k &:= \{y \in \mathbb{R}^{n_k} : g_{kj}(y) \le 0,\ j \in [m_k]\}, \\ P_k &:= \{y \in [\underline{x}_k, \overline{x}_k] :\ a_{kj}^T y \le b_{kj},\ j \in J_k\}, \\ Y_k &:= \{y_k \in \mathbb{R}^{n_k} : y_{ki} \in \mathbb{Z}, i \in I_k\}. \end{aligned} \tag{3}$$

The vector of variables $x \in \mathbb{R}^n$ is partitioned into $|K|$ blocks such that $n = \sum_{k \in K} n_k$, where n_k is the dimension of the k-th block, and $x_k \in \mathbb{R}^{n_k}$ denotes the variables of the k-th block. The vectors $\underline{x}, \overline{x} \in \mathbb{R}^n$ determine the lower and upper bounds on the variables.

The linear constraints defining the feasible set P are called *global*. The constraints defining the feasible set X_k are called *local*. The set X_k consists of the set G_k of m_k *local nonlinear constraints*, set P_k of $|J_k|$ *local linear constraints* and set Y_k of *integrality constraints*. In this paper, it is assumed that all the local nonlinear constraint functions $g_{kj} : \mathbb{R}^{n_k} \to \mathbb{R},\ j \in [m_k]$ are bounded, continuously differentiable and convex within the set $[\underline{x}_k, \overline{x}_k]$. Global linear constraints P are defined by $a_j \in \mathbb{R}^n, b_j \in \mathbb{R}, j \in J$ and local linear constraints P_k are defined by $a_{kj} \in \mathbb{R}^{n_k}, b_{kj} \in \mathbb{R}, j \in J_k$. The set Y_k defines the set of integer values of variables $x_{ki}, i \in I_k$, where I_k is an index set. The linear objective function is defined by $c^T x := \sum\limits_{k \in K} c_k^T x_k, c_k \in \mathbb{R}^{n_k}$.

Furthermore, we define sets

$$G := \prod_{k \in K} G_k, \quad Y := \prod_{k \in K} Y_k, \quad X := \prod_{k \in K} X_k. \tag{4}$$

The block-sizes n_k can have an influence on the performance of a decomposition algorithm. It is possible to reformulate a general sparse MINLP defined by factorable functions g_{kj} as a block-separable optimization problem with a given maximum block-size n_k by adding new variables and copy-constraints [27,31,32]. It has been shown that a MINLP can be reformulated as a separable program, where the size of all blocks is one. However, a reformulation may not preserve the convexity of constraints. A natural block-separable reformulation preserving the convexity of constraints is given by connected components of the Hessian adjacency graph, see (23).

3 DECOA

DECOA iteratively solves and improves an outer approximation (OA) problem, where the convex nonlinear set G is approximated by finitely many hyperplanes. In each iteration, the outer approximation is refined by generating new supporting hyperplanes. Due to the block-separability of the problem (1), the sample points for supporting hyperplanes are obtained by solving low-dimensional sub-problems. DECOA consists of two parts: *LP phase* and *MIP phase*. In the LP phase, the algorithm initializes the outer approximation of set G by solving a linear programming outer approximation (LP-OA) master problem. In the MIP phase, the algorithm refines the outer approximation of set G by solving a mixed-integer programming outer approximation (MIP-OA) master problem. In the end, the final MIP-OA master problem is a reformulation of problem (1). In the following subsections we describe the master problems and sub-problems and outline the basic version of DECOA. In the end, we describe the full DECOA algorithm with all improvements.

3.1 OA master problem

DECOA obtains solution estimate $\hat{x}$ by solving an *OA* master problem defined by

$$\begin{aligned} &\min \ c^T x, \\ &\text{s.t.}\ x \in P,\ x_k \in \widehat{X}_k,\ k \in K, \end{aligned} \tag{5}$$

where $\widehat{X}_k \supseteq X_k$ is a polyhedral outer approximation of set X_k. Note that $\widehat{X} := \prod_{k \in K} \widehat{X}_k$. The polyhedral outer approximation $\widehat{G}_k \supseteq G_k$ of convex nonlinear set G_k is defined by

$$\widehat{G}_k = \{x \in \mathbb{R}^{n_k} : \breve{g}_{kj}(x) \le 0,\ j \in [m_k]\}, \tag{6}$$

where

$$\breve{g}_{kj}(x) := \max\{\nabla g_{kj}(\hat{y})^T (x - \hat{y}) : \hat{y} \in T_k \subset \mathbb{R}^{n_k}\}. \tag{7}$$

T_k is a set of sample points and $\breve{g}_{kj}(x)$ denotes a piecewise linear underestimator of function g_{kj}. Supporting hyperplanes are defined by a linearization at sample point $\hat{y} \in T_k$. Note that the linearizations are computed only for active nonlinear constraints at point $\hat{y} \in T_k$, *i.e.* $g_{kj}(\hat{y}) = 0$. Furthermore, we define $\widehat{G} := \prod_{k \in K} \widehat{G}_k$.

Note that OA (5) can be infeasible, if the given MINLP model (1) is infeasible, e.g. because of data or model errors. Since most MIP solvers, like SCIP, are able to detect the infeasibility of a model, a feasibility flag can be returned after solving (5), which can be used to stop DECOA, if the MINLP model (1) is infeasible.

3.2 Basic DECOA

In this subsection we describe the basic version of DECOA. The refinement procedure is performed only by solving projection sub-problems. Iteratively, the algorithm computes a solution estimate $\hat{x}$ by solving MIP-OA master problem (5) defined by

$$\widehat{X}_k := Y_k \cap P_k \cap \widehat{G}_k, \ k \in K. \tag{8}$$

After solving the MIP-OA master problem, *projection* sub-problem (9) is solved for each $k \in K$

$$\begin{aligned} \hat{y}_k = \operatorname{argmin} &\|x_k - \hat{x}_k\|^2, \\ \text{s.t.} \quad & x_k \in G_k \cap P_k, \end{aligned} \tag{9}$$

where $\hat{x}_k$ is the k-th part of the solution $\hat{x}$ of MIP-OA problem (8). The solution $\hat{y}_k$ is used for updating the outer approximation $\widehat{G}$ by generating new supporting hyperplanes as defined in (7).

Algorithm 1 Basic DECOA

```
1: for $k \in K$ do $\widehat{G}_k \leftarrow \mathbb{R}^{n_k}$
2: repeat
3:     $\hat{x} \leftarrow$ SOLVEMIPOA$(P, \widehat{X})$
4:     for $k \in K$ do $\widehat{G}_k \leftarrow$ ADDPROJECTCUTS$(\hat{x}_k, P_k, G_k)$
5: until stopping criterion
```

Algorithm 1 describes the basic version of DECOA. Iteratively it solves MIP-OA master problem (8) by calling procedure SOLVEMIPOA. Then the algorithm calls procedure ADDPROJECTCUTS for the refinement of set $\widehat{G}$. It performs a projection from point $\hat{x}$ onto the feasible set by solving sub-problems (9) and adds linearization cuts at solution points $\hat{y}_k$. The algorithm iteratively performs these steps until a stopping criterion is fulfilled.

Theorem 1 proves that Algorithm 1 converges to the global optimum of problem (1). However, starting solving the MIP-OA (8) from scratch would be computationally demanding. In order to speed up the convergence, we design an algorithm which reduces the number of times a MIP-OA master problem has to be solved. The improved DECOA algorithm is presented in the two following subsections.

3.3 The LP phase

In order to generate rapidly an initial outer approximation $\widehat{G}$ and to reduce the number of iterations in the MIP phase, DECOA iteratively solves the LP-OA master problem and improves it by solving small sub-problems. *LP-OA* master problem (5) is defined by

$$\widehat{X}_k := P_k \cap \widehat{G}_k,\ \ k \in K. \tag{10}$$

To further improve the quality of set $\widehat{G}$, the following *line search* sub-problem can be solved for each $k \in K$

$$\begin{aligned}(\hat{\alpha}_k, \hat{y}_k) = \operatorname{argmax} \alpha,\\ \text{s.t. }\ & x = \alpha \hat{x}_k + (1-\alpha)\breve{x}_k,\\ & x \in G_k \cap P_k,\\ & \alpha \in [0, 1],\end{aligned} \tag{11}$$

where $\hat{x}_k$ is the k-th part of the solution $\hat{x}$ of LP-OA master problem (10) and $\breve{x}_k$ is an interior point of set $G_k \cap P_k$. The obtained solution point $\hat{y}$ is an additional support point for improving outer approximation $\widehat{G}$.

For solving line search sub-problems (11), one has to obtain an interior point $\breve{x}$. We consider the following NLP problem

$$\begin{aligned}\breve{x} = \operatorname{argmin} s,\\ \text{s.t. }\ & x \in P,\\ & x_k \in P_k,\\ & g_{kj}(x_k) \le s,\ \ j \in [m_k],\ k \in K,\ \ s \in \mathbb{R}.\end{aligned} \tag{12}$$

Note that problem (12) is convex, since the functions $g_{kj}(x_k) - s \le 0$ are convex. Given that the original problem (1) has a solution, then problem (12) also has a solution, i.e. $\breve{x} \in P \cap \prod_{k\in K} G_k \cap P_k$. It is important that point $\breve{x}$ is contained within the interior of set $P \cap \prod_{k\in K} G_k \cap P_k$. If point $\breve{x}$ lies on the boundary of set $P \cap \prod\limits_{k\in K} G_k \cap P_k$, the solution of problem (11) will always be the same, i.e. supporting hyperplanes will always be the same. In practice, the interior point $\breve{x}$ can be obtained by solving integer-relaxed NLP problem (1), where the objective function is a constant (zero), using an interior point-based NLP solver, such as IPOPT [34].

Algorithm 2 LP phase of DECOA

```
1: function OaStart
2:     for $k \in K$ do $\widehat{G}_k \leftarrow \mathbb{R}^{n_k}$
3:     repeat
4:         $\hat{x} \leftarrow$ solveLpOA($P, \widehat{X}$)
5:         for $k \in K$ do $\widehat{G}_k \leftarrow$ addProjectCuts($\hat{x}_k, P_k, G_k$)
6:     until no improvement
7:     $\breve{x} \leftarrow$ solveNLPZeroObj($\hat{x}, P, X$)
8:     repeat
9:         for $k \in K$ do $\widehat{G}_k \leftarrow$ addProjectCuts($\hat{x}_k, P_k, G_k$)
10:        for $k \in K$ do $\widehat{G}_k \leftarrow$ addLineSearchCuts($\hat{x}_k, \breve{x}_k, P_k, G_k$)
11:        $\hat{x} \leftarrow$ solveLpOA($P, \widehat{X}$)
12:    until no improvement
13:    $(\tilde{x}, \widehat{G}) \leftarrow$ addUnfixedNlpCuts($\hat{x}, P, X$)
14:    return $(\hat{x}, \breve{x}, \widehat{G})$
```

Algorithm 2 describes the LP phase of the DECOA algorithm for a rapid initialization of the polyhedral outer approximation. At the beginning, it solves the LP-OA master problem defined in (10) by calling procedure solveLpOA, and the projection sub-problems (9), and then adds linearization cuts at solution point $\hat{y}$. This loop, which is described in lines 3-5, is performed until there is no improvement, *i.e.* $c^T(\hat{x}^p - \hat{x}^{p+1}) < \varepsilon$, where ε is a desired tolerance.

Then, in order to conduct the line search, the algorithm finds the interior point $\breve{x}$ by calling the procedure SOLVENLPZEROOBJ. This procedure solves an NLP problem, obtained by relaxing the integrality constraints of problem (1), where the objective function is a constant (zero). Then the algorithm performs a similar loop as before, described in lines 7–10, with the procedure ADDLINESEARCHCUTS($\hat{x}, \breve{x}$). This procedure solves the line search sub-problems (11) between the LP-OA solution point $\hat{x}$ and the interior point $\breve{x}$, and adds linearization cuts at solution point $\hat{y}$ of the line search sub-problems. Finally, the algorithm calls the procedure ADDUNFIXEDNLPCUTS which computes a solution $\tilde{x}$ of integer-relaxed NLP problem (1) and adds linearization cuts at solution point $\tilde{x}$.

3.4 MIP phase

Once a good initial outer approximation has been obtained through the LP phase, the algorithm considers the integer constraints Y_k by defining the MIP-OA master problem (8). After the first solution estimate $\hat{x}$ has been obtained by solving MIP-OA master problem (8), DECOA computes a solution candidate $\tilde{x}$ by solving NLP master problem *with fixed integer variables* defined by

$$\begin{aligned} &\min\ c^T x, \\ &\text{s.t. } x \in P \cap X, \\ &\quad x_{ki} = \hat{x}_{ki},\ i \in I_k,\ k \in K, \end{aligned} \tag{13}$$

where $\hat{x}$ is the solution of MIP-OA master problem (8) and I_k is the set of integer variables in k-th block. Notice that if the outer approximation $\widehat{X}$ is still not close to set X, (13) does not necessarily yield a feasible solution.

Algorithm 3 Decomposition-based outer approximation algorithm

```
1: function OASOLVE
2:     $\overline{v} \leftarrow \infty$
3:     $x^* \leftarrow \emptyset$
4:     $(\hat{x}, \breve{x}, \widehat{G}) \leftarrow$ OASTART($P, X$)
5:     $\hat{x} \leftarrow$ SOLVEMIPOA($P, \widehat{X}$)
6:     repeat
7:         $(\tilde{x}, \widehat{G}) \leftarrow$ ADDFIXEDNLPCUTS($\hat{x}, P, X$)
8:         if $\tilde{x} \in X$ and $c^T\tilde{x} < \overline{v}$ then
9:             $x^* \leftarrow \tilde{x}$
10:            $\overline{v} \leftarrow c^T\tilde{x}$
11:            if $\overline{v} - c^T\hat{x} < \varepsilon$ then
12:                return $(\hat{x}, x^*, \widehat{G})$
13:        for $k \in K$ do $\widehat{G}_k \leftarrow$FIXANDREFINE($\tilde{x}_k, P, \widehat{X}_k$)
14:        for $k \in K$ do $\widehat{G}_k \leftarrow$ADDPROJECTCUTS($\hat{x}_k, P_k, G_k$)
15:        for $k \in K$ do $\widehat{G}_k \leftarrow$ADDLINESEARCHCUTS($\hat{x}_k, \breve{x}_k, P_k, G_k$)
16:        $\hat{x} \leftarrow$SOLVEMIPOA($P, \widehat{X}$)
17:    until $\overline{v} - c^T\hat{x} < \varepsilon$
18: return $(\hat{x}, x^*, \widehat{G})$
```

If solution point $\tilde{x}$ of problem (13) improves the best solution candidate, *i.e.* $\tilde{x} \in X$ and improves the upper bound of objective function value, then point $\tilde{x}$ is a new solution candidate of problem (1), which is denoted by x^*. Moreover, if the objective function value $c^T x^*$ is less than the current upper bound $\overline{v}$, we set $\overline{v}$ to $c^T x^*$.

In order to further refine outer approximation $\widehat{G}$ by exploiting the block-separability property of problem (1), we consider *partly-fixed* OA problems which are defined similar to MIP-OA problem (8), but the variables are fixed for all blocks except for one, *i.e.* for all $k \in K$:

$$
\begin{aligned}
\min\ & c^T x, \\
\text{s.t. } & x \in P \cap \widehat{X}, \\
& x_{mi} = \tilde{x}_{mi}, i \in n_m, m \in K \setminus \{k\},
\end{aligned}
\tag{14}
$$

where $\tilde{x}$ is a solution point of NLP problem (13).

The solution points of problem (14) can be used for the refinement of outer approximation $\widehat{G}$ as a base for solving projection sub-problem (9). Note that the solution of problem (14) provides us information about the fixation of integer variables in problem (13). If the fixations in problem (13) are feasible, then problem (14) has a feasible solution, otherwise problem (14) does not have a feasible solution, because global constraints P are not satisfied.

Algorithm 3 describes DECOA which computes solution estimate $\hat{x}$ by solving MIP-OA master problem (8) and solution candidate x^* by solving the NLP master problem with fixed integers (13). At the beginning, upper bound $\overline{v}$ of the optimal value of problem (1) and solution candidate x^* are set to ∞ and to $\emptyset$, respectively. Since the goal is to reduce the number of MIP-solver runs, the algorithm calls procedure OaStart, described in Algorithm 2 for initializing a good outer approximation. The procedure solveMipOA computes a solution estimate $\hat{x}$ by solving MIP-OA master problem (8).

When the first solution estimate $\hat{x}$ has been obtained, DECOA starts the main loop described in lines 5–18. At the beginning of the loop, procedure addFixedNlpCuts is called, which solves the NLP master problem with fixed integers (13). This procedure uses solution estimate $\hat{x}$ for integer variables fixations and returns solution point $\tilde{x}$, which might not be feasible. If the point $\tilde{x}$ is feasible and the objective function value $c^T \tilde{x}$ is lower than the current upper bound $\overline{v}$, the solution candidate x^* and the upper bound $\overline{v}$ are updated accordingly. Moreover, if the objective function gap between solution estimate $\hat{x}$ and solution candidate x^* is small enough, *i.e.* $\overline{v} - c^T \hat{x} < \varepsilon$, the algorithm stops. These steps are described in lines 8-12.

If the objective function gap between solution estimate $\hat{x}$ and solution candidate x^* is not closed, DECOA improves the outer approximation $\widehat{G}$ by generating new supporting hyperplanes. For refinement of set $\widehat{G}$, DECOA calls fixAndRefine which solves partly-fixed OA problem (14). The detailed description of this procedure is given in Algorithm 4. Like in Algorithm 2, in order to obtain sample points for new supporting hyperplanes, line search sub-problems (11) and projection sub-problems (9) are solved. The projection and line search sub-problems are solved using the solution point $\hat{x}$ of MIP-OA master problem (8). After refinement of set $\widehat{G}$, DECOA calls solveMipOa for computing a new solution estimate $\hat{x}$ by solving the problem (8). If the gap between the points $\hat{x}$ and point x^* is closed, DECOA terminates and returns solution estimate $\hat{x}$, solution candidate x^* and polyhedral outer approximation $\widehat{G}$, which is a reformulation of original problem (1).

Algorithm 4 Cut generation per block

```
1: function FIXANDREFINE($\tilde{x}_k, P, X_k$)
2:     repeat
3:         $\hat{x}_k \leftarrow$ SOLVEFIXMIPOA($\tilde{x}_k, P, \widehat{X}_k$)
4:         $\widehat{G}_k \leftarrow$ADDPROJECTCUTS($\hat{x}_k, P_k, G_k$)
5:     until integer variables of $\hat{x}$ are not changed
6: return ($\hat{x}, \widehat{G}$)
```

Algorithm 4 describes the function FIXANDREFINE which is used for refinement of set $\widehat{G}$. For each block $k \in K$, the function calls procedure SOLVEFIXMIPOA which solves partly-fixed OA master problem (14). Then the obtained solution point $\hat{x}$ is used for solving the projection sub-problems and adding linearization cuts by calling procedure ADDPROJECTCUTS. This procedure repeats until the integer variables of solution point $\hat{x}$ are not changed.

4 Proof of convergence

In this section, it is proven that basic DECOA as depicted in Algorithm 1 either converges to a global optimum of (1) in a finite number of iterations or generates a sequence which converges to a global optimum. In order to prove the convergence, it is assumed that all MIP-OA master problems (5), (8) and the projection sub-problem (9) are solved to optimality. We also prove the convergence of improved DECOA as outlined in Algorithm 3.

Due to the convexity, function $\breve{g}_{kj}(x)$ defined in (7) is an affine underestimator of function g_{kj} and, therefore, set $\widehat{X}^p$ consisting of the corresponding hyperplanes at iteration p is an outer approximation of set X. Since basic DECOA adds new supporting hyperplanes in each iteration, it creates a sequence of sets $\widehat{X}^p$ with the following property

$$\widehat{X}^0 \supset \ldots \supset \widehat{X}^{p-1} \supset \widehat{X}^p \supset X \tag{15}$$

Lemma 1 *If DECOA described in Algorithm 1 stops after $p < \infty$ iterations and the last solution $\hat{x}^p$ of OA master problem (5) fulfills all constraints of (1), the solution is also an optimal solution of the original problem (1).*

Proof We adapt the proof of [20]. Since DECOA stops at iteration p, $\hat{x}^p$ is an optimal solution of (5) and $\hat{x}^p$ has the optimal objective function value of (1) within $\widehat{X}^p \cap P$. From property (15) it is clear that $\widehat{X}^p$ also includes the feasible set X. Since $\hat{x}^p$ also satisfies the nonlinear and integrality constraints, it is also in the feasible set, *i.e.*, $\hat{x}^p \in P \cap X$. Because $\hat{x}^p$ minimizes the objective function within $\widehat{X}^p \cap P$, which includes the entire feasible set, and $\hat{x}^p \in P \cap X$, it is also an optimal solution of (1). □

In Theorem 1 we prove that Algorithm 1 generates a sequence of solution points converging to a global optimum. In order to prove this, we present intermediate results in Lemmas 2–5.

Lemma 2 *If current solution $\hat{x}^p \notin G$, Algorithm 1 excludes it from set $\widehat{X}^{p+1}$, i.e. $\hat{x}^p \notin \widehat{X}^{p+1}$.*

Proof Given that $\hat{x}^p \notin G$, $\exists(k, j)$ such that $g_{kj}(\hat{x}_k^p) > 0$. This means that for the solution $\hat{y}_k$ of (9) $\hat{y}_k \neq \hat{x}_k^p$. Note that $\hat{y}_k, \hat{x}_k^p \in P_k$. For this proof, we set $\tilde{G}_k := G_k \cap P_k = \{y \in \mathbb{R}^{n_k} : \tilde{g}_{kj}(y) \leq 0,\ j \in [\tilde{m}_k],\ \tilde{m}_k = |m_k| + |J_k|\}$ and, in (9), replace $G_k \cap P_k$ by $\tilde{G}_k$. Note that the linearization cuts of P_k are not added, since they are the same as linear constraints P_k. Hence, only linearization cuts of nonlinear constraints G_k are added.

Let $\mathcal{A}_k$ be the set of indices of active constraints at $\hat{y}_k$ of $\tilde{G}_k$, *i.e.* $\tilde{g}_{kj}(\hat{y}_k) = 0,\ j \in \mathcal{A}_k$. According to the KKT conditions of projection sub-problem (9), $\exists \mu_j \geq 0,\ j \in \mathcal{A}_k$, such that

$$\hat{x}_k^p - \hat{y}_k = \sum_{j \in \mathcal{A}_k} \mu_j \nabla \tilde{g}_{kj}(\hat{y}_k) \tag{16}$$

where μ correspond to constraints $\tilde{G}_k$. Multiplying (16) by $\hat{x}_k^p - \hat{y}_k$ we obtain

$$\left(\sum_{j \in \mathcal{A}_k} \mu_j \nabla \tilde{g}_{kj}(\hat{y}_k) \right)^T (\hat{x}_k^p - \hat{y}_k) = ||\hat{x}_k^p - \hat{y}_k||^2 > 0. \tag{17}$$

Given that $\mu_j \geq 0,\ j \in \mathcal{A}_k$, there exists at least one $j \in \mathcal{A}_k$ for which $\nabla \tilde{g}_{kj}(\hat{y}_k)^T(\hat{x}_k^p - \hat{y}_k) > 0$. As Algorithm 1 adds the cut $\nabla \tilde{g}_{kj}(\hat{y}_k)^T(x_k - \hat{y}_k) \leq 0$ to $\widehat{X}^{p+1}$ we have that $\hat{x}_k^p \notin \widehat{X}^{p+1}$. □

In Lemma 3 we show that if Algorithm 1 does not stop in a finite number of iterations, the sequence of solution points contains at least one convergent subsequence $\{\hat{x}^{p_i}\}_{i=1}^{\infty}$, where

$$\{p_1, p_2, \ldots\} \subseteq \{1, 2, \ldots\} \quad \text{and} \quad \{\hat{x}^{p_i}\}_{i=1}^{\infty} \subseteq \{\hat{x}^p\}_{p=1}^{\infty}.$$

Since subsequence $\{\hat{x}^{p_i}\}_{i=1}^{\infty}$ is convergent, there exists a limit $\lim_{i \to \infty} \hat{x}^{p_i} = z$. In Lemmas 4 and 5 , we show that z is not only within the feasible set of (1) but also an optimal solution of (1).

Lemma 3 *If Algorithm 1 does not stop in a finite number of iterations, it generates a convergent subsequence $\{\hat{x}^{p_i}\}_{i=1}^{\infty}$.*

Proof We adapt the proof of [20]. Since the algorithm has not terminated, none of the solutions of OA master problem (5) are in the feasible set, i.e., $\hat{x}^p \notin P \cap X$ for all $p = 1, 2, \ldots$ in the solution sequence. Therefore, all the points in the sequence $\{\hat{x}^p\}_{p=1}^{\infty}$ will be distinct due to Lemma 2. Since $\{\hat{x}^p\}_{p=1}^{\infty}$ contains an infinite number of different points, and all are in the compact set P, according to the Bolzano–Weierstrass Theorem, the sequence contains a convergent subsequence. □

Lemma 4 *The limit z of any convergent subsequence $\{\hat{x}^{p_i}\}_{i=1}^{\infty}$ generated in Algorithm 1 belongs to the feasible set of (1).*

Proof Let $\hat{x}_k^{p_j}$ and $\hat{x}_k^{p_{j+1}}$ be the points from sequence $\{\hat{x}_k^{p_i}\}_{i=1}^{\infty}$ and $\hat{y}^{p_j}$ is the sample point obtained by solving projection sub-problem (9) with point $\hat{x}_k^{p_j}$. Consider the following equality

$$\begin{aligned} ||\hat{x}_k^{p_j} - \hat{x}_k^{p_{j+1}}||^2 &= ||(\hat{x}_k^{p_j} - \hat{y}_k^{p_j}) - (\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j})||^2 \\ &= ||\hat{x}_k^{p_j} - \hat{y}_k^{p_j}||^2 + ||\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j}||^2 \\ &\quad - 2(\hat{x}_k^{p_j} - \hat{y}_k^{p_j})^T(\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j}). \end{aligned} \tag{18}$$

Consider the set $\tilde{G}_k$ of the proof of Lemma 2 containing the set of all constraints. Let $\mathcal{A}_k$ be the set of indices of active constraints $\tilde{G}_k$ at $\hat{y}_k^{p_j}$, *i.e.* $\tilde{g}_{ki}(\hat{y}_k^{p_j}) = 0,\ i \in \mathcal{A}_k$. Note that only linearization cuts of G_k are added. Since Algorithm 1 adds for each active nonlinear constraint $i \in \mathcal{A}_k$ the cut

$$\nabla \tilde{g}_{ki}(\hat{y}_k^{p_j})^T(x_k - \hat{y}_k^{p_j}) \leq 0, \tag{19}$$

we have

$$\nabla \tilde{g}_{ki}(\hat{y}_k^{p_j})^T(\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j}) \leq 0. \tag{20}$$

Using the KKT multipliers in (16) yields

$$\sum_{i \in \mathcal{A}_k} \mu_i \nabla \tilde{g}_{ki}(\hat{y}_k^{p_j})^T(\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j}) = (\hat{x}_k^{p_j} - \hat{y}_k^{p_j})^T(\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j}) \leq 0. \tag{21}$$

Since $||\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j}||^2 \geq 0$ and $(\hat{x}_k^{p_j} - \hat{y}_k^{p_j})^T(\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j}) \leq 0$, (18) implies

$$||\hat{x}_k^{p_j} - \hat{x}_k^{p_{j+1}}||^2 \geq ||\hat{x}_k^{p_j} - \hat{y}_k^{p_j}||^2. \tag{22}$$

By Lemma 3 sequence $\{\hat{x}_k^{p_i}\}_{i=1}^{\infty}$ is convergent, *i.e.* $\lim_{j\to\infty} \hat{x}_k^{p_j} = z_k$, we have that $\lim_{j\to\infty} ||\hat{x}_k^{p_j} - \hat{x}_k^{p_{j+1}}|| = 0$. This means that $\lim_{j\to\infty} ||\hat{y}_k^{p_j} - \hat{x}_k^{p_j}|| = 0$. Then we have that $\lim_{j\to\infty} ||z_k - \hat{y}_k^{p_j}||^2 = 0$. This implies $\lim_{j\to\infty} \hat{y}_k^{p_j} = z_k$. Since the sequence $\{\hat{y}^{p_j}\}_{j=1}^{\infty} \in G$ and the sequence $\{\hat{x}^{p_j}\}_{j=1}^{\infty} \in P \cap Y$, and these sequences have common limit point z, then point z is feasible, *i.e.* $z \in P \cap X$. □

Lemma 5 *The limit point of a convergent subsequence is a global minimum point of* (1).

Proof We adapt the proof of [20]. Because each set $\widehat{X}^p$ is an outer approximation of the feasible set X, $c^T\hat{x}^{p_i}$ gives a lower bound on the optimal value of the objective function. Due to property (15), sequence $\{c^T\hat{x}^{p_i}\}_{i=1}^{\infty}$ is nondecreasing and since the objective function is continuous, we get $\lim_{i\to\infty} c^T\hat{x}^{p_i} = c^T z$. According to Lemma 4, limit point z is within the feasible set $P \cap X$ and, because it is a minimizer of the objective function within a set including the entire feasible set, it is also an optimal solution to (1). □

Since Lemmas 4 and 5 apply to all convergent subsequences generated by solving OA master problems (5), any limit point of such sequence will be a global optimum. We summarize the convergence results in the next theorem.

Theorem 1 *Algorithm* 1 *either finds a global optimum of* (1) *in a finite number of iterations or generates a sequence* $\{\hat{x}^{p_i}\}_{i=1}^{\infty}$ *converging to a global optimum.*

Proof Suppose the algorithm stops in a finite number of iterations. Then the last solution of OA master problem (5) satisfies all constraints and according to Lemma 1 it is a global optimum of (1). In case the algorithm does not stop in a finite number of iterations, it generates a sequence converging to a global optimum of (1) according to Lemmas 3 and 5. □

In Theorem 2 we prove that improved DECOA described in Algorithm 3 also converges to a global optimum of (1).

Theorem 2 *DECOA described in Algorithm 3 either finds a global optimum of (1) in a finite number of iterations or generates a sequence* $\{\hat{x}^{p_i}\}_{i=1}^{\infty}$ *converging to a global optimum.*

Proof The core idea of improved DECOA, described in Algorithm 3, is the same as in basic DECOA described in Algorithm 1. In the Algorithm 3 we introduce enhancements such as LP-OA master problem, and line search sub-problems for speeding up the convergence of Algorithm 1. Hence improved Algorithm 3 refines outer approximation $\widehat{X}$ faster, because in each iteration the additional methods make the outer approximation $\widehat{X}$ smaller. Moreover, all conditions assumed in the proof of Theorem 1 remain valid. Therefore, the proof is similar to the proof of Theorem 1. □

5 Implementation of DECOA

Algorithm 3 was implemented with Pyomo [17], an algebraic modelling language in Python, as part of the parallel MINLP-solver Decogo [29]. The implementation of Decogo is not finished, in particular parallel solving of sub-problems has not been implemented yet. The solver utilizes SCIP 5.0 [16] for solving MIP problems and IPOPT 3.12.8 [35] for solving LP and NLP problems. Note that it is possible to use other suitable solvers which can interface with Pyomo.

Very often problems are not given in a block-separable form. Therefore, a block structure identification of the original problem and its automatic reformulation into a block-separable form have been implemented. The block structure identification is based on the idea of connected components of a Hessian adjacency graph.

Consider a MINLP problem defined by n variables and by $|M|$ functions $h_m, m \in M$. Consider a Hessian adjacency graph $\mathcal{G} = (V, E)$ defined by the following vertex and edge sets

$$\begin{aligned} V &= \{1, \dots, n\}, \\ E &= \{(i, j) \in V \times V : \frac{\partial^2 h_m}{\partial x_i \partial x_j} \neq 0,\ m \in M\}. \end{aligned} \tag{23}$$

In order to subdivide the set of variables into $|K|$ blocks, we compute the connected components $V_k, k \in K$, of $\mathcal{G}$ with $\bigcup_{k \in K} V_k = V$. We obtain the list of variables $V_k \subset V,\ k \in K$, such that $n = \sum_{k \in K} n_k$, where $n_k = |V_k|$.

In the implementation, we don't compute the Hessian of functions h_m. Instead, we iterate over the (nonlinear) expressions of functions h_m. If two variables x_i and x_j are contained in the same nonlinear expression, we insert the edge (i, j) to the edge set E of $\mathcal{G}$.

Using the blocks $V_k, k \in K$, which correspond to the connected components of graph $\mathcal{G}$, we reformulate the original problem into a the block-separable MINLP problem described in (1). We perform this procedure by adding new variables and constraints such that the objective function and the global constraints are linear. Note that the reformulated problem remains convex.

As mentioned in Sect. 3, we add the supporting hyperplanes for each active constraints at point $\hat{y} \in T_k$ according to the formula

$$g_{kj}(\hat{y}) + \nabla g_{kj}(\hat{y})^T (x - \hat{y}) \leq 0,\ \hat{y} \in T_k. \tag{24}$$

Theoretically, we have $g_{kj}(\hat{y}) = 0$. In practice, the value $g_{kj}(\hat{y})$ is often very small, but, because of the numerical accuracy, it might not be identical to zero. To guarantee that the linearization cuts are valid, in practice we consider the non-zero value of $g_{k_j}(\hat{y})$ in (24).

DECOA described in Algorithm 3 terminates based on the relative gap, *i.e.*

$$\frac{|\overline{v} - c^T \hat{x}|}{10^{-12} + |\overline{v}|} < \varepsilon, \tag{25}$$

where ε is a desired tolerance. In addition to it, the loops in the LP phase, described in Algorithm 2, are terminated if there is no improvement of the objective function value, *i.e.* $c^T(\hat{x}^{p+1} - \hat{x}^p) < \delta$, where δ is a desired tolerance.

6 Numerical results

DECOA described in Algorithm 3 has been tested on convex MINLP problems from MINLPLib [33]. Some instances don't have a reasonable block structure, *i.e.* the number of blocks might be equal to the number of variables or the instance might have only one block. In order to avoid this issue and to show the potential of decomposition, we filtered all convex instances from MINLPLib using the following criterion:

$$1 < |K| < N, \tag{26}$$

where $|K|$ is the number of blocks and N the total number of variables. In the MINLPlib the number of blocks is given by identifier *#Blocks in Hessian of Lagrangian*, which is available for each problem. The number of selected instances is 70 and the number of variables varies from 11 to 2720 with an average value 613. In Table 1 we provide more detailed statistics on this set of instances.

As termination criteria, the relative gap tolerance was set to 0.0001 and the LP phase improvement tolerance was set to 0.01. The master problem and sub-problems were solved to optimality. All computational experiments were performed using a computer with Intel Core i7-7820HQ 2.9 GHz CPU and 16 GB RAM.

6.1 Effect of line search and fix-and-refine

In order to understand the impact of the line search and the fix-and-refine procedure, described in Algorithm 4, we run four variants of Algorithm 3:

i Only projection, *i.e.* line search and fix-and-refine were not performed;
ii Projection with fix-and-refine, *i.e.* line search was not performed;
iii Projection with line search, *i.e.* fix-and-refine was not performed;
iv Projection with line search and with fix-and-refine.

For each run, we computed the average number of MIP-solver runs and the average time spent on solving LP-OA master problems (10), for MIP-OA master problems (8), and for all sub-problems. Note that the sub-problem solution time includes the time spent on solving projection (9), line search (11) and partly-fixed OA (14) sub-problems. Note that, the NLP time is not presented. Since DECOA can be well parallelized, *i.e.* all sub-problems can be solved in parallel, we computed an estimated parallelized sub-problem time. The estimated parallelized sub-problem time is computed by taking the maximum time needed to solve the sub-problems in each parallel step. This value might be too low, since it assumes that the number of cores is equal to the number of blocks and it does not take the time needed for communication overhead into account. Nevertheless, this number gives a good estimate of possible time improvement.

Figure 1 shows that for most instances, the number of MIP runs remains the same regardless of the problem size. Moreover, for big problems, the algorithm needs not more than 2 MIP runs in order to close the gap, and this property is valid for all variants of the algorithm. The same behavior can also be observed in Fig. 2. It shows that most of the problems were solved with no more than 3 MIP runs regardless of the algorithm variant. This plot shows that the lowest average value of MIP runs can be obtained by running the algorithm with the fixed-and-refine procedure. Moreover, the fix-and-refine procedure helps to solve some problems with fewer MIP runs. However, running the algorithm with fix-and-refine is computationally demanding. This issue is illustrated in Fig. 3, which shows that the sub-problem time for the

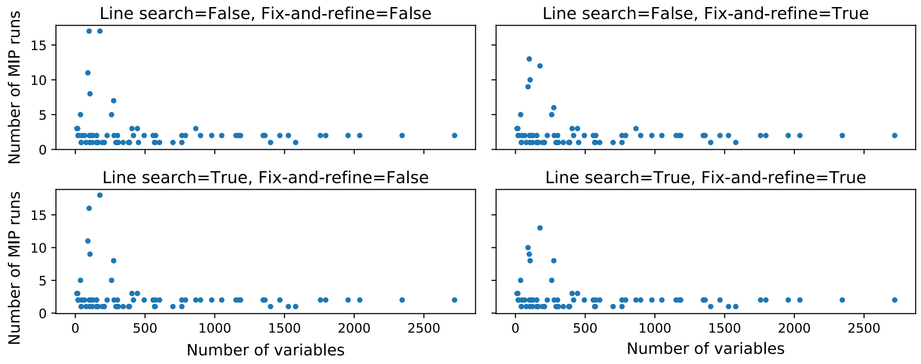

Fig. 1 Number of MIP runs is independent of the problem size

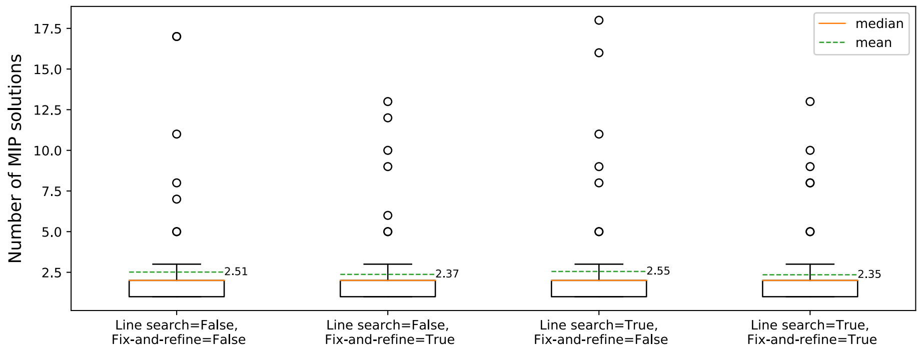

Fig. 2 The distribution of the number of MIP runs for four variants of Algorithm 3

algorithm with fix-and-refine is the highest. Moreover, this chart shows that, for each variant, the algorithm spends most of its time on solving sub-problems. In order to see the potential of parallelization, we computed the estimated parallelized sub-problem time. The computed estimate gives results lower than the LP time or MIP time.

From Fig. 3 one can notice that the average time spent on solving LP-OA master problems and MIP-OA master problems is approximately equal. Due to this observation and the fact that the LP problems are easier to solve than MIP problems, the LP-OA master problems were solved on average more times than MIP-OA master problems. Solving more LP-OA master problems at the beginning helps to initialize a good outer approximation and, therefore, to reduce the number of MIP runs. Similar gains in reduction of MIP runs have been achieved in [25]. In contrary to DECOA, in [25] has been proposed to improve the quality of polyhedral OA with extended formulations, which are based on convexity detection of the constraints.

6.2 Comparison to other MINLP solvers

In this subsection we compare the DECOA algorithm with two MINLP solvers which do not make use of the decomposition structure of the problems. For this purpose, we have chosen the branch-and-bound-based SCIP solver 5.0.1 [16] and Pyomo-based toolbox MindtPy 0.1.0 [2]. All settings for SCIP were set to default. In order to compare DECOA with OA, for MindtPy we set OA as a solution strategy with SCIP 5.0.1. and Ipopt 3.12.8 as a MIP solver

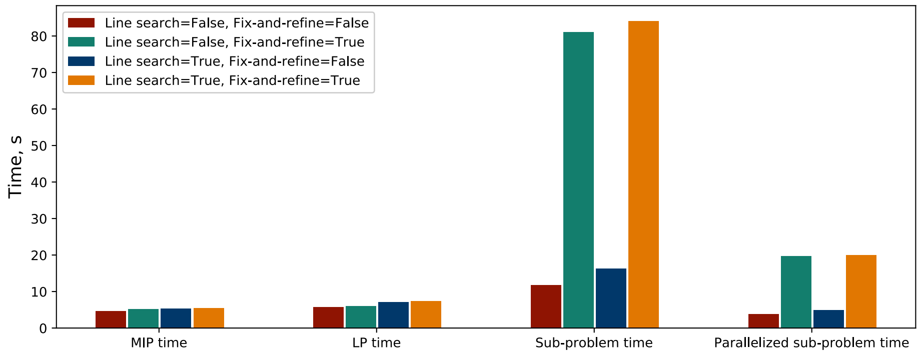

Fig. 3 The average time spent on solving master problems and sub-problems. MIP time corresponds to the time spent on solving MIP-OA master problems, LP time corresponds to the time spent on solving LP-OA master problems, and sub-problem time corresponds to the time spent on solving projection, line-search and partly-fixed OA sub-problems. Note that the NLP time is not presented. The parallelized sub-problem time is the maximum time needed to solve all sub-problems in parallel

and NLP solver, respectively. Moreover, the iteration limit for MindtPy was set to 100. All other settings for MindtPy were set to default.

For the comparisons with both solvers, we use the variant of Algorithm 3 without line-search and fix-and-refine. It is the least computationally demanding variant of Algorithm 3, as has been shown in Fig. 3. The test instances were selected from MINLPLib [33] using condition (26).

Table 1 presents the results for DECOA and SCIP for each instance individually. For each instance, it presents also its statistic, i.e. problem size N and average blocksize $\overline{N}_k$ after reformulation. For each instance, we measured the total solution time T of the DECOA run. Note that the total time T does not include time spent on automatic reformulation, described in Sect. 5. T_{MIP} denotes the time spent on solving MIP problems and N_{MIP} denotes the number of MIP runs. T_{LP} and T_{NLP} denote the time spent on solving LP and NLP problems respectively. T_{sub} denotes the time spent on solving sub-problems, *i.e.* the time spent on solving projection sub-problems (9). T_{SCIP} denotes the time spent on solving the original problem with SCIP.

In Table 1 we compare the solution time of SCIP and DECOA for each instance individually. However, comparing solution time of both solvers can't be realistic, since they are implemented using different programming languages, i.e. DECOA using Python and SCIP using C. It is known that Python is slower than C. One of the reasons for that, Python is interpreted language and C-compiled.

Table 1 shows that currently for 9 % of the test set, DECOA shows a shorter solution time than SCIP. Moreover, for 6 % of the test set, the solution time is very similar to SCIP, *i.e.* SCIP time is within 80 % of DECOA time. Moreover, for almost all problems, T_{MIP} is very small, and T_{sub} is relatively large. Hence, since all sub-problems can be solved in parallel, there is a clear indication that running time for DECOA can be significantly reduced, see Fig. 3.

From Table 1 one can conclude that T_{LP} is also high. Its average fraction of the total time T is 18%. It is followed by T_{MIP} and T_{NLP}, which have average fractions of the total time 12% and 7% respectively. As has been discussed before, even though the LP problems are easier to solve than MIP problems, the number of solved LP problems in the LP phase is higher than the number of solved MIP problems.

Table 1 Performance comparison per instance for variant of Algorithm 3 without line-search and fix-and-refine with the SCIP solver

	Instance name	N	$\overline{N}_k$	T (s)	T_{MIP} (s)	N_{MIP}	T_{LP} (s)	T_{NLP} (s)	T_{sub} (s)	T_{SCIP} (s)
1	batch	46	4.8	6.2	0.4	2	0.8	0.1	4.5	2.2
2	batch0812	100	5.7	11.7	0.5	2	2.2	0.4	7.9	1.8
3	batchdes	19	4.0	1.4	0.1	2	0.2	0.1	0.9	0.6
4	batchs101006m	278	10.2	31.9	3.4	2	8.6	0.8	17.4	19.5
5	batchs121208m	406	12.2	44.6	14.4	3	9.7	1.9	16.0	44.5
6	batchs151208m	445	12.4	63.5	22.8	3	16.8	2.3	18.6	56.6
7	batchs201210m	558	13.7	86.2	15.7	2	34.6	2.6	28.6	45.2
8	clay0203h	90	12.9	9.7	3.6	11	0.1	2.2	3.0	10.7
9	clay0204h	164	18.2	3.0	1.6	1	0.1	0.6	0.6	30.6
10	clay0205h	260	23.6	65.1	59.8	5	0.2	2.0	2.4	117.2
11	clay0303h	99	9.9	14.5	3.2	17	0.1	3.9	6.3	27.1
12	clay0304h	176	13.5	29.5	12.9	17	0.1	6.5	8.5	24.0
13	clay0305h	275	17.2	114.1	103.5	7	0.2	4.9	4.6	173.5
14	enpro48pb	153	11.9	9.4	1.9	2	1.7	0.2	5.1	6.5
15	enpro56pb	127	10.7	8.9	2.0	2	1.8	0.3	4.4	5.8
16	fac1	22	8.0	1.8	0.1	2	0.4	0.1	1.0	0.1
17	fac3	66	17.2	5.2	0.2	2	1.4	0.2	2.8	2.4
18	pollut	42	3.0	9.8	0.1	1	1.3	0.1	7.2	0.2
19	ravempb	112	7.5	8.2	1.7	2	1.1	0.3	4.6	8.0
20	rsyn0805h	308	77.0	4.4	0.1	1	1.4	0.7	1.9	1.2
21	rsyn0805m02h	700	100.0	13.6	0.9	1	4.4	1.0	6.0	3.7
22	rsyn0805m03h	1050	105.0	23.0	1.3	2	6.9	1.6	10.5	3.8
23	rsyn0805m04h	1400	107.7	40.6	0.5	1	17.1	3.5	14.9	6.8
24	rsyn0810h	343	49.0	4.7	0.1	1	1.3	1.0	1.8	1.4
25	rsyn0810m02h	790	60.8	22.5	2.8	2	5.6	1.5	9.6	5.9
26	rsyn0810m03h	1185	62.4	34.7	2.2	2	9.0	2.0	15.3	13.7
27	rsyn0810m04h	1580	63.2	49.5	0.8	1	17.8	4.0	18.1	6.4
28	rsyn0815h	387	35.2	7.7	0.2	1	2.0	1.1	3.2	1.9
29	rsyn0815m02h	898	42.8	28.7	1.7	2	6.3	2.4	12.8	3.9
30	rsyn0815m03h	1347	43.5	50.4	3.3	2	11.0	4.0	20.4	13.0
31	rsyn0815m04h	1796	43.8	67.4	1.9	2	16.0	4.5	27.0	4.6
32	rsyn0820h	417	29.8	11.3	0.4	2	3.0	0.6	5.4	4.4
33	rsyn0820m02h	978	36.2	32.2	1.8	2	5.9	2.6	14.9	6.5
34	rsyn0820m03h	1467	36.7	48.8	2.6	2	9.7	3.1	20.5	10.1
35	rsyn0820m04h	1956	36.9	83.8	2.2	2	19.9	6.4	31.7	12.9
36	rsyn0830h	494	24.7	16.9	0.7	2	3.5	1.1	8.0	8.7
37	rsyn0830m02h	1172	30.1	43.5	2.9	2	7.8	2.1	19.0	20.9
38	rsyn0830m03h	1758	30.3	74.8	3.0	2	14.9	3.1	30.1	16.8
39	rsyn0830m04h	2344	30.4	116.6	4.0	2	25.2	4.2	42.6	45.6
40	rsyn0840h	568	21.0	18.2	0.3	1	3.9	1.0	8.6	4.2

Table 1 continued

	Instance name	N	$\overline{N}_k$	T (s)	T_{MIP} (s)	N_{MIP}	T_{LP} (s)	T_{NLP} (s)	T_{sub} (s)	T_{SCIP} (s)
41	`rsyn0840m02h`	1360	25.7	51.3	1.6	2	8.5	2.1	22.2	9.4
42	`rsyn0840m03h`	2040	25.8	101.8	2.8	2	17.5	4.3	38.7	17.6
43	`rsyn0840m04h`	2720	25.9	160.3	4.5	2	29.6	5.5	54.8	46.3
44	`syn05h`	42	10.5	1.5	0.0	1	0.3	0.2	0.8	0.4
45	`syn05m02h`	104	14.9	3.5	0.1	1	0.7	0.2	1.9	0.6
46	`syn05m03h`	156	15.6	6.0	0.1	1	1.4	0.3	3.2	0.9
47	`syn05m04h`	208	16.0	7.7	0.1	1	1.6	0.4	4.4	0.9
48	`syn10h`	77	11.0	1.6	0.1	1	0.3	0.3	0.7	0.3
49	`syn10m02h`	194	14.9	6.8	0.1	1	1.4	0.4	3.8	1.4
50	`syn10m03h`	291	15.3	10.2	0.1	1	1.8	0.5	5.7	2.3
51	`syn10m04h`	388	15.5	14.0	0.2	1	2.6	0.7	7.4	2.3
52	`syn15h`	121	11.0	4.8	0.1	1	0.9	0.4	2.7	0.6
53	`syn15m02h`	302	14.4	11.0	0.1	1	1.7	0.5	6.1	1.4
54	`syn15m03h`	453	14.6	17.7	0.2	1	2.6	1.3	9.1	2.2
55	`syn15m04h`	604	14.7	23.5	0.2	1	3.4	0.8	12.2	3.0
56	`syn20h`	151	10.8	6.6	0.1	1	1.2	0.5	3.7	1.7
57	`syn20m02h`	382	14.1	12.9	0.2	1	1.9	0.6	7.1	3.5
58	`syn20m03h`	573	14.3	20.3	0.2	1	2.8	0.8	10.5	4.4
59	`syn20m04h`	764	14.4	32.3	0.2	1	4.5	1.1	15.8	4.0
60	`syn30h`	228	11.4	11.6	0.3	2	1.5	0.8	6.8	3.3
61	`syn30m02h`	576	14.8	27.4	0.6	2	3.8	0.7	14.7	6.7
62	`syn30m03h`	864	14.9	51.2	1.4	3	6.0	3.4	24.3	7.2
63	`syn30m04h`	1152	15.0	68.0	1.3	2	8.3	2.0	30.4	14.0
64	`syn40h`	302	11.2	16.2	0.5	2	2.0	1.0	9.0	3.2
65	`syn40m02h`	764	14.4	37.6	0.9	2	5.0	0.9	18.4	2.0
66	`syn40m04h`	1528	14.6	103.4	2.4	2	11.7	8.0	38.8	19.3
67	`synthes2`	11	4.0	1.6	0.2	3	0.3	0.1	1.0	2.0
68	`synthes3`	17	4.3	2.5	0.2	3	0.3	0.2	1.6	0.7
69	`tls2`	37	13.7	2.7	1.1	5	0.2	0.4	0.9	0.3
70	`tls4`	105	24.2	25.7	22.2	8	0.2	0.8	1.5	19.6

Table 2 presents the results for DECOA and OA for each instance individually. Both for DECOA and for OA, the number of MIP runs N_{MIP} and total time T are presented. Additionally for OA, the solver status after finishing the solution process is provided.

Table 2 shows the OA method failed to converge for 20% of the instances due to either iteration limit or solver exception. For some instances, MindtPy failed to close the gap due to infeasibility of NLP sub-problem, i.e. infeasible combination of values for integer variables. The results in Table 2 present that for almost all instances, the number of MIP runs N_{MIP} for DECOA is less than the number of MIP runs N_{MIP} for OA. However, the solution time T for DECOA is either bigger or smaller than the solution time T for OA depending on the number of MIP runs. If the number of MIP runs N_{MIP} for OA is big, i.e. $N_{MIP} > 10$, then for almost all instances, the solution time T for DECOA is smaller than the solution time T

Table 2 Performance comparison per instance for variant of Algorithm 3 without line-search and fix-and-refine with MindtPy using OA strategy

	Instance name	DECOA		OA		
		N_{MIP}	T(s)	N_{MIP}	T (s)	Status
1	`batch`	2	6.2	3	1.5	Converged
2	`batch0812`	2	11.7	–	–	Iterations limit
3	`batchdes`	2	1.4	2	0.3	Converged
4	`batchs101006m`	2	31.9	11	44.4	Converged
5	`batchs121208m`	3	44.6	5	37.9	Converged
6	`batchs151208m`	3	63.5	–	–	Iterations limit
7	`batchs201210m`	2	86.2	9	176.1	Converged
8	`clay0203h`	11	9.7	–	–	Iterations limit
9	`clay0204h`	1	3.0	17	45.5	Converged
10	`clay0205h`	5	65.1	–	–	Exception
11	`clay0303h`	17	14.5	5	6.2	Converged
12	`clay0304h`	17	29.5	–	–	Iterations limit
13	`clay0305h`	7	114.1	–	–	Exception
14	`enpro48pb`	2	9.4	3	4.4	Converged
15	`enpro56pb`	2	8.9	2	3.7	Converged
16	`fac1`	2	1.8	–	–	Exception
17	`fac3`	2	5.2	7	1.8	Converged
18	`pollut`	1	9.8	–	–	Exception
19	`ravempb`	2	8.2	–	–	Exception
20	`rsyn0805h`	1	4.4	2	1.2	Converged
21	`rsyn0805m02h`	1	13.6	6	12.2	Converged
22	`rsyn0805m03h`	2	23.0	12	23.5	Converged
23	`rsyn0805m04h`	1	40.6	–	–	Iterations limit
24	`rsyn0810h`	1	4.7	1	1.1	Converged
25	`rsyn0810m02h`	2	22.5	43	103.7	Converged
26	`rsyn0810m03h`	2	34.7	4	18.7	Converged
27	`rsyn0810m04h`	1	49.5	20	65.1	Converged
28	`rsyn0815h`	1	7.7	2	1.7	Converged
29	`rsyn0815m02h`	2	28.7	8	14.4	Converged
30	`rsyn0815m03h`	2	50.4	7	35.9	Converged
31	`rsyn0815m04h`	2	67.4	28	100.2	Converged
32	`rsyn0820h`	2	11.3	–	–	Iterations limit
33	`rsyn0820m02h`	2	32.2	8	15.9	Converged
34	`rsyn0820m03h`	2	48.8	5	26.9	Converged
35	`rsyn0820m04h`	2	83.8	10	44.1	Converged
36	`rsyn0830h`	2	16.9	5	5.7	Converged
37	`rsyn0830m02h`	2	43.5	–	–	Iterations limit
38	`rsyn0830m03h`	2	74.8	3	12.7	Converged
39	`rsyn0830m04h`	2	116.6	4	27.0	Converged
40	`rsyn0840h`	1	18.2	3	3.0	Converged

Table 2 continued

	Instance name	DECOA		OA		
		N_{MIP}	T(s)	N_{MIP}	T (s)	Status
41	`rsyn0840m02h`	2	51.3	4	12.4	Converged
42	`rsyn0840m03h`	2	101.8	4	21.3	Converged
43	`rsyn0840m04h`	2	160.3	15	100.6	Converged
44	`syn05h`	1	1.5	2	0.3	Converged
45	`syn05m02h`	1	3.5	2	0.5	Converged
46	`syn05m03h`	1	6.0	2	0.6	Converged
47	`syn05m04h`	1	7.7	2	0.7	Converged
48	`syn10h`	1	1.6	1	0.3	Converged
49	`syn10m02h`	1	6.8	2	0.7	Converged
50	`syn10m03h`	1	10.2	2	1.0	Converged
51	`syn10m04h`	1	14.0	2	1.3	Converged
52	`syn15h`	1	4.8	2	0.5	Converged
53	`syn15m02h`	1	11.0	2	1.0	Converged
54	`syn15m03h`	1	17.7	2	2.4	Converged
55	`syn15m04h`	1	23.5	2	2.0	Converged
56	`syn20h`	1	6.6	3	0.9	Converged
57	`syn20m02h`	1	12.9	3	1.8	Converged
58	`syn20m03h`	1	20.3	4	3.4	Converged
59	`syn20m04h`	1	32.3	2	3.4	Converged
60	`syn30h`	2	11.6	4	1.7	Converged
61	`syn30m02h`	2	27.4	4	3.4	Converged
62	`syn30m03h`	3	51.2	4	5.9	Converged
63	`syn30m04h`	2	68.0	4	8.0	Converged
64	`syn40h`	2	16.2	5	2.9	Converged
65	`syn40m02h`	2	37.6	3	4.2	Converged
66	`syn40m04h`	2	103.4	5	15.6	Converged
67	`synthes2`	3	1.6	4	0.4	Converged
68	`synthes3`	3	2.5	7	0.9	Converged
69	`tls2`	5	2.7	–	–	Exception
70	`tls4`	8	25.7	–	–	Exception

of OA, i.e. DECOA is more efficient than OA for these problems. This situation is illustrated very well with instance `clay0204h`. For instances with a small number of MIP runs N_{MIP} for OA, i.e. $N_{MIP} < 10$, the solution time T for OA is smaller than the solution time T for DECOA.

7 Conclusions and future work

This paper introduces a new decomposition-based outer approximation (DECOA) algorithm for solving convex block-separable MINLP problems described in (1). It iteratively solves and refines an outer approximation (OA) problem by generating new supporting hyperplanes. Due to the block-separability of the problem (1), the sample points for supporting hyperplanes are obtained by solving low-dimensional sub-problems. Moreover, the sub-problems can be solved in parallel. The algorithm is designed such that the MIP-OA master problems are solved as few times as possible, since solving them might be computationally demanding.

Four variants of DECOA have been tested on a set of convex MINLP instances. The experiments have shown that for each case, the average number of MIP runs is small. Moreover, the results show that the average number of MIP runs is independent of the problem size. In addition to this, the time spent on solving sub-problems is bigger than time to solve LP and MIP problems.

The performance of DECOA has been compared to the branch-and-bound MINLP solver SCIP and to the OA method. Even though DECOA is based on a Python implementation, it can even be faster for some (9%) of the instances than an advanced implementation like SCIP. Probably this is due to the effect of the decomposition and the fact that it requires less MIP runs. Comparison to OA shows that DECOA reduces the number of MIP runs and it is more efficient in cases when the problem needs to be solved with a high number of MIP runs.

Even though DECOA is clearly defined and proven to converge, there are possibilities to improve its efficiency. It is possible to obtain a couple of solutions from the MIP solver and project them onto the feasible set. This could increase the number of new supporting hyperplanes in one iteration. Unfortunately, Pyomo does not facilitate working with a set of MIP solution candidates. The numerical results show that the time for solving MIP master problems is small, and reducing the time for solving LP master problems and sub-problems would significantly improve the performance of DECOA. Therefore, it would be interesting to work on reducing the number of iterations during the LP phase, and on faster solving the projection sub-problems (9). Also the current implementation could be improved, *i.e.* by implementing the parallelization, which could reduce the running time of DECOA significantly. The possible advantage of DECOA over branch-and-bound solvers would be with large-scale problems, which cannot be solved in reasonable time by branch-and-bound. However, this has to be verified by systematic experiments. In the future, we aim to generalize DECOA for solving nonconvex MINLP problems.

Acknowledgements Open Access funding provided by Projekt DEAL. This work has been funded by Grant RTI2018-095993 from the Spanish Ministry in part financed by the European Regional Development Fund (ERDF) and by Grant 03ET4053B of the German Federal Ministry for Economic Affairs and Energy. We are grateful to the anonymous referees for their encouragement and many remarks.

References

1. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optim. Methods Softw. **24**(4–5), 597–634 (2009)

2. Bernal, D.E., Chen, Q., Gong, F., Grossmann, I.E.: Mixed-integer nonlinear decomposition toolbox for Pyomo (MindtPy). In: 13th International Symposium on Process Systems Engineering (PSE 2018). Elsevier, Amsterdam (2018)
3. Borndörfer, R., Löbel, A., Reuther, M., Schlechte, T., Weider, S.: Rapid branching. Public Transp. **5**, 3–23 (2013)
4. Burer, S., Letchford, A.: Non-convex mixed-integer nonlinear programming: a survey. Surv. Oper. Res. Manag. Sci. **17**(2), 97–106 (2012)
5. Burlacu, R., Geißler, B., Schewe, L.: Solving mixed-integer nonlinear programmes using adaptively refined mixed-integer linear programmes. Optim. Methods. Softw. **35**(1), 37–64 (2020)
6. Bussieck, M.R., Vigerske, S.: MINLP Solver Software. http://www.math.hu-berlin.de/~stefan/minlpsoft.pdf, (2014)
7. Dakin, R.J.: A tree-search algorithm for mixed integer programming problems. Comput. J. **8**(3), 250–255 (1965)
8. D'Ambrosio, C., Frangioni, A., Liberti, L., Lodi, A.: A storm of feasibility pumps for nonconvex MINLP. Math. Program. **136**(2), 375–402 (2012)
9. Desrosiers, J., Lübbecke, M.: Branch-price-and-cut algorithms. In: Cochran, J., Cox, L., Keskinocak, P., Kharoufeh, J., Smith, J. (eds.) Wiley Encyclopedia of Operations Research and Management Science. Wiley, London (2010)
10. Duran, M., Grossmann, I.: An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Math. Program. **36**, 307–339 (1986)
11. Feltenmark, S., Kiwiel, K.C.: Dual applications of proximal bundle methods including Lagrangian relaxation of nonconvex problems. SIAM J. Optim. **10**(3), 697–721 (2000)
12. Fletcher, R., Leyffer, S.: Solving mixed integer nonlinear programs by outer approximation. Math. Program. **66**(3(A)), 327–349 (1994)
13. Flippo, O.E., Rinnooy-Kan, A.H.G.: Decomposition in general mathematical programming. Math. Program. **60**, 361–382 (1993)
14. Geoffrion, A.M.: Lagrangian relaxation for integer programming. Math. Program. Stud. **2**, 82–114 (1974)
15. Geoffrion, A.M.: Generalized Benders decomposition. J. Optim. Theory Appl. **10**(4), 237–260 (1972)
16. Gleixner, A., Eifler, L., Gally, T., Gamrath, G., Gemander, P., Gottwald, R.L., Hendel, G., Hojny, C., Koch, T., Miltenberger, M., Müller, B., Pfetsch, M.E., Puchert, C., Rehfeldt, D., Schlösser, F., Serrano, F., Shinano, Y., Viernickel, J.M., Vigerske, S., Weninger, D., Witt, J.T., Witzig, J.: The SCIP Optimization Suite 5.0. Technical report, http://www.optimization-online.org/DB_HTML/2017/12/6385.html (2017)
17. Hart, W.E., Laird, C.D., Watson, J.-P., Woodruff, D.L., Hackebeil, G.A., Nicholson, B.L., Siirola, J.D.: Pyomo-optimization Modeling in Python, second edition, vol. 67. Springer, Berlin (2017)
18. Hunting, M.: The AIMMS outer approximation algorithm for MINLP. AIMMS B.V, Technical report (2011)
19. Kronqvist, J., Bernal, D.E., Lundell, A., Grossmann, I.E.: A review and comparison of solvers for convex MINLP. Optim. Eng. **20**(2), 397–455 (2018)
20. Kronqvist, J., Lundell, A., Westerlund, T.: The extended supporting hyperplane algorithm for convex mixed-integer nonlinear programming. J. Glob. Optim. **64**(2), 249–272 (2016)
21. Lemaréchal, C., Renaud, A.: A geometric study of duality gaps, with applications. Math. Program. **90**, 399–427 (2001)
22. Leyffer, S., Sartenaer, A., Wanufelle, E.: Branch-and-Refine for Mixed Integer Nonconvex Global Optimization. Technical report, Preprint ANL/MCS-P1547-0908,Mathematics and Computer Science Division, Argonne National Laboratory (2008)
23. Lin, Y., Schrage, L.: The global solver in the LINDO API. Optim. Methods Softw. **24**(4–5), 657–668 (2009)
24. Lübbecke, M.E., Desrosiers, J.: Selected topics in column generation. Oper. Res. **53**(6), 1007–1023 (2005)
25. Lubin, M., Yamangil, E., Bent, R., Vielma, J.P.: Polyhedral approximation in mixed-integer convex optimization. Math. Program. **172**(1), 139–168 (2018)
26. Misener, R., Floudas, C.: ANTIGONE: algorithms for coNTinuous/integer global optimization of nonlinear equations. J. Glob. Optim. **59**(2–3), 503–526 (2014)
27. Nowak, I.: Relaxation and Decomposition Methods for Mixed Integer Nonlinear Programming. Birkhäuser, Basel (2005)
28. Nowak, I.: Parallel decomposition methods for nonconvex optimization-recent advances and new directions. In: Proceedings of MAGO (2014)
29. Nowak, I., Breitfeld, N., Hendrix, E.M.T., Njacheun-Njanzoua, G.: Decomposition-based Inner- and outer-refinement algorithms for global optimization. J. Glob. Optim. **72**(2), 305–321 (2018)
30. Ralphs, T., Galati, M.: Decomposition and dynamic cut generation in integer linear programming. Math. Program. **106**(2), 261–285 (2006)

31. Tawarmalani, M., Sahinidis, N.: A polyhedral branch-and-cut approach to global optimization. Math. Program. **103**(2), 225–249 (2005)
32. Vigerske, S.: Decomposition in Multistage Stochastic Programming and a Constraint Integer Programming Approach to Mixed-Integer Nonlinear Programming. Ph.D. thesis, Humboldt-Universität zu Berlin (2012)
33. Vigerske, S.: MINLPLib. http://minlplib.org/index.html (2018)
34. Wächter, A.: An interior point algorithm for large-scale nonlinear optimization with applications in process engineering. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, USA, http://researcher.watson.ibm.com/researcher/files/us-andreasw/thesis.pdf (2002)
35. Wächter, A., Lorenz, B.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program. **106**(1), 25–57 (2006)
36. Westerlund, T., Petterson, F.: An extended cutting plane method for solving convex MINLP problems. Comput. Chem. Eng. **21**, 131–136 (1995)