



UNIVERSIDAD
DE MÁLAGA



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Universidad de Málaga
Departamento de Arquitectura de Computadores

TESIS DOCTORAL

DECOMPOSITION METHODS FOR MIXED-INTEGER NONLINEAR PROGRAMMING

En cooperación con HAW Hamburg, Alemania


Autor:	M.Sc. Pavlo Muts
Directores:	Dr. Eligius M.T. Hendrix & Dr. Ivo Nowak
Programa de Doctorado:	Tecnologías Informáticas
Centro:	E.T.S. de Ingeniería Informática

Málaga, Junio 2021



UNIVERSIDAD
DE MÁLAGA

AUTOR: Pavlo Muts

 <https://orcid.org/0000-0002-0665-9629>

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga



Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional:

<http://creativecommons.org/licenses/by-nc-nd/4.0/legalcode>

Cualquier parte de esta obra se puede reproducir sin autorización
pero con el reconocimiento y atribución de los autores.

No se puede hacer uso comercial de la obra y no se puede alterar, transformar o hacer obras derivadas.

Esta Tesis Doctoral está depositada en el Repositorio Institucional de la Universidad de Málaga (RIUMA): riuma.uma.es



UNIVERSIDAD
DE MÁLAGA



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Universidad de Málaga
Departamento de Arquitectura de Computadores

Ph.D. Thesis

Decomposition methods for mixed-integer nonlinear programming

In cooperation with HAW Hamburg, Germany

Author: M.Sc. Pavlo Muts
Supervisors: Dr. Eligius M.T. Hendrix
Dr. Ivo Nowak

Málaga, June 2021

Preface

Mixed-integer nonlinear programming (MINLP) is an important and challenging field of optimization. The problems from this class can contain continuous and integral variables as well as linear and nonlinear constraints. This class of problems has a pivotal role in science and industry, since it provides an accurate way to describe phenomena in different areas like chemical and mechanical engineering, supply chain, management, etc. Most of the state-of-the-art algorithms for solving nonconvex MINLP problems are based on branch-and-bound. The main drawback of this approach is that the search tree may grow very fast preventing the algorithm to find a high-quality solution in a reasonable time. An alternative to avoid generating one big search tree is to make use of decomposition to make the solution process more tractable. Decomposition provides a general framework where one splits the original problem into smaller sub-problems and combines their solutions into an easier global master problem.

This thesis deals with decomposition methods for mixed-integer nonlinear programming. The main objective of this thesis is to develop alternative approaches to branch-and-bound based on decomposition-based successive approximation methods. For industry and science, it is important to compute an optimal solution, or at least, improve the best available so far. Moreover, this should be done within a reasonable time. Therefore, the goal is to design efficient algorithms to solve large-scale problems that have a direct practical application. In particular, we focus on models that have an application in energy system planning and operation. In this thesis, two main research lines can be distinguished. The first deals with Outer Approximation methods while the second studies a Column Generation approach. We investigate and analyse theoretical and practical aspects of both ideas within a decomposition framework. The main purpose of this study is to develop systematic decomposition-based successive approximation approaches to solve large-scale problems using Outer Approximation and Column Generation. Chapter 1 introduces an important concept needed for

decomposition, i.e. a block-separable reformulation of a MINLP problem. In addition, it describes the above-mentioned methods, including branch-and-bound, and several other key concepts needed for this thesis, e.g. Inner Approximation, etc.

Chapters 2, 3 and 4 investigate the use of Outer Approximation. Chapter 2 presents a decomposition-based Outer Approximation algorithm for solving convex MINLP problems based on construction of supporting hyperplanes for a feasible set. Chapter 3 extends decomposition-based Outer Approximation algorithm to nonconvex MINLP problems by introducing a piecewise nonconvex Outer Approximation of a nonconvex feasible set. Another perspective of the Outer Approximation definition for nonconvex problems is considered in Chapter 4. It presents a decomposition-based Inner and Outer Refinement algorithm, which constructs an Outer Approximation while computing the Inner Approximation using Column Generation. The Outer Approximation used in the Inner and Outer Refinement algorithm is based on the multi-objective view of the so-called resource-constrained version of the original problem.

Two chapters are devoted to Column Generation. Chapter 4 presents a Column Generation algorithm to compute an Inner Approximation of the original problem. Moreover, it describes a partition-based heuristic algorithm which uses an Inner Approximation refinement. Chapter 5 discusses several acceleration techniques for Column Generation. Furthermore, it presents a Column Generation-based heuristic algorithm that can be applied to any MINLP problem. The algorithm utilizes a projection-based primal heuristic to generate several high-quality solution candidates.

Chapter 6 contains a short description of the implementation in Python of the MINLP solver DECOGO. Chapter 7 summarizes the findings obtained during the elaboration of this thesis.

Acknowledgements

In the first place, I would like to express my sincere gratitude to my supervisors Eligius M.T. Hendrix and Ivo Nowak who gave me an opportunity to dive into very interesting topics of optimization. Without their trust, effort, patience and very good guidance from the beginning, this thesis would not have been possible. I would like to thank my tutor Inmaculada García for her advice and experience. Moreover, I am very grateful to her for helping me with the Spanish translation of the part of this thesis. I am very grateful to the reviewers Jan Kronqvist and Frédéric Messine for their time and effort to provide many very good comments which improved the thesis significantly.

I am grateful to every member of the Heinrich Blasius Institute at HAW Hamburg for creating a comfortable work environment and organizing nice social activities. I would like to thank Ouyang Wu for an interesting collaboration during the last year.

I am grateful for having a reliable source of funding during my entire time as a PhD student by the Grant 03ET4053B of the German Federal Ministry of Economic Affairs and Energy within the joint project together with the Institute of Energy Engineering and Environmental Protection of TU Berlin. I would like to thank to all collaborators from TU Berlin. In particular, I am grateful to Stefan Bruche for providing great test instances in energy systems planning and operation and writing a paper together.

During these years, I have had the opportunity to visit numerous scientific conferences around Europe. There, I met many nice and kind people who were happy to share their scientific experience and to provide a help during this project. In particular, I am grateful to Stefan Vigerske for converting entire MINLPLib into Pyomo format and to Jan Kronqvist and Andreas Lundell for sharing their knowledge and experience about Outer Approximation.

Last but not least, I wish to express my gratitude to my family and my friends for their support while writing this thesis. In particular, I would like to thank my lovely wife Kateryna for being always supportive. I am grateful to her for finding the time to listen to me and helping me with different aspects of writing the thesis. I would have never finished it without her love and kindness.

Contents

Preface	i
Acknowledgements	iii
List of Figures	ix
List of Tables	xi
List of Algorithms	xiv
Acronyms	xv
1 Introduction	1
1.1 Block-separable formulation of MINLP	1
1.2 MINLPLib – a collection of MINLP instances	3
1.3 Natural block structure identification	4
1.4 Branch-and-bound	5
1.5 Outer Approximation	9
1.6 Resource-constrained reformulation	12
1.6.1 Definition of the resource-constrained program	13
1.6.2 Multi-objective perspective	14
1.6.3 Dimension reduction of the resources	15
1.6.4 Supported nondominated points	16
1.7 Inner Approximation	18
1.8 Multi- and single-tree methods	20

1.9	Research questions	21
2	A Decomposition-based Outer Approximation Algorithm for convex MINLP	23
2.1	Introduction	23
2.2	DECOA	25
2.2.1	OA master problem	25
2.2.2	Basic DECOA	26
2.2.3	The LP phase	27
2.2.4	The MIP phase	30
2.3	Proof of convergence	32
2.4	Numerical results	36
2.4.1	Effect of line-search and fix-and-refine	37
2.4.2	Comparison to other MINLP solvers	39
2.5	Conclusions	46
3	A Decomposition-based Outer Approximation Algorithm for nonconvex MINLP	49
3.1	Introduction	49
3.2	Piecewise DC Outer Approximation	50
3.3	OA initialization	53
3.4	The local search	56
3.5	The main algorithm	57
3.6	Numerical illustration	59
3.7	Conclusions	60
4	A Decomposition-based Inner and Outer Refinement Algorithm for nonconvex MINLP	63
4.1	Introduction	63
4.2	Column Generation	64
4.2.1	Initialization of LP-IA	65
4.2.2	A Column Generation algorithm	66
4.3	A DIOR algorithm for computing a MIP outer approximation	67
4.3.1	An LP outer approximation	68
4.3.2	A MIP outer approximation	68
4.3.3	Disjunctive cuts	69
4.3.4	Pareto line-search	69

4.3.5	DIOR using Pareto line-search	70
4.3.6	Proof of convergence	71
4.4	A DIOR algorithm for computing a MIP inner approximation	75
4.4.1	MIP inner approximation	75
4.4.2	Refinement of MIP-IA	76
4.4.3	DIOR using a MIP inner approximation	82
4.5	Numerical results	83
4.5.1	Experiment with Algorithm 4.4 (DIOR1)	83
4.5.2	Experiments with Algorithm 4.9 (DIOR2)	84
4.6	Conclusions	87
5	A heuristic Column Generation Algorithm for solving energy system planning problems	89
5.1	Introduction	89
5.2	Traditional Column Generation	90
5.2.1	Handling the linear block (sub-problem)	90
5.2.2	Column Generation using MINLP sub-problems	91
5.2.3	Initialization of the column set	92
5.3	Acceleration of Column Generation	93
5.3.1	Fast Column Generation using NLP local search and rounding	93
5.3.2	CG using a Frank-Wolfe algorithm	94
5.4	A heuristic algorithm for finding solution candidates	97
5.5	Main algorithm	99
5.6	Convergence analysis	100
5.6.1	Convergence of Column Generation (Algorithm 5.9)	100
5.6.2	Convergence of the Frank-Wolfe algorithm (Algorithm 5.6)	101
5.7	Numerical results	101
5.7.1	DESSLib model instances	102
5.7.2	Effect of linear block integration into LP-IA and fast CG with the Frank-Wolfe approach	103
5.7.3	Impact of using the solution pool in Algorithm 5.8	105
5.7.4	Comparison to other approaches	106
5.8	Conclusions	110
6	The implementation of the DECOGO solver	111
6.1	Motivation	111
6.2	Structure and classes	112

6.2.1	Model	112
6.2.2	Problem	113
6.2.3	Solver	115
6.2.4	Utility	116
7	Conclusions	117
7.1	Outer Approximation	117
7.2	Column Generation	118
A	Publications arising from this thesis	121
A.1	Journal publications	121
A.2	Submitted journal publication	121
A.3	Publications in international conference proceedings	122
B	Other publications produced during the elaboration of this thesis	123
B.1	Book chapter	123
B.2	Publication in international conference proceedings	123
	Resumen en español	125
	Bibliography	141

List of Figures

- 1.1 Example of block-separable problem structure defined by (1.2) for the case $|K| = 3$. The black edges represent set P defined by coupling constraints. The blue dots show variables x_k and the blue edges represent set X_k defined by local constraints. 3
- 1.2 Resulting branch-and-bound tree for Example 1.1. Indices at the values for lower and upper bounds are omitted. 9
- 1.3 Outer approximation of nonlinear feasible set G defined by convex nonlinear constraint functions. The OA is defined by the set of hyperplanes and is represented by the blue dots. 10
- 1.4 Comparison of bilinear function $z = xy$ and corresponding McCormick underestimators on interval $x \in [-1, 1], y \in [-1, 1]$ 12
- 1.5 Resource constraint space of blocks $k = 1, 2$. Image spaces W_1 and W_2 in blue with extreme points as circles. Pareto front in black, with extreme points as a star. Supported Pareto points are marked with a green square. As a red square, the ideal \underline{w}_k (left-under) and the nadir point \bar{w}_k (right-up). 17
- 1.6 Inner approximation of set X denoted by $\text{conv}(S)$ is illustrated by the interior of the blue polygon. The grey shaded area illustrates nonlinear feasible set G . The black lines depict the mixed-integer nonlinear feasible set X . The blue points represent set of feasible points $S \subset X$. The orange points illustrate infeasible region inside of $\text{conv}(S)$ 19

LIST OF FIGURES

2.1	Projection of point \hat{x} onto set G defined by (2.5). The blue dots represent outer approximation \hat{G} of nonlinear feasible set G . The green line illustrates linear global constraints P	27
2.2	The line-search procedure between interior point \check{x} and OA point \hat{x} . The blue dots illustrate outer approximation \hat{G} of nonlinear feasible set G . The green line illustrates linear global constraints P	29
2.3	Number of MIP runs with respect to the instance size.	38
2.4	The distribution of the number of MIP runs for four variants of Algorithm 2.3 over selected instances.	39
2.5	The average time spent to solve master problems and sub-problems. . .	40
3.1	Comparison of original function $g(x)$, convexified function $h(x)$ and polyhedral DC underestimator $\check{g}(x)$ for function $g(x) = 3/(4x + 6) - x^2$. . .	52
3.2	Adaptive partition of interval $[p_\ell, p_{\ell+1}]$, defined by (3.15).	53
4.1	Steps 5-10 of Algorithm 4.4 for Example 1.4. The blue arrows represent a line-search towards the feasible set defined by OA solution \hat{w} and ideal point $\underline{w}_1 = (-8, 0)$ and $\underline{w}_2 = (-6.2, 5)$. The grey shaded area represents eliminated cones.	84
5.1	Convergence of the IA objective of Algorithm 5.9 with and without linear block integration for instance S16L16-1.	104
5.2	Convergence of the IA objective value of Algorithm 5.9 for S4L4-1 with and without the fast FW Column Generation.	105
5.3	Solution pool for S16L16-1.	106
5.4	Algorithm 5.9 computing time versus problem size of all instances in Table 5.2.	109
6.1	Class inheritance diagram of the master problems.	114
6.2	Class inheritance diagram of the sub-problems.	115

List of Tables

2.1	Performance comparison per instance for variant of Algorithm 2.3 without line-search and fix-and-refine with the SCIP solver	40
2.2	Performance comparison per instance for the variant of Algorithm 2.3 without line-search and fix-and-refine with MindtPy using OA strategy.	43
4.1	Performance of CG Algorithm 4.2, CG.	85
4.2	Performance of Algorithm 4.9, DIOR2.	86
4.3	Comparing Algorithm 4.9 with the SCIP solver. All values in seconds.	87
5.1	Solution quality comparison of Algorithm 5.9 solution ν_{CG} to the primal BARON solution ν_B and best known solution ν^* . Note that negative value means that the primal bound has been improved. All values are given as percentage.	107
5.2	Characteristics of selected test instances and performance comparison of Column Generation Algorithm 5.9 and BARON.	108

List of Algorithms

1.1	Generic branch-and-bound algorithm	6
2.1	Basic DECOA	26
2.2	LP phase of DECOA	28
2.3	DECOA algorithm for convex problems	31
2.4	Cut generation per block	32
3.1	Cut and break-point generation	54
3.2	Solving sub-problems using OA	55
3.3	OA initialization	55
3.4	OA-based local search	56
3.5	DECOA for nonconvex MINLP problems	58
3.6	Fixation-based cut and break-point generation	59
4.1	Initialization of LP-IA	66
4.2	Column Generation	67
4.3	Initialize DIOR	70
4.4	DIOR for computing a MIP outer approximation	71
4.5	Select block for refinement	78
4.6	Refinement of the cell D_{ku_k}	79
4.7	CG for sub-paths	80
4.8	Inner refinement	81
4.9	The heuristic DIOR for computing a MIP inner approximation	82
5.1	Generation of columns	92

LIST OF TABLES

5.2	Column generation	92
5.3	IA initialization	93
5.4	Approximate sub-problem solving	94
5.5	Approximate Column Generation	95
5.6	Fast Column Generation using a Frank-Wolfe approach	96
5.7	Initial heuristic algorithm to compute a solution candidate	98
5.8	Heuristic algorithm to compute solution candidates	99
5.9	The heuristic CG Algorithm	100

Acronyms

BB	Branch-and-bound
CG	Column Generation
DC	Difference of Convex Functions
DECOA	Decomposition-based Outer Approximation
DESS	Decentralized Energy Supply System
ECP	Extended Cutting Plane
ESH	Extended Supporting Hyperplane
FW	Frank-Wolfe
IA	Inner Approximation
LP	Linear Programming
LP-IA	Linear Programming Inner Approximation
LP-OA	Linear Programming Outer Approximation
MINLP	Mixed-integer Nonlinear Programming
MIP	Mixed-integer Programming
MIP-IA	Mixed-integer Programming Inner Approximation

Acronyms

MIP-OA	Mixed-integer Programming Outer Approximation
MOP	Multi-objective Problem
NDP	Nondominated Point
NLP	Nonlinear Programming
OA	Outer Approximation
OBBT	Optimization-based Bound Tightening
RCP	Resource-constrained Program
SOS2	Special Ordered Set of type 2

This chapter introduces several key concepts that will be repeated throughout the thesis. It first describes a general and block-separable formulation of mixed-integer nonlinear programming (MINLP) problems. Moreover, it illustrates one of the ideas on how to reformulate a general problem into a block-separable formulation. A large collection of optimization problems in the so-called MINLPLib is briefly described. Moreover, several key concepts to be used in the thesis are introduced, (i) Outer Approximation (OA), (ii) resource-constrained program (RCP) and its properties, and (iii) Inner Approximation (IA), which is a basis for Column Generation (CG). Several other relevant concepts are also discussed in this chapter.

1.1 Block-separable formulation of MINLP

A general mixed-integer nonlinear program (MINLP) can be written as

$$\begin{aligned}
 & \min_{x \in \mathbb{R}^p} f(x) \\
 & \text{s. t. } u(x) \leq 0, \\
 & \quad h(x) = 0, \\
 & \quad x \in [\underline{x}, \bar{x}], \\
 & \quad x_i \in \mathbb{Z}, i \in I,
 \end{aligned} \tag{1.1}$$

where $I \subseteq \{1, \dots, p\}$ is the index set of the integer variables; scalar function f and vector functions u, h are linear or nonlinear. If f and u are convex and h is affine, then problem (1.1) is *convex*. Otherwise, the problem is classified as a *nonconvex* MINLP.

Throughout this thesis, we consider a *block-separable* (or *quasi-separable*) reformu-

1. INTRODUCTION

lation of MINLP problem (1.1) of the form

$$\min c^T x \quad \text{s. t. } x \in P, \quad x_k \in X_k, \quad k \in K \quad (1.2)$$

with *global (coupling)* linear constraints

$$P := \{x \in \mathbb{R}^n : a_i^T x \leq b_i, \quad i \in M_1, \quad a_j^T x = b_j, \quad j \in M_2\} \quad (1.3)$$

and *local* constraints

$$X_k := G_k \cap L_k \cap Y_k, \quad (1.4)$$

where

$$\begin{aligned} G_k &:= \{y \in [x_k, \bar{x}_k] \subset \mathbb{R}^{n_k} : g_{ki}(y) \leq 0, \quad i \in [m_k]\}, \\ L_k &:= \{y \in \mathbb{R}^{n_k} : a_{ki}^T y \leq b_{ki}, \quad i \in J_k\}, \\ Y_k &:= \{y \in \mathbb{R}^{n_k} : y_i \in \mathbb{Z}, \quad i \in I_k\}. \end{aligned} \quad (1.5)$$

The vector of variables $x \in \mathbb{R}^n$ is partitioned into $|K|$ disjoint blocks such that $n = \sum_{k \in K} n_k$, where n_k is the dimension of block k and K denotes an index set. The symbol x_k denotes the variables of block k , $x_k \in \mathbb{R}^{n_k}$. The vectors $\underline{x}, \bar{x} \in \mathbb{R}^n$ denote lower and upper bounds on the variables.

The linear constraints defining feasible set P are called *global*. Set P is defined by $a_i^T x = \sum_{k \in K} a_{ki}^T x_k$, $a_{ki} \in \mathbb{R}^{n_k}$, $b_i \in \mathbb{R}$, $i \in [m]$, $m = |M_1| + |M_2|$. Here, we use the notation $[m] := \{1, \dots, m\}$ for an index set that contains m elements. The constraints defining feasible set X_k are called *local*. Set G_k captures the m_k *local nonlinear constraints*. The constraint functions, $g_{kj} : \mathbb{R}^{n_k} \rightarrow \mathbb{R}$, are bounded and continuously differentiable within the set $[x_k, \bar{x}_k]$. Set L_k captures the $|J_k|$ *local linear constraints* and set Y_k is defined by *integer values* of variables x_{ki} , $i \in I_k$, where I_k is an index set. The linear objective function is defined by $c^T x := \sum_{k \in K} c_k^T x_k$, $c_k \in \mathbb{R}^{n_k}$. Moreover, we define the combined sets as follows

$$X := \prod_{k \in K} X_k, \quad G := \prod_{k \in K} G_k, \quad Y := \prod_{k \in K} Y_k. \quad (1.6)$$

An example of problem structure defined by (1.2) is sketched in Figure 1.1. It illustrates that variables from different blocks are connected with coupling constraints and, additionally, the variables within the blocks are connected with local nonlinear constraints. Decomposition methods for solving block-separable MINLP problem (1.2) are based on subdivision of the problem into easier subproblems and combining their solutions into an easier global master problem. Decomposition approaches emerged in the sixties, e.g. Benders decomposition for MIP problems [8] and Dantzig-Wolfe decomposition for LP

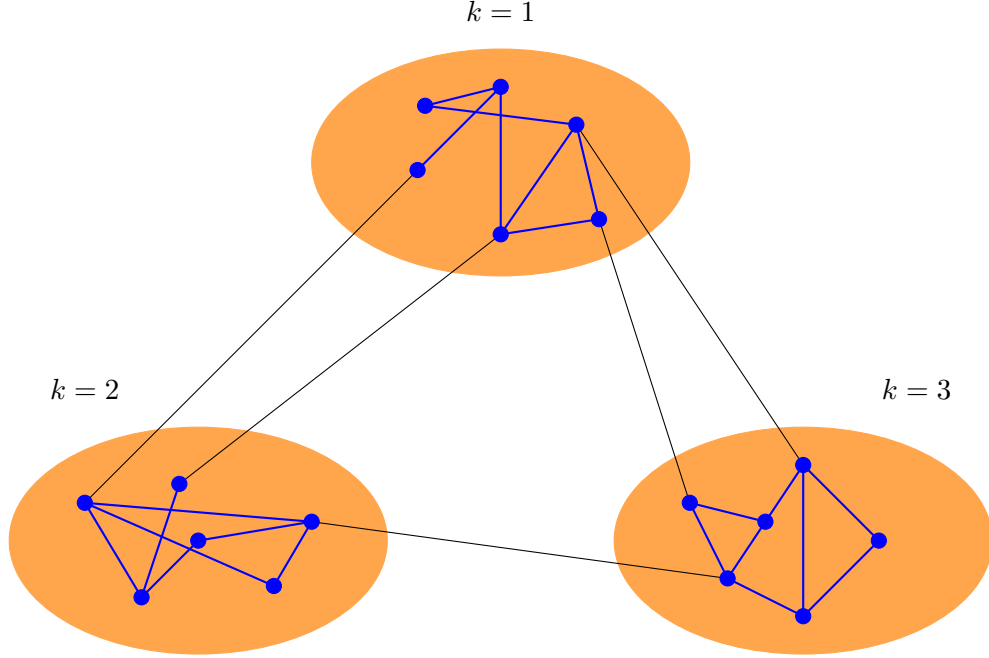


Figure 1.1: Example of block-separable problem structure defined by (1.2) for the case $|K| = 3$. The black edges represent set P defined by coupling constraints. The blue dots show variables x_k and the blue edges represent set X_k defined by local constraints.

problems [23, 24]. There exist several other decomposition approaches, e.g. Lagrangian decomposition [43], etc.

It has been shown that any MINLP problem (1.1) can be reformulated as a block-separable with a given arbitrary maximum block-size n_k by adding new variables and global copy-constraints [91, 100, 104]. In [91], an approach is sketched to define the blocks, which does not require to add new copy-constraints and it adds only new variables. It is based on detection of natural block structure using the Hessian of constraint functions of problem (1.1). More details of this approach are presented in Section 1.3.

1.2 MINLPLib – a collection of MINLP instances

MINLPLib is a large collection of optimization problems [78]. Since 2001, it assembles a set of real-world instances varying from small-scale to large-scale [16]. These instances originate from many different applications (chemical, civil and electrical engineering, finance, management, operations research, agricultural economics, etc.) and its source

1. INTRODUCTION

is indicated in the library. The purpose of the library is to provide an access to the research community to interesting models developed in different environments. The collection is continuously growing, and, as of 28 January 2021, it consists of 1750 MINLP and NLP instances.

The primary format for all instances is the GAMS (`.gms`) format. Nevertheless, many other formats are available, e.g. AMPL (`.mod`, `.nl`). The library contains instances with different properties, e.g. convex and nonconvex problems, quadratic problems, binary problems, etc. Such categorizing of the instances helps to select easily a subset of instances with desired properties. Moreover, each instance contains the best known primal and dual bounds and the best incumbent solution point.

A subset of instances from MINLPLib was used for testing and benchmarking the algorithms, described in the following chapters.

1.3 Natural block structure identification

Usually, problems are given in a general form, as in (1.1). To reformulate these problems into block-separable problem (1.2), we perform two steps:

1. Block structure identification, i.e. computation of the set K . This step is based on identifying connected components of a Hessian adjacency graph.
2. Reformulation itself. This procedure adds new auxiliary variables such that the local constraints are nonlinear while the global constraints and objective function are linear.

Consider MINLP problem (1.1) defined by p variables and functions f, u and h . For the sake of simplicity, define $\phi = (f, u, h)^T$, $\phi : \mathbb{R}^p \rightarrow \mathbb{R}^M$. Consider a Hessian adjacency graph $\mathcal{G} = (V, E)$ defined by the following vertex and edge sets

$$\begin{aligned} V &= \{1, \dots, p\}, \\ E &= \{(i, j) \in V \times V : \frac{\partial^2 \phi_\ell}{\partial x_i \partial x_j} \neq 0, \ell \in [M]\}. \end{aligned} \tag{1.7}$$

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

In fact, we do not compute the Hessian of function ϕ . Instead, we iterate over the (nonlinear) expressions of function ϕ . 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 , which correspond to the connected components of graph \mathcal{G} , we reformulate the original problem into a block-separable MINLP problem, described in (1.2). We perform this procedure by adding new variables and constraints such that the objective function and global constraints are linear and the local constraints are nonlinear.

1.4 Branch-and-bound

The branch-and-bound (BB) method emerged in the early sixties [62] for solving combinatorial optimization problems, e.g. for the travelling salesman problem [67]. Due to its generality, the approach was further elaborated and applied to different classes of optimization problems [21, 30, 52]. BB provides a basis for most of current state-of-the-art solvers for MINLP problems. Examples of commercial solvers are ANTIGONE [79], BARON [97, 100], SCIP [1, 2], etc. Open-source solvers include Cbc [36], Couenne [7], GALINI [19], etc. Usually, these solvers are based on a variant of branch-and-bound or on a combination of variants, i.e. branch-cut-and-price [26], branch-decompose-and-cut [96], branch-and-refine [64], branch-and-reduce [101], α BB [3, 34], etc.

The BB algorithm recursively splits (**branches**) the original problem into smaller disjoint sub-problems until the optimal solution is found and verified. These sub-problems are stored in a tree structure. The idea of **bounding** consists of pruning the nodes of the tree (sub-problems) which do not contain an optimal solution. The important rules of BB methods are: branching (a strategy how a feasible domain is partitioned to create new sub-problems), selection (a strategy how the search tree is explored) and elimination (a strategy how nodes of the tree are pruned to prevent its unnecessary partition) [54, 80].

Algorithm 1.1 presents a generic branch-and-bound procedure [49] to compute the minimum of a continuous function f over a compact feasible set X . At every stage of the algorithm, there exist a global upper bound f^U of the optimal objective function value. Moreover, the procedure maintains a list of sub-domains Λ , which are subsequently splitted (branched). At the beginning, the algorithm computes an enclosure C – also called convex superset C – of set X by calling procedure $\text{ENCLOSURE}(X)$. The algorithm uses enclosure C to compute an initial lower and upper bound, denoted by f_1^L and f_1^U , respectively. The lower bound f_1^L over set C is computed by calling procedure $\text{GETLOWERBOUND}(f, C)$. There exist a variety of approaches [51, 52] to compute the lower bound of the optimal objective function value over set C , e.g. McCormick envelopes [74, 88], interval arithmetic [49], etc. If the algorithm failed to compute a

1. INTRODUCTION

Algorithm 1.1 Generic branch-and-bound algorithm

```

1: function BB( $f, X$ )
2:    $\Lambda \leftarrow \emptyset$ ,  $f^U \leftarrow \infty$ ,  $C \leftarrow \text{ENCLOSURE}(X)$  #  $X \subset C$ 
3:    $f_1^L \leftarrow \text{GETLOWERBOUND}(f, C)$ 
4:    $\hat{x} \leftarrow \text{GETFEASIBLEPOINT}(f, C, X)$ 
5:   if  $\hat{x} \in C \cap X$  then
6:      $\Lambda \leftarrow C$ ,  $f^U \leftarrow f(\hat{x})$ ,  $\ell \leftarrow 1$ 
7:   while  $\Lambda \neq \emptyset$  do
8:      $\Omega \leftarrow \text{SELECT}(\Lambda)$ ,  $\Lambda \leftarrow \Lambda \setminus \{\Omega\}$ ,  $(\Omega_{\ell+1}, \dots, \Omega_{\ell+p}) \leftarrow \text{BRANCH}(\Omega)$ 
9:     for  $i := \ell + 1$  to  $\ell + p$  do
10:       $f_i^L \leftarrow \text{GETLOWERBOUND}(f, \Omega_i)$ 
11:      for  $i := \ell + 1$  to  $\ell + p$  do
12:        if  $\Omega_i \cap X = \emptyset$  or  $f_i^L > f^U$  then # prune
13:           $f_i^L \leftarrow \infty$ ,  $\Lambda \leftarrow \Lambda \setminus \{\Omega_i\}$ 
14:          if  $f_i^L < f^U$  then
15:             $x_i \leftarrow \text{GETFEASIBLEPOINT}(f, \Omega_i, X)$ ,  $f_i^U \leftarrow f(x_i)$ 
16:            if  $f_i^U < f^U$  then
17:               $f^U \leftarrow f_i^U$ ,  $\hat{x} \leftarrow x_i$ 
18:              for  $\Omega_j \in \Lambda$  do # prune
19:                if  $f_j^L > f^U$  then  $\Lambda \leftarrow \Lambda \setminus \{\Omega_j\}$ 
20:                if  $f_i^L > f^U - \epsilon$  then return  $\hat{x}$  # stop
21:                if  $\text{size}(\Omega_i) > \delta$  then  $\Lambda \leftarrow \Lambda \cup \{\Omega_i\}$ 
22:       $\ell \leftarrow \ell + p$ 
23:   return  $\hat{x}$ 

```

feasible solution $\hat{x} \in C \cap X$ by calling procedure $\text{GETFEASIBLEPOINT}(f, C, X)$, the problem is infeasible and the algorithm terminates. Otherwise, list Λ is initilaized with C and global upper bound f^U is updated with f_1^U . In Algorithm 1.1, index ℓ represents a number of generated sub-domains. Note that ℓ does not give the number of elements in list Λ .

The algorithm iteratively calls procedure SELECT , which selects a sub-domain Ω from list Λ . Then, it removes this sub-domain from list Λ and splits it into p partition sets by calling procedure BRANCH . Over each partition set Ω_i , the algorithm computes a related lower bound f_i^L by calling procedure $\text{GETLOWERBOUND}(f, \Omega_i)$. Then, for each partition Ω_i , the algorithm checks whether a feasible point can be computed, i.e.

it checks whether $\Omega_i \cap X \neq \emptyset$. If $\Omega_i \cap X = \emptyset$, then it is not possible to compute a feasible solution regarding the partition Ω_i . Therefore, Ω_i can be eliminated from list Λ . At this place, the algorithm also compares the value of lower bound f_i^L with global upper bound f^U . If lower bound f_i^L is greater than global upper bound f^U , then it means that partition Ω_i can not contain an optimal solution point. In this case, partition Ω_i is left out (pruned) from list Λ .

In the next step, the algorithm compares lower bound f_i^L with global upper bound f^U . If lower bound f_i^L is smaller than f^U , then algorithm computes a feasible point $x_i \in \Omega_i \cap X$ by calling procedure `GETFEASIBLEPOINT`(f, Ω_i, X) and sets upper bound f_i^U to $f(x_i)$. Then, the algorithm compares global upper bound f^U to upper bound f_i^U , computed over set Ω_i . If f_i^U improves f^U , i.e. $f_i^U < f^U$, then f^U and \hat{x} are updated with f_i^U and x_i , respectively. If global upper bound f^U was updated, then the algorithm performs another pruning operation by comparing lower bound f_j^L to global upper bound f^U for all sub-domains $\Omega_j \in \Lambda$. This operation eliminates all sub-domains Ω_j with $f_j^L > f^U$ from the search tree. For simplicity, we assume that procedure `GETFEASIBLEPOINT` can always either compute a feasible point or return an infeasibility flag. However, in practice, to find a feasible point might be as difficult as to compute an optimal solution. In this case, Algorithm 1.1 should be modified such that it can start the branching procedure without global upper bound f^U . Then, the algorithm can initialize global upper bound f^U by computing a feasible solution regarding one of the partitions Ω_i . Note that global upper bound f^U is necessary to terminate the algorithm.

It may occur that further partition of sub-domain Ω_i will not lead to a significant improvement of existing bounds. To avoid this issue, the algorithm incorporates an additional rule of adding partition set Ω_i to list Λ based on its size, i.e.

$$size(\Omega) = \max_{u, w \in \Omega} \|u - w\|. \quad (1.8)$$

If the size of sub-domain Ω_i is above a given threshold δ , then this partition is added to list Λ . The algorithm terminates in two cases: (i) when list Λ is empty or (ii) if the difference between lower bound f_i^L and global upper bound f^U fulfills an ϵ -accuracy.

The goal of the algorithm is to compute quickly a high-quality upper bound f^U and verify it by a lower bound. The performance of the algorithm depends on the strategies how selection, branching and pruning are performed. There are several ideas to select the next node of the tree (sub-domain) to perform further refinement [54, 81]. One can select the nodes with the lowest lower bound; take the sub-domains with the largest size, which are relatively unexplored; perform breadth-first search. There exist

1. INTRODUCTION

various branching strategies, e.g. fixing the variable [81], dividing the domain of a single variable into several sub-domains (spatial branching) [7]. Pruning can be performed in different ways and in various situations, e.g. when the new partition set yields an infeasible problem; after improvement of the global upper bound, the sub-problems with the larger lower bound can be discarded; size of the sub-domain is smaller than a predefined accuracy; the lower bound of sub-problems is improved after generating cutting planes [49, 81].

The branch-and-bound method is time-consuming. Depending on the size of the given problem, the number of nodes in the tree can be too large to fit in the computer memory and it becomes very difficult for the algorithm to compute a high-quality lower bound in a reasonable time. The branch-and-bound algorithm can be parallelized, but it is a challenging task [50].

Example 1.1. We present a simple numerical example from [101] to illustrate the ideas of BB, presented by Algorithm 1.1. For simplicity, the discussed example is defined in a general way. Consider the following nonconvex continuous optimization problem

$$\min_{x \in X} \{f(x) = -x_1 - x_2\}$$

with $X = \{0 \leq x_1 \leq 6, 0 \leq x_2 \leq 4, x_1 x_2 \leq 4\}$. This problem has a global optimum of $-6\frac{2}{3}$ at point $(6, \frac{2}{3})$ and a local optimum of -5 at point $(1, 4)$.

Figure 1.2 illustrates the resulting BB tree created by Algorithm 1.1. At each node, the algorithm computes a lower bound by constructing a convex relaxation with a separable reformulation [101] and an upper bound by performing a local search. The procedure starts by computing upper and lower bounds over the whole set X . These results are stored at the root node of the tree. For branching, the algorithm selects variable x_2 , since branching regarding x_1 would not improve the existing bounds. In the next step, the procedure splits the feasible set into two subsets by dividing the domain of variable x_2 into two pieces, i.e. $x_2 \leq 2$ and $x_2 \geq 2$. The right node is pruned, since the yielded lower bound of the sub-problem is greater than the best upper bound. This means that this sub-domain does not contain an optimal solution point and can be discarded from further exploration. The left node yields a tighter lower bound and its corresponding sub-problem is branched again regarding variable x_2 . The branching of this sub-problem results in adding inequalities $x_2 \leq 1$ and $x_2 \geq 1$. The right node is pruned, since the lower bound is greater than the best known upper bound. At the left node, the algorithm proves optimality within an accuracy of 0.01.

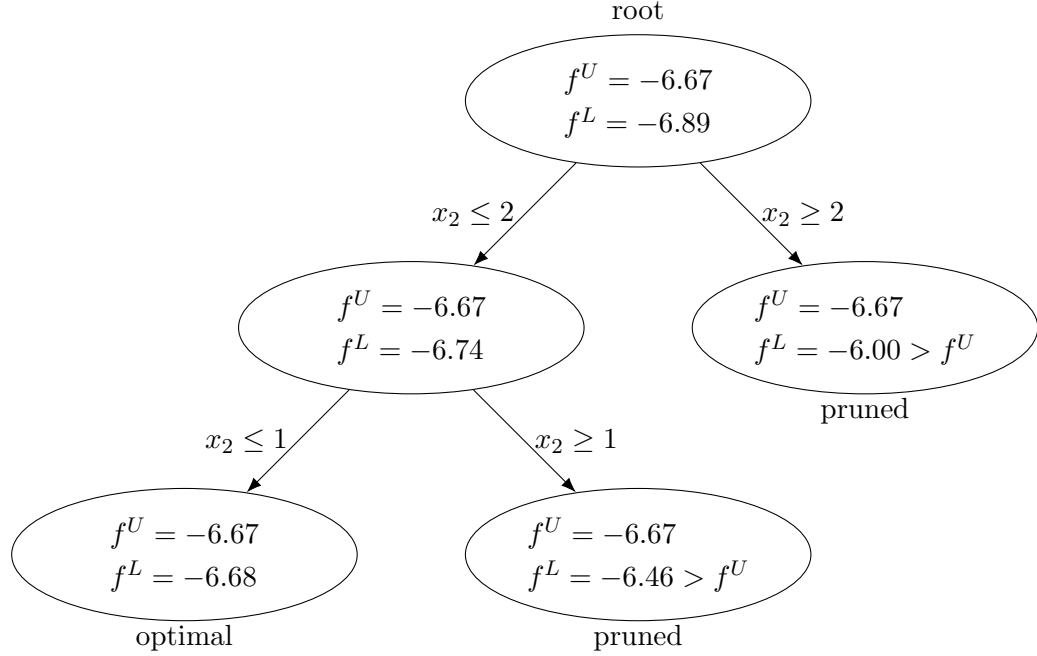


Figure 1.2: Resulting branch-and-bound tree for Example 1.1. Indices at the values for lower and upper bounds are omitted.

1.5 Outer Approximation

Outer Approximation (OA) is a well-known approach to solve MINLP problems. The idea of the method is to construct a linear representation of the original MINLP problem. The technique sequentially refines an OA of the nonlinear feasible set to eliminate infeasible solutions. To conduct an OA algorithm, one typically solves a MIP and an NLP relaxation of the original MINLP problem. Usually, OA is represented by a MIP problem, which is refined by adding new cutting planes. The OA construction differs depending on the nature of the nonlinear feasible set.

Consider the case, when the original MINLP is defined by convex differentiable functions. To approximate the nonlinear feasible set, it is sufficient to compute linearizations of constraint functions at any point. Such linearizations form valid cuts, i.e. linear inequalities that are satisfied for all feasible solutions, but exclude infeasible parts of the whole search space. These cuts provide an outer approximation of the nonlinear feasible set, see Figure 1.3. This property gives a theoretical foundation for solution approaches relying on OA. The first OA method for convex MINLP emerged in the eighties [28], based on solving alternately MIP and NLP sub-problem. In [33], the

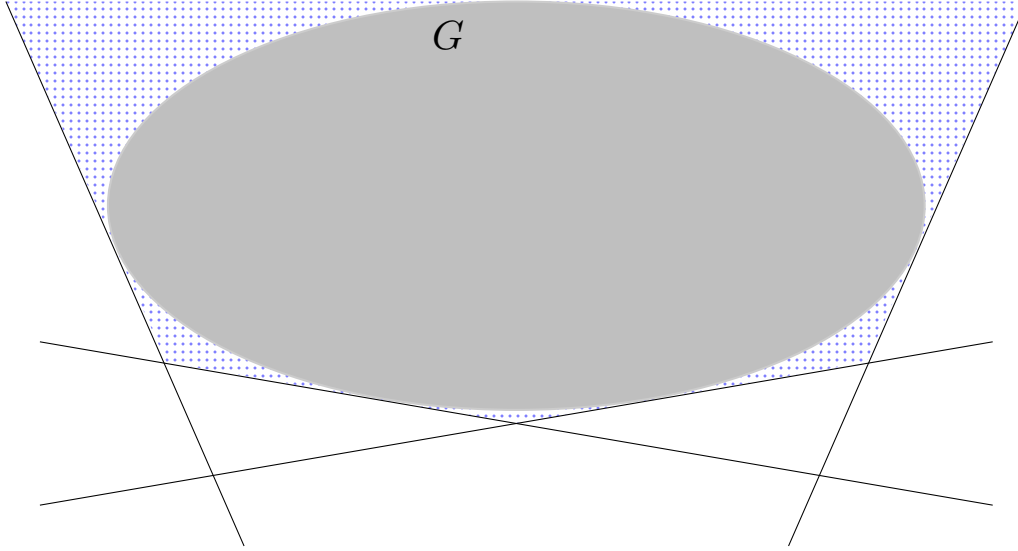


Figure 1.3: Outer approximation of nonlinear feasible set G defined by convex nonlinear constraint functions. The OA is defined by the set of hyperplanes and is represented by the blue dots.

authors further elaborated this approach by extending it to the wider class of convex MINLP problems. Like OA, the Extended Cutting Plane (ECP) algorithm constructs linearizations for convex constraint functions, but it does not use an NLP sub-problem [108]. Unlike the OA and ECP, the Extended Supporting Hyperplane (ESH) algorithm constructs supporting hyperplanes to the feasible set [60]. Figure 1.3 depicts the idea of supporting hyperplanes to the convex nonlinear feasible set. Similar to OA, there exist other algorithms for convex MINLP, e.g. generalized Benders decomposition [42], quadratic OA [58, 99], etc. The OA-based SHOT solver [70] is currently the best solver for convex MINLP [59]. There are other OA-based solvers, e.g. AOA [53], BONMIN [13], DICOPT [46, 56], Pajarito [20], etc.

Now, consider the case when the nonlinear feasible set is defined by nonconvex constraint functions. Unlike in OA for convex problems, the linearizations of nonconvex constraint functions yield not necessarily valid hyperplanes. Therefore, to guarantee that the hyperplanes are valid, one typically employs a convex outer approximation of the nonconvex feasible set. Construction of a convex outer approximation is often called a convexification process [101]. The convexification aims to achieve a tight convex outer approximation of a nonconvex feasible set. Most of the approaches that convexify a nonconvex feasible set rely on properties of particular mathematical structures, e.g. concave, quadratic, bilinear, fractional functions; difference of convex functions [49], etc.

[51, 52] provide a comprehensive overview on these and other mathematical structures and how they can be exploited. Note that the separable reformulation, used to compute lower bounds in Example 1.1, is an additional example of the mathematical structures mentioned above. The α BB approach uses a convexification technique, which can be applied to arbitrary twice differentiable functions [34, 73, 76].

We describe an example of a mathematical structure and its convex under- and overestimators developed by McCormick. For polynomials, he proposed a recursive approach to derive convex under- and overestimators of bilinear combinations of its terms [74]. Consider the following set

$$\Omega = \{(x, y, z) \in \mathbb{R}^3 : x \in [\underline{x}, \bar{x}], y \in [\underline{y}, \bar{y}], z = xy\}. \quad (1.9)$$

The bilinear function that defines set Ω in (1.9) can be under- and overestimated by the following inequalities

$$\begin{aligned} z &\geq \underline{x}y + \underline{y}x - \underline{x}\underline{y}, \\ z &\geq \bar{x}y + \bar{y}x - \bar{x}\bar{y}, \\ z &\leq \underline{x}y + \bar{y}x - \underline{x}\bar{y}, \\ z &\leq \bar{x}y + \underline{y}x - \bar{x}\underline{y}. \end{aligned} \quad (1.10)$$

McCormick inequalities (1.10) represent the convex hull of set Ω [4]. Figure 1.4 illustrates the comparison between convex underestimators (1.10) and the bilinear function used to define set Ω in (1.9).

In order to utilize mathematical structures, most of the solvers analyze constraint expressions and, if necessary, transform them such that the particular mathematical structure is separated. This is done by employing so-called lifted reformulations, where additional variables and constraints are introduced [65]. Mathematical structures are used in branch-and-bound to derive valid lower bounds. For example, McCormick envelopes (1.10) are extensively used in the current state-of-the-art solvers, e.g. BARON [97, 100], Couenne [7], SCIP [1, 2], etc. Several existing approaches do not use explicitly branch-and-bound to get tight lower bounds, i.e. they do not consider splitting the original problem into several sub-problems. Instead, they compute a lower bound for the entire problem by building a piecewise nonconvex outer approximation using mathematical structures. [71] proposed an approach to construct a tight nonconvex outer approximation by iteratively adding partition points. The Alpine solver [88, 89] keeps the number of partition points constant and iteratively refines a nonconvex outer approximation by adapting these points.

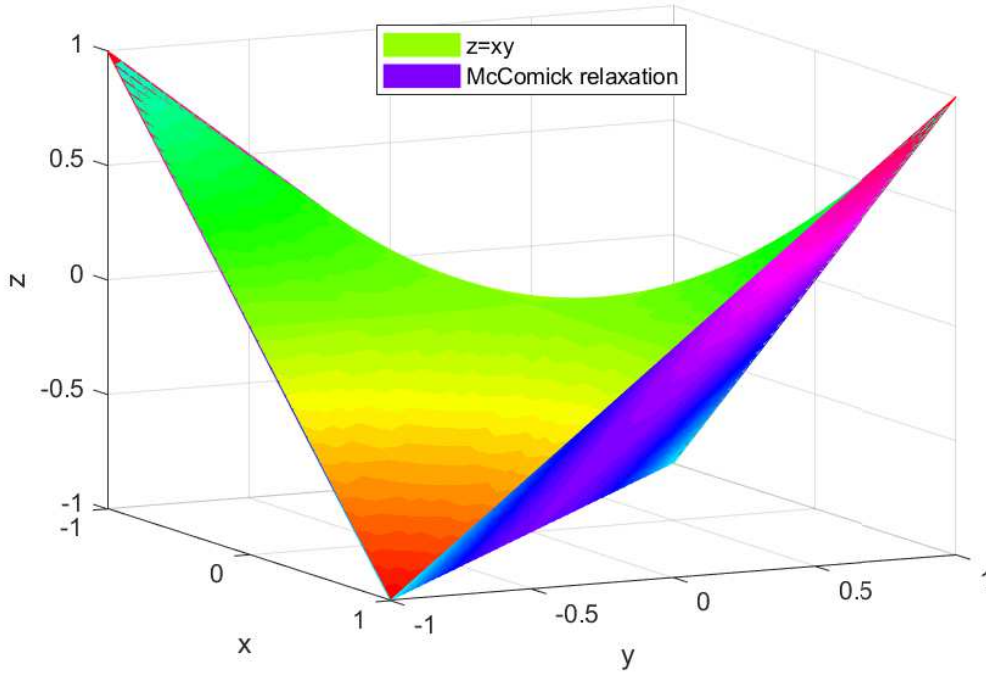


Figure 1.4: Comparison of bilinear function $z = xy$ and corresponding McCormick underestimators on interval $x \in [-1, 1], y \in [-1, 1]$.

Chapter 2 deals with the question whether it is possible to solve convex MINLP problems by applying OA and decomposition with a moderate number of MIP problems. Moreover, it investigates the influence of different cut generation methods on overall algorithm performance.

Finally, Chapter 3 focuses on the question how a piecewise convex relaxation can be constructed for an arbitrary twice differentiable function. Furthermore, it focuses on the question how to set the partition points such that a MIP problem, used for formulation of the nonconvex outer approximation, can be solved in a reasonable time.

1.6 Resource-constrained reformulation

For instances of MINLP problem (1.2) with many variables, solution via traditional MINLP approaches may not be feasible in reasonable time. In stochastic programming or when optimizing systems with differential equations, the optimization problems are typically defined by sub-problems consisting of many variables. For these problems, the number of global constraints connecting the sub-problems might be significantly

smaller than the sub-problem size. For such problems, a resource-constrained view can be promising. This idea has been developed by [12]. In this section, we define a resource-constrained program based on the reformulation of problem (1.2) and discuss its properties.

1.6.1 Definition of the resource-constrained program

Consider problem (1.2) with m global constraints. Recall that $c_k, k \in K$ and $a_{ki}, i \in [m], k \in K$ denote objective vector and vectors appearing in the left-hand side of global constraints P , respectively. Define the matrix A_k by

$$A_{ki} := \begin{cases} c_k^T & : i = 0, \\ a_{ki}^T & : i \in [m], \end{cases} \quad (1.11)$$

and consider the transformed feasible set:

$$W_k := \{A_k x_k : x_k \in X_k\} \subset \mathbb{R}^{m+1}. \quad (1.12)$$

The variables $w_k := A_k x_k$ define the *resources* of block k . The full vector of resources w consists of concatenated vectors $w_k, k \in K$, i.e. $w = (w_1^T, \dots, w_{|K|}^T)^T$. Example 1.4 illustrates transformation (1.12). The resources describe how the objective value and global constraint values $a_i^T x$ are distributed among the blocks. Note that for sparse MINLP problems, the number of nonzero resources, for which $A_{ki} \neq 0$, can be much smaller than m , see Section 1.6.3. Let

$$H := \{w \in \prod_{k \in K} \mathbb{R}^{m+1} : \sum_{k \in K} w_{ki} \leq b_i, \ i \in M_1, \ \sum_{k \in K} w_{kj} = b_j, \ j \in M_2\} \quad (1.13)$$

denote the global constraints in the resource space. We reformulate problem (1.2) as a resource-constrained program (RCP) in the following way

$$\min \sum_{k \in K} w_{k0} \quad \text{s. t. } w \in H, \ w_k \in W_k, \ k \in K. \quad (1.14)$$

Proposition 1.2. Problem (1.2) and (1.14) are equivalent to the following two-level program

$$\min \sum_{k \in K} w_{k0}^* \quad \text{s. t. } w \in H, \quad (1.15)$$

1. INTRODUCTION

where w_{k0}^* is the optimal value of the RCP sub-problem given by

$$\begin{aligned} w_{k0}^* &:= \min c_k^T x_k \\ \text{s. t. } &A_{ki}x_k \leq w_{ki}, \quad i \in M_1, \\ &A_{ki}x_k = w_{ki}, \quad i \in M_2, \\ &x_k \in X_k, \quad k \in K. \end{aligned} \tag{1.16}$$

In other words, the objective values corresponding to the global optimal solution points of (1.2) and (1.14) are equal to the objective value corresponding to the global optimal solution point of (1.15).

Proof. Problem (1.2) can be formulated as

$$\begin{aligned} \min & \sum_{k \in K} c_k^T x_k \\ \text{s. t. } &A_{ki}x_k \leq w_{ki}, \quad i \in M_1, \\ &A_{ki}x_k = w_{ki}, \quad i \in M_2, \\ &w \in H, \quad x_k \in X_k, \quad k \in K. \end{aligned} \tag{1.17}$$

This shows that (1.2) and (1.14) are equivalent. For a given solution (w^*, x^*) of (1.17), it follows that x^* fulfills (1.16). Hence, (1.15) is equivalent to (1.17). \square

1.6.2 Multi-objective perspective

The multi-objective view on (1.14) changes the focus from the complete image set W_k to the relevant set of Pareto optimal points. A similar reformulation is presented in [12]. Consider the following multi-objective sub-problem (MOP) of block k , where we aim to minimize $|M_1| + 1$ resources simultaneously

$$\min (A_{ki}x_k)_{i \in M_1 \cup \{0\}} \quad \text{s. t. } x_k \in X_k. \tag{1.18}$$

A feasible solution $x_k \in X_k$ with $w_k = A_k x_k$ dominates another solution $y_k \in X_k$ with $v_k = A_k y_k$ if $w_{ki} \leq v_{ki}$ for all $i \in M_1 \cup \{0\}$ and $w_{ki} < v_{ki}$ for at least one index $i \in M_1 \cup \{0\}$. A feasible solution $x_k \in X_k$ is efficient (or Pareto optimal) if there does not exist another solution that dominates it. An image of a Pareto optimal solution is also known as a nondominated point (NDP). In other words, an NDP is a feasible objective vector for which none of its components can be improved without making at least one of its other components worse. The set of nondominated points of problem (1.18) is called the Pareto front and is denoted as follows

$$W_k^* := \{w \in W_k : w = (w_i)_{i \in M_1 \cup \{0\}} \text{ is a NDP of (1.18)}\} \tag{1.19}$$

and correspondingly, set

$$W^* := \prod_{k \in K} W_k^*. \quad (1.20)$$

The lower bound of the Pareto front W_k^* is defined by ideal point \underline{w}_k as follows

$$\underline{w}_{ki} = \min (A_{ki}x_k) \quad \text{s. t. } x_k \in X_k, \quad i \in M_1 \cup \{0\}. \quad (1.21)$$

In a similar way, the upper bound of Pareto front W_k^* is defined by nadir point \bar{w}_k , see [77] for more details.

Proposition 1.3. The solution of problem (1.14) is attained at $w^* \in W^*$, i.e. w^* is the solution of the following problem

$$\min \sum_{k \in K} w_{k0} \quad \text{s. t. } w \in H, \quad w_k \in W_k^*, \quad k \in K. \quad (1.22)$$

Proof. Assume that a part \hat{w}_k^* of the optimal solution w^* does not belong to the Pareto front, i.e. $\hat{w}_k^* \notin W_k^*$. Then, $\exists \hat{w}_k \in W_k^*$ that dominates w_k^* , i.e. $\hat{w}_{ki} \leq w_{ki}^*$ for $i \in \{0\} \cup M_1$. Consider \hat{w} the corresponding solution, where in w^* the parts w_k^* are replaced by \hat{w}_k . Then \hat{w} is feasible for RCP given

$$\sum_{k \in K} \hat{w}_{ki} \leq \sum_{k \in K} w_{ki}^* \leq b_i, \quad (1.23)$$

for $i \in M_1$, and its objective function value is at least as good as that of w^* as

$$\sum_{k \in K} \hat{w}_{k0} \leq \sum_{k \in K} w_{k0}^*. \quad (1.24)$$

which means that the optimum is attained at a nondominated point $\hat{w} \in W^*$. \square

1.6.3 Dimension reduction of the resources

In many practical problems, some of the blocks may not appear in a global constraint. To make use of this characteristic in terms of dimension reduction, we should consider the exact properties. Consider the index set of *relevant resources*

$$\begin{aligned} M_{1k} &:= \{i \in \{0\} \cup M_1 : A_{ki} \neq 0\}, \\ M_{2k} &:= \{i \in M_2 : A_{ki} \neq 0\}. \end{aligned} \quad (1.25)$$

1. INTRODUCTION

Then RCP problem (1.14) can be formulated as:

$$\begin{aligned}
& \min \sum_{k \in K, 0 \in M_{1k}} w_{k0} \\
& \text{s. t.} \quad \sum_{k \in K, i \in M_{1k}} w_{ki} \leq b_i, \quad i \in M_1, \\
& \quad \sum_{k \in K, i \in M_{2k}} w_{ki} = b_i, \quad i \in M_2, \\
& \quad w_k \in \Pi_k(W_k), \quad k \in K,
\end{aligned} \tag{1.26}$$

where the projection operator $\Pi_k : \mathbb{R}^{m+1} \rightarrow \mathbb{R}^{|M_{1k}|+|M_{2k}|}$ is defined by

$$\Pi_k(w) := (w_i)_{i \in M_{1k} \cup M_{2k}}. \tag{1.27}$$

Similarly, following Proposition 1.3, (1.22) is equivalent to

$$\begin{aligned}
& \min \sum_{k \in K, 0 \in M_{1k}} w_{k0} \\
& \text{s. t.} \quad \sum_{k \in K, i \in M_{1k}} w_{ki} \leq b_i, \quad i \in M_1, \\
& \quad \sum_{k \in K, i \in M_{2k}} w_{ki} = b_i, \quad i \in M_2, \\
& \quad w_k \in \Pi_k(W_k^*), \quad k \in K.
\end{aligned} \tag{1.28}$$

Formulations (1.26) and (1.28) are of interest, because the number $\sum_{k \in K} |M_{1k}| + |M_{2k}|$ of relevant resources is usually significantly smaller than number of variables n in the original problem. This is the case for sparse optimization models, for which model components are coupled by a moderate number of global constraints. For instance, let $m = 1499$, $|K| = 50$ and number of nonzero rows of $A_k, k \in K$ to be 100, i.e. $|M_{1k}| + |M_{2k}| = 100, k \in K$. Then, the overall number of resource variables is 75000. This number is reduced to 5000, if we consider only the nonzero rows of matrix A_k .

1.6.4 Supported nondominated points

A common method to compute a NDP is the weighted sum method [77], which solves an optimization problem with a single objective obtained as a positive (convex) combination of the objective functions of the multiobjective problem:

$$\min d^T A_{M_{1k}} x_k \quad \text{s. t.} \quad x_k \in X_k. \tag{1.29}$$

For a positive weight vector $d \in \mathbb{R}_+^{|M_{1k}|}$, any optimal solution of (1.29) is an efficient solution of (1.18), i.e. its image is nondominated. Such a solution and its image define

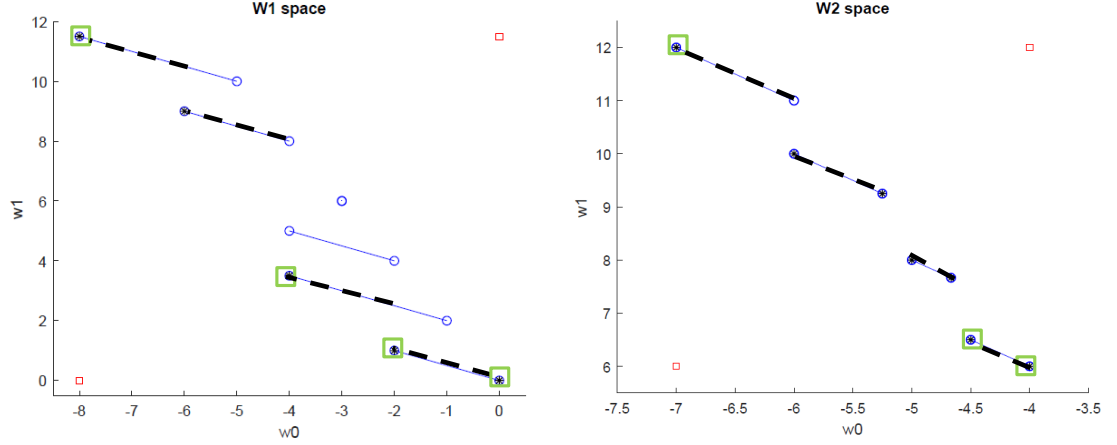


Figure 1.5: Resource constraint space of blocks $k = 1, 2$. Image spaces W_1 and W_2 in blue with extreme points as circles. Pareto front in black, with extreme points as a star. Supported Pareto points are marked with a green square. As a red square, the ideal \underline{w}_k (left-under) and the nadir point \bar{w}_k (right-up).

a *supported efficient solution* and a *supported NDP*, respectively. Thus, an efficient solution x_k is supported if there exists a positive vector d for which x_k is an optimal solution of (1.29), otherwise x_k is unsupported.

Example 1.4. We introduce a numerical example which can be calculated by hand to illustrate the introduced concepts. We present this example in block-separable form (1.2). Let $n = 4$, $K = \{1, 2\}$, $c = (-1, -2, -1, -1)$, $A = (2, 1, 2, 1)$, $b = 10$, $\underline{x} = (0, 0, 2, 1)$ and $\bar{x} = (5, 1.5, 5, 3)$. Integer variables are $I_1 = \{1\}$, $I_2 = \{3\}$ and the local constraints $g_{11}(x_1, x_2) = 3x_2 - x_1^3 + 6x_1^2 - 8x_1 - 3$ and $g_{21}(x_3, x_4) = x_4 - \frac{5}{x_3} - 5$. One can verify that the optimal solution is $x = (1, 1.5, 2, 2.5)$ with objective value -8.5 . In the resource space, this corresponds to the points $w_1 = (-4, 3.5)$ in space W_1 and $w_2 = (-4.5, 6.5)$ in space W_2 .

Figure 1.5 sketches the image spaces W_1 and W_2 with the corresponding Pareto front. For block $k = 2$, the image nearly coincides with the Pareto front W_2^* , although the number of supported points is limited. For block $k = 1$, one can observe more clearly parts of the feasible space that are dominated.

1. INTRODUCTION

1.7 Inner Approximation

In this section, we introduce the concept of Inner Approximation (IA) of original problem (1.2), see Figure 1.6. Consider the following problem

$$\min c^T x \quad \text{s. t. } x \in P, \quad x_k \in \text{conv}(X_k), \quad k \in K. \quad (1.30)$$

Problem (1.30) defines a *convex relaxation* of original problem (1.2). Note that convex relaxation (1.30) is equivalent to the *Lagrangian relaxation* of problem (1.2) regarding the global constraints, see Lemma 3.7 of [91] for the proof. The resource-constrained formulation of problem (1.30) is defined by

$$\min \sum_{k \in K} w_{k0} \quad \text{s. t. } w \in H, \quad w_k \in \text{conv}(W_k), \quad k \in K. \quad (1.31)$$

The quality of convex relaxation (1.30) of MINLP (1.2) depends strongly on the *duality gap*, defined by

$$\text{gap} := \text{val}(1.2) - \text{val}(1.30), \quad (1.32)$$

where $\text{val}(1.2)$ and $\text{val}(1.30)$ denote an optimal objective value of problem (1.2) and problem (1.30), respectively. Given a finite set of feasible points

$$S_k := \{y_{kj}\}_{j \in [S_k]} \subset X_k, \quad (1.33)$$

we have that

$$\min c^T x \quad \text{s. t. } x \in P, \quad x_k \in \text{conv}(S_k), \quad k \in K \quad (1.34)$$

is an inner approximation (IA) of (1.30), see Figure 1.6. Here, we use the notation $[S_k] := \{1, \dots, |S_k|\}$ for an index set of discrete set S_k . The resource-constrained variant of (1.34) is given by

$$\min \sum_{k \in K} w_{k0} \quad \text{s. t. } w \in H, \quad w_k \in \text{conv}(R_k), \quad k \in K, \quad (1.35)$$

where $R_k := \{A_k y : y \in S_k\}$ is a set of columns $r_{kj} \in \mathbb{R}^{m+1}$. Using (1.35), we define a linear programming inner approximation (LP-IA) problem as follows

$$\min \sum_{k \in K} w_{k0}(z_k) \quad \text{s. t. } w(z) \in H, \quad z_k \in \Delta_{|R_k|}, \quad k \in K, \quad (1.36)$$

where

$$w(z) := \sum_{k \in K} w_k(z_k), \quad w_k(z_k) := \sum_{j \in [R_k]} z_{kj} r_{kj}, \quad r_{kj} \in R_k, \quad z_k \in \Delta_{|R_k|}, \quad (1.37)$$

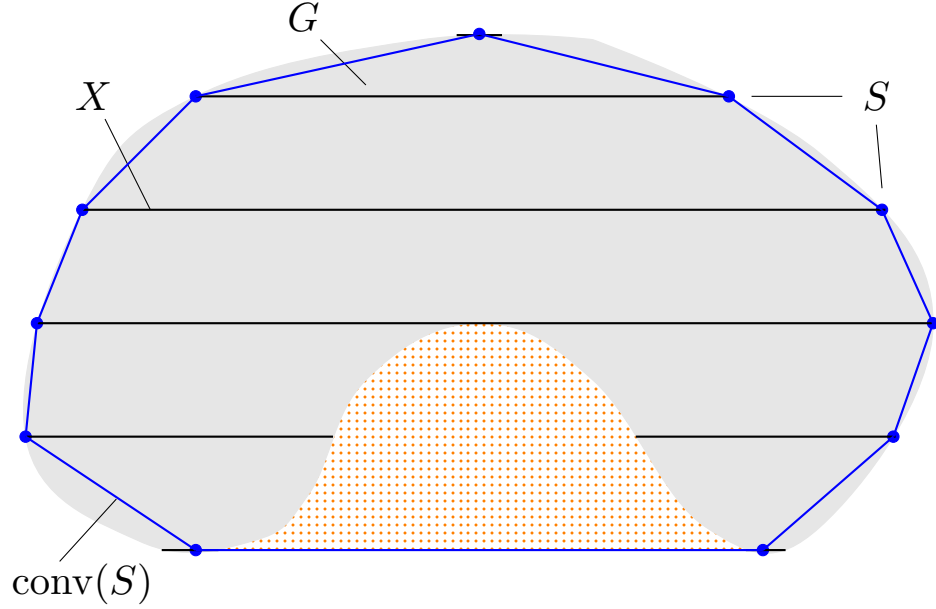


Figure 1.6: Inner approximation of set X denoted by $\text{conv}(S)$ is illustrated by the interior of the blue polygon. The grey shaded area illustrates nonlinear feasible set G . The black lines depict the mixed-integer nonlinear feasible set X . The blue points represent set of feasible points $S \subset X$. The orange points illustrate infeasible region inside of $\text{conv}(S)$.

and $\Delta_{|R_k|}$ defines the standard simplex in the following way

$$\Delta_{|R_k|} = \left\{ z \in \mathbb{R}^{|R_k|} : \sum_{j \in [R_k]} z_j = 1, z_j \geq 0, j \in [R_k] \right\}. \quad (1.38)$$

There exist several methods to solve a Lagrangian relaxation (convex relaxation (1.30)). [43] provides a systematic approach to solve mixed-integer linear programming problems by applying decomposition and Lagrangian relaxation. [31, 63] describe Lagrangian decomposition approaches for solving nonconvex problems. These methods require sub-problems to be solved exactly.

Another technique to solve convex relaxation (1.30) is Column Generation (CG). This approach deals with the whole set of variables implicitly, which makes it an attractive method to solve large-scale problems. Such approach first appeared in the late fifties [35]. In the early sixties, [23, 24] introduced a Column Generation method for LP problems based on Dantzig-Wolfe decomposition. [25, 68] provide a comprehensive overview of the CG method.

1. INTRODUCTION

The computation of convex relaxation (1.30) using Column Generation is based on solving large easy LP master problems (1.34) and smaller nonconvex MINLP sub-problems over set $X_k, k \in K$. In contrast to the Lagrangian decomposition methods, CG does not need optimal solution points of sub-problems. It is sufficient to compute feasible points with a negative reduced cost [91]. In order to be efficient, a fast sub-solver for these sub-problems is necessary. Examples of fast sub-solvers are (truncated) branch-and-cut [38], dynamic programming-based constrained shortest path [29], or MIP-OA [59, 88], etc.

The question of Chapter 4 is how to compute the solution of the original problem starting from the convex relaxation computed by CG. For this purpose, it investigates the potential of OA and IA refinement procedures.

Finally, Chapter 5 mainly concentrates on the question how different acceleration techniques affect the CG convergence. Moreover, it studies the capabilities of a primal heuristic algorithm which is based on projecting a solution of LP-IA problem (1.34) onto the feasible set. Moreover, it investigates the potential of generating a solution pool for computing multiple near-optimal solutions.

1.8 Multi- and single-tree methods

The approaches that involve solving a MIP problem (like OA) can be distinguished as single- and multi-tree approaches [70]. If the algorithm solves a sequence of individual MIP instances, i.e. a new branch-and-bound tree is built in each call of a MIP solver, it is called a multi-tree approach. Solving a sequence of MIP problems does not require a deep integration with a MIP solver, and this approach can be easily implemented. The procedure can be briefly described as follows: (i) calling a MIP solver to solve the existing MIP instance, (ii) handling the result of the last solver call, e.g. updating the MIP instance by cut generation, checking termination criteria, etc. An advantage of multi-tree algorithms is that they possess the possibility to modify the model during the solution process. However, one has to construct a new branch-and-bound tree on each solver call, which might be computationally demanding.

A single-tree algorithm solves only one MIP problem, i.e. it uses the same branch-and-bound tree throughout the solution process. The MIP problem can be refined by adding new lazy constraints through solver callbacks. The solver callbacks are activated whenever a new integer-feasible solution is found. Then, the main algorithm determines whether a new cutting plane should be generated. If a new constraint is generated, it is added to the existing branch-and-bound tree. After that, the solver

continues the solution process with the same tree. Examples of solvers that use single-tree approach are BONMIN [13], AOA [53], Minotaur [72], etc. A single-tree approach is less computationally demanding than the multi-tree approach, but it is possible to make a multi-tree approach efficient as well [70].

1.9 Research questions

In this section, we sum up the research questions discussed above and outline the thesis.

1. Chapter 2 deals with the question whether it is possible to solve convex MINLP problems by applying OA and decomposition with a moderate number of MIP problems. Moreover, it investigates the influence of different cut generation methods on overall algorithm performance.
2. Chapter 3 focuses on the question how a piecewise convex relaxation can be constructed for an arbitrary twice differentiable function. Furthermore, it focuses on the question how to set the partition points such that a MIP problem, used for formulation of the nonconvex outer approximation, can be solved in a reasonable time.
3. The question of Chapter 4 is how to compute the solution of the original problem starting from the convex relaxation computed by CG. For this purpose, it investigates the potential of OA and IA refinement procedures.
4. Chapter 5 mainly concentrates on the question how different acceleration techniques affect the CG convergence. Moreover, it studies the capabilities of a primal heuristic algorithm which is based on projecting a solution of LP-IA problem (1.34) onto the feasible set. Moreover, it investigates the potential of generating a solution pool for computing multiple near-optimal solutions.

Chapter 6 briefly describes the implementation of the DECOGO solver. Finally, Chapter 7 summarizes the work done during the investigation of this thesis and the conclusions obtained from each part.

A Decomposition-based Outer Approximation Algorithm for convex MINLP

Outer approximation (OA) solves a sequence of MIP problems. Solving a large number of MIP instances slows down the convergence of the method. This chapter concentrates on solving convex MINLP problems with OA. The focus of this chapter is on the following research questions: (i) how to solve convex MINLP problems by applying OA and decomposition with a moderate number of MIP problems, (ii) how different cut generation approaches influence the overall algorithm performance. In this chapter, we present two multi-tree Decomposition-based Outer Approximation (DECOA) Algorithms for solving convex MINLP problems. The first one is a basic algorithm, based on solving only MIP instances. The second one is an improved algorithm which contains enhancements for speeding up the convergence.

2.1 Introduction

Outer Approximation is a successive approximation method to solve an optimization problem. In contrast to BB, these methods do not use a single global search tree. Instead, they construct a sequence of trees by solving MIP problems. There exist several OA-based approaches to solve convex MINLP problems by successive linearization of nonlinear constraints, e.g. OA [28], ECP [108], ESH [60], etc. In order to be efficient, the ESH algorithm incorporates additional features, e.g. usage of LP for a rapid OA generation, several cutting plane generation methods, etc.

This chapter describes a multi-tree Decomposition-based Outer Approximation (DECOA) Algorithm for convex MINLP problems. Like ESH, DECOA constructs a MIP outer approximation by generating supporting hyperplanes. These hyperplanes

2. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR CONVEX MINLP

are obtained by linearization of nonlinear constraint functions. The key difference to all other OA approaches is that DECOA uses a decomposition-based cut generation, i.e. linearization cuts are constructed only by solving small sub-problems.

One of the questions of this chapter is whether the number of MIP problems to be solved to reach convergence of the decomposition algorithm can be reduced. In order to generate a tight OA, one can utilize separability of convex nonlinear constraint functions [61]. This approach reformulates constraint functions such that they contain only one nonlinear term. The comparison conducted in [61] shows that OA-based solvers perform better on reformulated problems with separated nonlinear terms. When a general problem (1.1) is reformulated into block-separable problem (1.2), we perform implicitly similar separation as in [61]. However, we do not control the number of nonlinear terms in the reformulated nonlinear constraint functions. Therefore, we may have more than one nonlinear term in the nonlinear constraint function, which may deteriorate the quality of initial OA.

In this chapter, in addition to implicit separation of nonlinear terms of nonlinear constraint functions, we consider several other possibilities to generate quickly a tight OA. For this purpose, we present two versions of the algorithm: basic and enhanced. The basic version uses the projection to generate linearizations, i.e. infeasible points are projected onto the feasible set by solving small sub-problems. In addition to it, it utilizes only a MIP problem for the OA definition. For the basic version of the algorithm, the number of solved MIP problems might be large. This is justified by the experiments with the OA method presented in Section 2.4.

To reduce the number of solved MIP instances and to rapidly generate a tight OA, we present an enhanced two-phase multi-tree DECOA algorithm. In the first phase, the LP phase, the procedure solves a sequence of LP problems. In the second phase, the MIP phase, the algorithm solves a sequence of MIP problems. In contrast to the basic version of DECOA, the enhanced version uses a line-search and fix-and-refine procedure to generate additional hyperplanes. The research question is how these additional methods reduce the number of MIP instances to be solved and influence an overall algorithm performance.

Note that this chapter considers the situation, where the nonlinear feasible set G_k is a convex set, i.e. nonlinear constraint functions $g_{kj}, j \in [m_k], k \in K$ are convex.

This chapter is organized as follows. Section 2.2 presents basic and enhanced DECOA algorithms. Section 2.3 gives a proof of convergence for both DECOA versions. Section 2.4 presents numerical experiments with DECOA on convex MINLP problems from MINLPLib. Finally, Section 2.5 summarizes findings.

2.2 DECOA

DECOA iteratively solves and improves an OA problem, where 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 problem (1.2), 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. The final MIP-OA master problem is a reformulation of problem (1.2). In the following subsections, we describe master problems and sub-problems and outline the basic version of DECOA and prove its convergence. In the end, we describe the full DECOA algorithm with all improvements.

2.2.1 OA master problem

DECOA obtains a solution estimate \hat{x} by solving the following OA master problem

$$\begin{aligned} \min \quad & c^T x \\ \text{s. t. } \quad & x \in P, \quad x_k \in \hat{X}_k, \quad k \in K, \end{aligned} \tag{2.1}$$

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

$$\hat{G}_k := \{x \in \mathbb{R}^{n_k} : \check{g}_{kj}(x) \leq 0, \quad j \in [m_k]\}, \tag{2.2}$$

where

$$\check{g}_{kj}(x) := \max \{ \nabla g_{kj}(\hat{y})^T (x - \hat{y}) : \hat{y} \in T_k \subset \mathbb{R}^{n_k} \}. \tag{2.3}$$

T_k is a set of sample points at the boundary of set G_k and $\check{g}_{kj}(x)$ 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 $\hat{G} := \prod_{k \in K} \hat{G}_k$.

Note that OA (2.1) can be infeasible, if given MINLP model (1.2) is infeasible, e.g. because of data or model errors. Since most MIP solvers, are able to detect the

2. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR CONVEX MINLP

infeasibility of a model, a feasibility flag can be returned after solving (2.1), which stops DECOA, if MINLP model (1.2) is infeasible.

2.2.2 Basic DECOA

In this section, 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 (2.1) defined by

$$\hat{X}_k := Y_k \cap L_k \cap \hat{G}_k, \quad k \in K, \quad (2.4)$$

where Y_k and L_k are integer and linear local constraints defined in (1.5), respectively. After solving the MIP-OA master problem, the following *projection* sub-problem is solved for each block $k \in K$

$$\begin{aligned} \hat{y}_k &= \operatorname{argmin} \|x_k - \hat{x}_k\|^2 \\ \text{s. t. } x_k &\in G_k \cap L_k, \end{aligned} \quad (2.5)$$

where \hat{x}_k is the k -th part of the solution \hat{x} of MIP-OA problem (2.4). The solution \hat{y}_k is used for updating outer approximation \hat{G} by generating new supporting hyperplanes as defined in (2.3). This process is illustrated by Figure 2.1.

Algorithm 2.1 Basic DECOA

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

Algorithm 2.1 describes the basic version of DECOA. Iteratively it solves MIP-OA master problem (2.4) by calling procedure SOLVEMIPOA. Then, the algorithm calls procedure ADDPROJECTCUTS for refinement of set \hat{G} . It performs the projection from point \hat{x} onto the feasible set by solving sub-problems (2.5) and adds linearization cuts at solution points \hat{y}_k . The algorithm iteratively performs these steps until a stopping criterion is fulfilled.

In Section 2.3, Theorem 2.6 proves that Algorithm 2.1 converges to the global optimum of problem (1.2). However, solving only MIP-OA problem (2.4) would be computationally demanding. In order to speed up the convergence, we design an algorithm which reduces the number of times the MIP-OA master problem has to be solved. The improved DECOA algorithm is presented in the two following subsections.

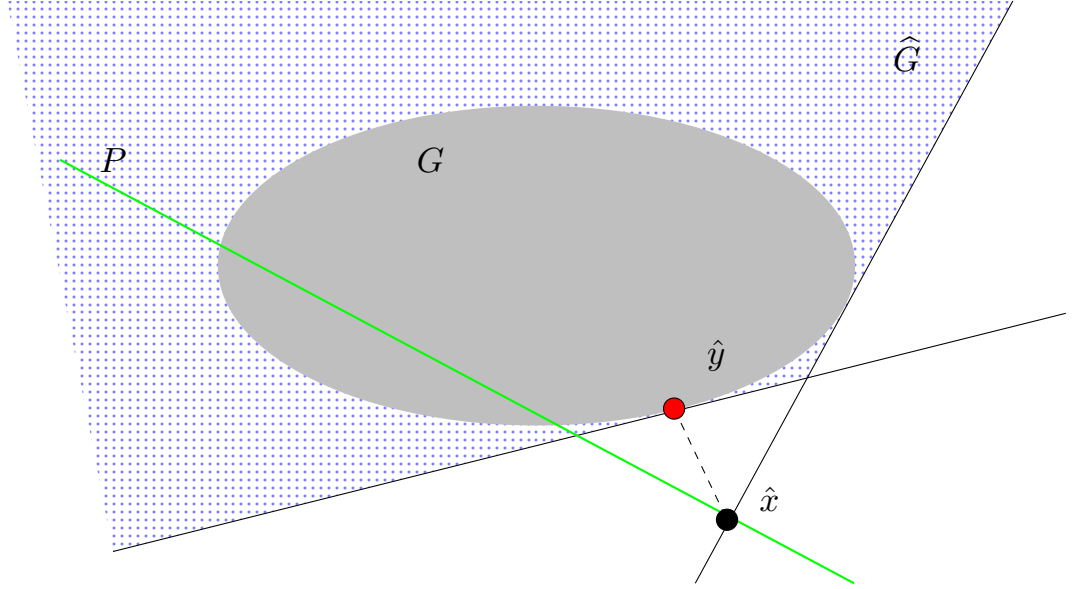


Figure 2.1: Projection of point \hat{x} onto set G defined by (2.5). The blue dots represent outer approximation \hat{G} of nonlinear feasible set G . The green line illustrates linear global constraints P .

2.2.3 The LP phase

In order to generate rapidly an initial outer approximation \hat{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 (2.1) is defined by

$$\hat{X}_k := L_k \cap \hat{G}_k, \quad k \in K. \quad (2.6)$$

Note that integer constraints Y_k are not included in definition (2.6). To further improve the quality of set \hat{G} , the following *line-search* sub-problem can be solved for each $k \in K$

$$\begin{aligned} (\hat{y}_k, \hat{\alpha}_k) = \operatorname{argmax} \quad & \alpha, \\ \text{s. t. } \quad & x = \alpha \hat{x}_k + (1 - \alpha) \check{x}_k, \\ & x \in G_k \cap L_k, \\ & \alpha \in [0, 1], \end{aligned} \quad (2.7)$$

where \hat{x}_k is the k -th part of the solution \hat{x} of LP-OA master problem (2.6) and \check{x}_k is an interior point of set $G_k \cap L_k$. The obtained solution point \hat{y} is an additional sample point for improving outer approximation \hat{G} . The procedure of line-search (2.7) is illustrated in Figure 2.2.

2. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR CONVEX MINLP

For solving line-search sub-problems (2.7), one has to obtain an interior point \check{x} . We consider the following NLP problem

$$\begin{aligned} \check{x} &= \operatorname{argmin} s \\ \text{s. t. } x &\in P, \\ x_k &\in L_k, \\ g_{kj}(x_k) &\leq s, \quad j \in [m_k], \quad k \in K, \quad s \in \mathbb{R}. \end{aligned} \tag{2.8}$$

Note that problem (2.8) is convex, since functions $g_{kj}(x_k) - s \leq 0$ are convex. Given that original problem (1.2) has a solution, then problem (2.8) also has a solution, i.e. $\check{x} \in P \cap \prod_{k \in K} G_k \cap L_k$. It is important that point \check{x} is contained within the interior of set $P \cap \prod_{k \in K} G_k \cap L_k$. If point \check{x} lies on the boundary of set $P \cap \prod_{k \in K} G_k \cap L_k$, the solution of problem (2.7) will always be the same, i.e. supporting hyperplanes will always be the same. In practice, interior point \check{x} can be obtained by solving integer-relaxed NLP problem (1.2), where the objective function is a constant (zero), using an interior point-based NLP solver such as IPOPT [105]. However, such an interior point may not result in the solution of problem (2.8).

Algorithm 2.2 LP phase of DECOA

```

1: function OASTART( $P, X$ )
2:   for  $k \in K$  do  $\hat{G}_k \leftarrow \mathbb{R}^{n_k}$ 
3:   repeat
4:      $\hat{x} \leftarrow \text{SOLVE LPOA}(P, \hat{X})$ 
5:     for  $k \in K$  do  $\hat{G}_k \leftarrow \text{ADDPROJECTCUTS}(\hat{x}_k, L_k, G_k)$ 
6:   until no improvement
7:    $\check{x} \leftarrow \text{SOLVENLPZEROOBJ}(\hat{x}, P, X)$ 
8:   repeat
9:     for  $k \in K$  do  $\hat{G}_k \leftarrow \text{ADDPROJECTCUTS}(\hat{x}_k, L_k, G_k)$ 
10:    for  $k \in K$  do  $\hat{G}_k \leftarrow \text{ADDLINESEARCHCUTS}(\hat{x}_k, \check{x}_k, L_k, G_k)$ 
11:     $\hat{x} \leftarrow \text{SOLVE LPOA}(P, \hat{X})$ 
12:  until no improvement
13:   $(\tilde{x}, \tilde{G}) \leftarrow \text{ADDUNFIXEDNLPCUTS}(\hat{x}, P, X)$ 
14:  return  $(\hat{x}, \check{x}, \tilde{G})$ 

```

Algorithm 2.2 describes the LP phase of the DECOA algorithm for a rapid initialization of the polyhedral outer approximation. At the beginning, it iteratively solves

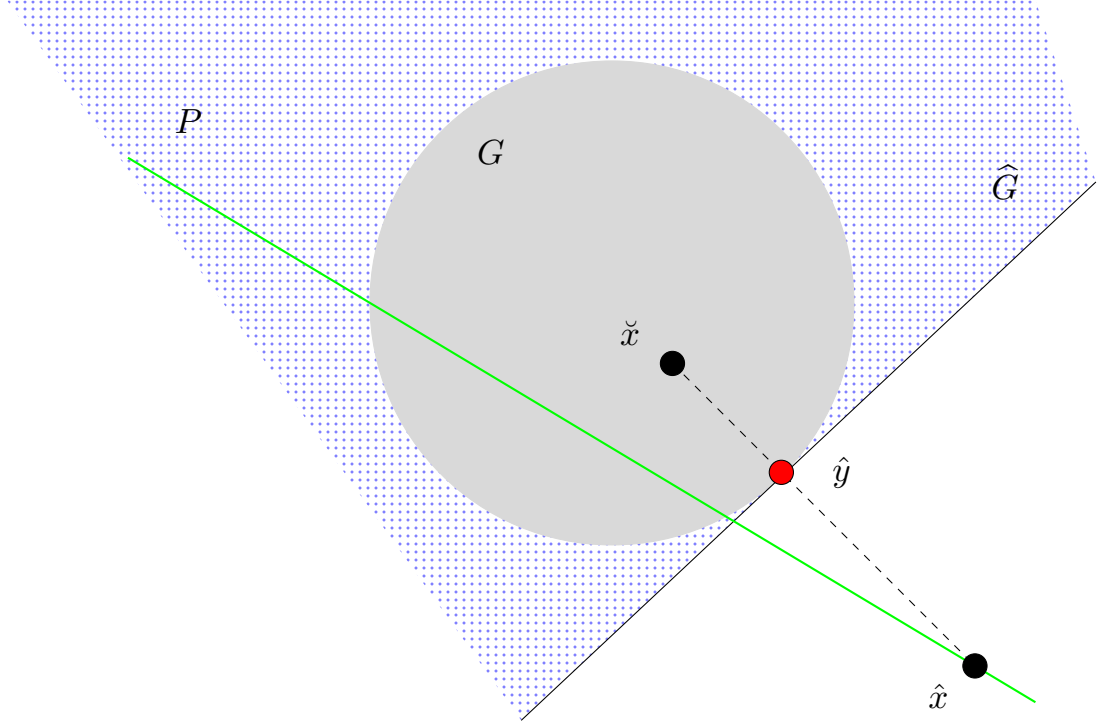


Figure 2.2: The line-search procedure between interior point \tilde{x} and OA point \hat{x} . The blue dots illustrate outer approximation \hat{G} of nonlinear feasible set G . The green line illustrates linear global constraints P .

the LP-OA master problem defined in (2.6) by calling procedure SOLVELPOA. Then, it calls procedure ADDPROJECTCUTS, which solves projection sub-problems (2.5) and 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. $c^T(\hat{x}^p - \hat{x}^{p+1}) < \epsilon$, where ϵ is a desired tolerance.

In order to conduct the line-search, the algorithm finds an interior point \tilde{x} by calling procedure SOLVENLPZEROOBJ. This procedure solves an NLP problem obtained by relaxing the integrality constraints of problem (1.2) and setting its objective function to the zero vector. Next, the algorithm performs a similar loop as before, described in lines 7-10, with procedure ADDLINESEARCHCUTS(\hat{x}, \tilde{x}). This procedure solves line-search sub-problems (2.7) between the LP-OA solution point \hat{x} and interior point \tilde{x} and adds linearization cuts at solution point \hat{y} of the line-search sub-problems. Finally, the algorithm calls procedure ADDUNFIXEDNLP CUTS which computes solution \tilde{x} of integer-relaxed NLP problem (1.2) and adds linearization cuts at solution point \tilde{x} .

2. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR CONVEX MINLP

2.2.4 The MIP phase

Once a tight initial outer approximation has been obtained through the LP phase, the algorithm considers integer constraints Y_k by defining MIP-OA master problem (2.4). After solving MIP-OA master problem (2.4) for the first time and obtaining solution estimate \hat{x} , DECOA computes a solution candidate \tilde{x} by solving the following NLP master problem with *fixed integer variables*

$$\begin{aligned} \min \quad & c^T x \\ \text{s. t. } \quad & x \in P \cap X, \\ & x_{ki} = \hat{x}_{ki}, \quad i \in I_k, \quad k \in K, \end{aligned} \tag{2.9}$$

where I_k is a set of integer variables in block k . Notice that if outer approximation \hat{X} is still not close to set X , (2.9) does not necessarily yield a feasible solution. Let x^* be a primal solution point of problem (1.2) and \bar{v} be a primal bound corresponding to point x^* . If \tilde{x} is feasible and its objective value is lower than \bar{v} , i.e. $\tilde{x} \in X$ and $c^T \tilde{x} \leq \bar{v}$, then x^* is updated with \tilde{x} .

In order to refine further outer approximation \hat{G} by exploiting the block-separability property of problem (1.2), we consider *partly-fixed* OA problems which are defined similar to MIP-OA problem (2.4), but the variables are fixed for all blocks except for one, i.e. for all $k \in K$:

$$\begin{aligned} \min \quad & c^T x \\ \text{s. t. } \quad & x \in P \cap \hat{X}, \\ & x_{\ell i} = \tilde{x}_{\ell i}, \quad i \in n_\ell, \quad \ell \in K \setminus \{k\}, \end{aligned} \tag{2.10}$$

where \tilde{x} is a solution point of NLP problem (2.9).

The solution points of problem (2.10) can be used for the refinement of outer approximation \hat{G} as a base for solving projection sub-problem (2.5). Note that the solution of problem (2.10) provides us information about the fixation of integer variables in problem (2.9). If the fixations in problem (2.9) are feasible, then problem (2.10) has a feasible solution, otherwise, problem (2.10) does not have a feasible solution, because global constraints in set P are not satisfied.

Algorithm 2.3 describes a multi-tree DECOA algorithm which computes a solution estimate \hat{x} by solving MIP-OA master problem (2.4) and solution candidate x^* by solving NLP master problem with fixed integers (2.9). Initially, an upper bound \bar{v} of the optimal value of problem (1.2) 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 2.3 DECOA algorithm for convex problems

```

1: function OASOLVE
2:    $\bar{v} \leftarrow \infty$ 
3:    $x^* \leftarrow \emptyset$ 
4:    $(\hat{x}, \tilde{x}, \hat{G}) \leftarrow \text{OASTART}(P, X)$ 
5:    $\hat{x} \leftarrow \text{SOLVEMIP OA}(P, \hat{X})$ 
6:   repeat
7:      $(\tilde{x}, \hat{G}) \leftarrow \text{ADDFIXEDNLP CUTS}(\hat{x}, P, X)$ 
8:     if  $\tilde{x} \in X$  and  $c^T \tilde{x} < \bar{v}$  then
9:        $x^* \leftarrow \tilde{x}$ 
10:       $\bar{v} \leftarrow c^T \tilde{x}$ 
11:      if  $\bar{v} - c^T \hat{x} < \epsilon$  then
12:        return  $(\hat{x}, x^*, \hat{G})$ 
13:      for  $k \in K$  do  $\hat{G}_k \leftarrow \text{FIXANDREFINE}(\tilde{x}_k, P, \hat{X}_k)$ 
14:      for  $k \in K$  do  $\hat{G}_k \leftarrow \text{ADDPROJECT CUTS}(\hat{x}_k, L_k, G_k)$ 
15:      for  $k \in K$  do  $\hat{G}_k \leftarrow \text{ADDLINESEARCH CUTS}(\hat{x}_k, \tilde{x}_k, L_k, G_k)$ 
16:       $\hat{x} \leftarrow \text{SOLVEMIP OA}(P, \hat{X})$ 
17:    until  $\bar{v} - c^T \hat{x} < \epsilon$ 
18: return  $(\hat{x}, x^*, \hat{G})$ 

```

algorithm calls procedure OASTART, described in Algorithm 2.2, for initializing a tight outer approximation. Procedure SOLVEMIP OA computes solution estimate \hat{x} by solving MIP-OA master problem (2.4).

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 ADDFIXEDNLP CUTS solves NLP master problem with fixed integers (2.9). This procedure uses solution estimate \hat{x} to fix integer variables and returns solution point \tilde{x} , which might be infeasible. If the point \tilde{x} is feasible and objective function value $c^T \tilde{x}$ is lower than the current upper bound \bar{v} , solution candidate x^* and upper bound \bar{v} are updated accordingly. Moreover, if the objective function gap between solution estimate \hat{x} and solution candidate x^* is small enough, i.e. $\bar{v} - c^T \hat{x} < \epsilon$, 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 outer approximation \hat{G} by generating new supporting hyperplanes. For refinement of set \hat{G} , DECOA calls FIXANDREFINE which solves

2. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR CONVEX MINLP

partly-fixed OA problem (2.10). The detailed description of this procedure is given in Algorithm 2.4. Similar to Algorithm 2.2, in order to obtain sample points for new supporting hyperplanes, line-search sub-problems (2.7) and projection sub-problems (2.5) are solved using solution point \hat{x} of MIP-OA master problem (2.4). After refinement of set \hat{G} , DECOA calls SOLVEMIP OA for computing a new solution estimate \hat{x} by solving problem (2.4). If the gap between point \hat{x} and point x^* is closed, DECOA terminates and returns solution estimate \hat{x} , solution candidate x^* and polyhedral outer approximation \hat{G} , which is a reformulation of original problem (1.2).

Algorithm 2.4 Cut generation per block

```

1: function FIXANDREFINE( $\tilde{x}_k, P, \hat{X}_k$ )
2:   repeat
3:      $\hat{x}_k \leftarrow \text{SOLVEFIXMIP OA}(\tilde{x}_k, P, \hat{X}_k)$ 
4:      $\hat{G}_k \leftarrow \text{ADDPROJECTCUTS}(\hat{x}_k, L_k, G_k)$ 
5:   until integer variables of  $\hat{x}$  are not changed
6: return ( $\hat{x}, \hat{G}$ )

```

Algorithm 2.4 describes procedure FIXANDREFINE, which is used for refinement of set \hat{G} . For each block $k \in K$, the algorithm calls procedure SOLVEFIXMIP OA, which solves partly-fixed OA master problem (2.10). 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.

2.3 Proof of convergence

In this section, we prove that the basic DECOA, depicted in Algorithm 2.1, either converges to a global optimum of (1.2) 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 (2.1), (2.4) and projection sub-problems (2.5) are solved to optimality. We also prove the convergence of improved DECOA as outlined in Algorithm 2.3.

Due to the convexity, function $\check{g}_{kj}(x)$ defined in (2.3) is an affine underestimator of function g_{kj} and, therefore, set \hat{X}^p , consisting of the corresponding hyperplanes at iteration p , is an outer approximation of set X . Since the basic DECOA adds new supporting hyperplanes in each iteration, it creates a sequence of sets \hat{X}^p with the following property

$$\widehat{X}^0 \supset \dots \supset \widehat{X}^{p-1} \supset \widehat{X}^p \supset X \quad (2.11)$$

Lemma 2.1. If DECOA, described in Algorithm 2.1, stops after $p < \infty$ iterations and the last solution \hat{x}^p of OA master problem (2.1) fulfills all constraints of (1.2), the solution is also an optimal solution of original problem (1.2).

Proof. We adapt the proof of [60]. Since DECOA stops at iteration p , \hat{x}^p is an optimal solution of (2.1) and \hat{x}^p has the optimal objective function value of (1.2) within $\widehat{X}^p \cap P$. From property (2.11) it is clear that \widehat{X}^p also includes feasible set X . Since \hat{x}^p also satisfies the nonlinear and integrality constraints, it is also in the feasible set, i.e. $\hat{x}^p \in P \cap X$. \hat{x}^p minimizes the objective function within $\widehat{X}^p \cap P$, which includes the entire feasible set, and $\hat{x}^p \in P \cap X$, so it is also an optimal solution of (1.2). \square

In Theorem 2.1, we prove that Algorithm 2.1 generates a sequence of solution points converging to a global optimum. In order to prove this, we present intermediate results in Lemmas 2.2–2.5.

Lemma 2.2. If current solution $\hat{x}^p \notin G$, then Algorithm 2.1 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 (2.5) $\hat{y}_k \neq \hat{x}_k^p$. Note that $\hat{y}_k, \hat{x}_k^p \in L_k$. For this proof, we set $\tilde{G}_k := G_k \cap L_k = \{y \in \mathbb{R}^{n_k} : \tilde{g}_{kj}(y) \leq 0, j \in [\tilde{m}_k], \tilde{m}_k = |m_k| + |J_k|\}$ and replace $G_k \cap L_k$ by \tilde{G}_k in (2.5). Note that the linearization cuts of L_k are not added, since they are the same as linear constraints L_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 (2.5), $\exists \mu_j \geq 0, j \in \mathcal{A}_k$, such that

$$\hat{x}_k^p - \hat{y}_k = \sum_{j \in \mathcal{A}_k} \mu_j \nabla \tilde{g}_{kj}(\hat{y}_k), \quad (2.12)$$

where μ corresponds to constraints \tilde{G}_k . Multiplying (2.12) by $\hat{x}_k^p - \hat{y}_k$ we obtain

$$\left(\sum_{j \in \mathcal{A}_k} \mu_j \nabla \tilde{g}_{kj}(\hat{y}_k) \right)^T (\hat{x}_k^p - \hat{y}_k) = \|\hat{x}_k^p - \hat{y}_k\|^2 > 0. \quad (2.13)$$

Given that $\mu_j \geq 0, j \in \mathcal{A}_k$, there exists at least one $j \in \mathcal{A}_k$ for which $\nabla \tilde{g}_{kj}(\hat{y}_k)^T (\hat{x}_k^p - \hat{y}_k) > 0$. Since Algorithm 2.1 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 $\hat{x}_k^p \notin \widehat{X}^{p+1}$. \square

2. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR CONVEX MINLP

In Lemma 2.3 we show that if Algorithm 2.1 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

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

Since subsequence $\{\hat{x}^{p_i}\}_{i=1}^\infty$ is convergent, there exists a limit $\lim_{i \rightarrow \infty} \hat{x}^{p_i} = z$. In Lemmas 2.4 and 2.5, we show that z is not only within the feasible set of (1.2) but also an optimal solution of (1.2).

Lemma 2.3. If Algorithm 2.1 does not stop in a finite number of iterations, then it generates a convergent subsequence $\{\hat{x}^{p_i}\}_{i=1}^\infty$.

Proof. We adapt the proof of [60]. Since the algorithm has not terminated, none of the solutions of OA master problem (2.1) are in the feasible set, i.e. $\hat{x}^p \notin P \cap X$ for all $p = 1, 2, \dots$ in the solution sequence. Therefore, all the points in the sequence $\{\hat{x}^p\}_{p=1}^\infty$ will be distinct due to Lemma 2.2. Since $\{\hat{x}^p\}_{p=1}^\infty$ contains an infinite number of different points, which are in the compact set P , according to the Bolzano-Weierstrass Theorem, the sequence contains a convergent subsequence. \square

Lemma 2.4. The limit z of any convergent subsequence $\{\hat{x}^{p_i}\}_{i=1}^\infty$, generated in Algorithm 2.1, belongs to the feasible set of (1.2).

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 (2.5) with point $\hat{x}_k^{p_j}$. Consider the following equality

$$\begin{aligned} \|\hat{x}_k^{p_j} - \hat{x}_k^{p_{j+1}}\|^2 &= \|(\hat{x}_k^{p_j} - \hat{y}_k^{p_j}) - (\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j})\|^2 \\ &= \|\hat{x}_k^{p_j} - \hat{y}_k^{p_j}\|^2 + \|\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j}\|^2 \\ &\quad - 2(\hat{x}_k^{p_j} - \hat{y}_k^{p_j})^T (\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j}). \end{aligned} \tag{2.14}$$

Consider the set \tilde{G}_k from the proof of Lemma 2.2 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 2.1 adds for each active nonlinear constraint $i \in \mathcal{A}_k$ the cut

$$\nabla \tilde{g}_{ki}(\hat{y}_k^{p_j})^T (x_k - \hat{y}_k^{p_j}) \leq 0, \tag{2.15}$$

we have

$$\nabla \tilde{g}_{ki}(\hat{y}_k^{p_j})^T (\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j}) \leq 0. \tag{2.16}$$

Using the KKT multipliers in (2.12) and taking into account (2.16), yields

$$\sum_{i \in \mathcal{A}_k} \mu_i \nabla \tilde{g}_{ki}(\hat{y}_k^{p_j})^T (\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j}) = (\hat{x}_k^{p_j} - \hat{y}_k^{p_j})^T (\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j}) \leq 0. \quad (2.17)$$

Since $\|\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j}\|^2 \geq 0$ and $(\hat{x}_k^{p_j} - \hat{y}_k^{p_j})^T (\hat{x}_k^{p_{j+1}} - \hat{y}_k^{p_j}) \leq 0$, (2.14) implies

$$\|\hat{x}_k^{p_j} - \hat{x}_k^{p_{j+1}}\|^2 \geq \|\hat{x}_k^{p_j} - \hat{y}_k^{p_j}\|^2. \quad (2.18)$$

By Lemma 2.3 sequence $\{\hat{x}_k^{p_i}\}_{i=1}^\infty$ is convergent, i.e. $\lim_{j \rightarrow \infty} \hat{x}_k^{p_j} = z_k$, we have that $\lim_{j \rightarrow \infty} \|\hat{x}_k^{p_j} - \hat{x}_k^{p_{j+1}}\| = 0$. This means that $\lim_{j \rightarrow \infty} \|\hat{y}_k^{p_j} - \hat{x}_k^{p_j}\| = 0$. Then we have that $\lim_{j \rightarrow \infty} \|z_k - \hat{y}_k^{p_j}\|^2 = 0$. This implies $\lim_{j \rightarrow \infty} \hat{y}_k^{p_j} = z_k$. 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. $z \in P \cap X$. \square

Lemma 2.5. The limit point of a convergent subsequence, generated in Algorithm 2.1, is a global minimum point of (1.2).

Proof. We adapt the proof of [60]. Because each set \hat{X}^p is an outer approximation of feasible set X , $c^T \hat{x}^{p_i}$ gives a lower bound of the optimal value of the objective function. Due to property (2.11), sequence $\{c^T \hat{x}^{p_i}\}_{i=1}^\infty$ is nondecreasing and since the objective function is continuous, we get $\lim_{i \rightarrow \infty} c^T \hat{x}^{p_i} = c^T z$. According to Lemma 2.4, 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 (1.2). \square

Since Lemmas 2.4 and 2.5 apply to all convergent subsequences generated by OA master problems (2.1), any limit point of such sequence is a global optimum. We summarize the convergence results in the next theorem.

Theorem 2.6. Algorithm 2.1 either finds a global optimum of (1.2) 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 (2.1) satisfies all constraints and according to Lemma 2.1 it is a global optimum of (1.2). In case the algorithm does not stop in a finite number of iterations, it generates a sequence converging to a global optimum of (1.2) according to Lemmas 2.3 and 2.5. \square

In Theorem 2.7, we prove that improved DECOA described in Algorithm 2.3 also converges to a global optimum of (1.2).

2. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR CONVEX MINLP

Theorem 2.7. *DECOA described in Algorithm 2.3 either finds a global optimum of (1.2) 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 2.3, is the same as in basic DECOA described in Algorithm 2.1. In the Algorithm 2.3, we introduce enhancements such as the LP-OA master problem and line-search sub-problems for speeding up the convergence of Algorithm 2.1. Hence, improved Algorithm 2.3 refines outer approximation \hat{X} faster, because, in each iteration, the additional methods make the outer approximation \hat{X} tighter. Moreover, all conditions assumed in the proof of Theorem 2.6 remain valid. Therefore, the proof is similar to the proof of Theorem 2.6. \square

2.4 Numerical results

Algorithm 2.3 was implemented with Pyomo [48], an algebraic modelling language in Python, as a part of the MINLP solver DECOGO. For the experiments, we utilized SCIP 5.0 [44] for solving MIP problems and IPOPT 3.12.8 [106] for solving LP and NLP problems. All computational experiments were performed using a computer with Intel Core i7-7820HQ 2.9 GHz CPU and 16 GB RAM.

DECOA described in Algorithm 2.3 has been tested on convex MINLP problems from MINLPLib [78]. Some instances do not 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:

$$1 < |K| < N, \quad (2.19)$$

where $|K|$ is the number of blocks and N is 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 2.1, we provide more detailed statistics on this set of instances. The selected instances were reformulated into the block-separable form using the technique, described in Section 1.3. Note that, in this case, the convexity of the reformulated problems is unchanged.

As mentioned in Section 2.2, we add the supporting hyperplanes for each active

constraint at point $\hat{y} \in T_k$ according to the formula

$$g_{kj}(\hat{y}) + \nabla g_{kj}(\hat{y})^T(x - \hat{y}) \leq 0, \quad \hat{y} \in T_k. \quad (2.20)$$

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 $g_{kj}(\hat{y})$ in (2.20).

DECOA described in Algorithm 2.3 terminates based on the relative gap, i.e.

$$\frac{|\bar{v} - c^T \hat{x}|}{10^{-12} + |\bar{v}|} < \epsilon, \quad (2.21)$$

where ϵ is a desired tolerance. Moreover, the loops in the LP phase, described in Algorithm 2.2, 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.

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.

2.4.1 Effect of line-search and fix-and-refine

In order to understand the impact of the line-search and fix-and-refine procedure, described in Algorithm 2.4, we ran four variants of Algorithm 2.3:

1. Only projection, i.e. line-search and fix-and-refine were not performed.
2. Projection with fix-and-refine, i.e. line-search was not performed.
3. Projection with line-search, i.e. fix-and-refine was not performed.
4. 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 (2.6), MIP-OA master problems (2.4) and all sub-problems. Note that the sub-problem solution time includes the time spent on solving projection (2.5), line-search (2.7) and partly-fixed OA (2.10) 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 an estimate of possible time improvement.

2. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR CONVEX MINLP

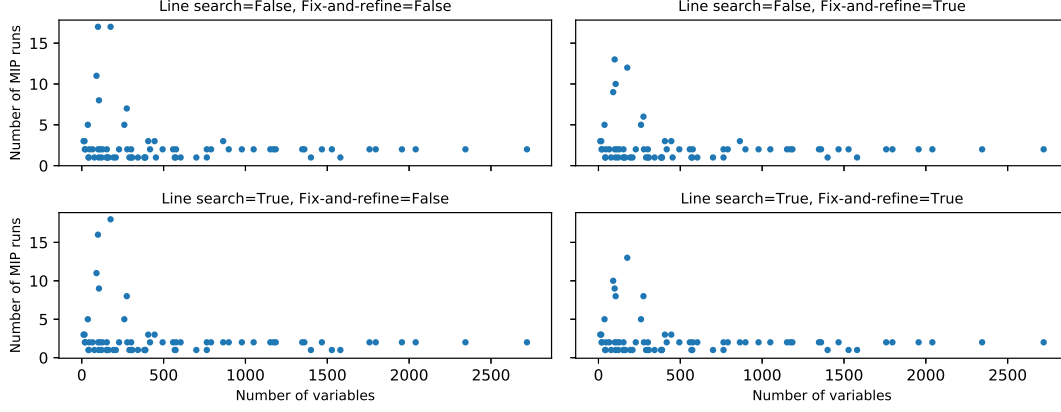


Figure 2.3: Number of MIP runs with respect to the instance size.

Figure 2.3 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 Figure 2.4. 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 Figure 2.5, which shows that the sub-problem time for the 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.

Figure 2.5 presents the average time spent on solving master problems and sub-problems. Sub-problem time corresponds to the time spent on solving projection, line-search and partly-fixed OA sub-problems. Master problem time presents only the time for solving LP-OA and MIP-OA master problems and without the time for solving NLP problems. From Figure 2.5, 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

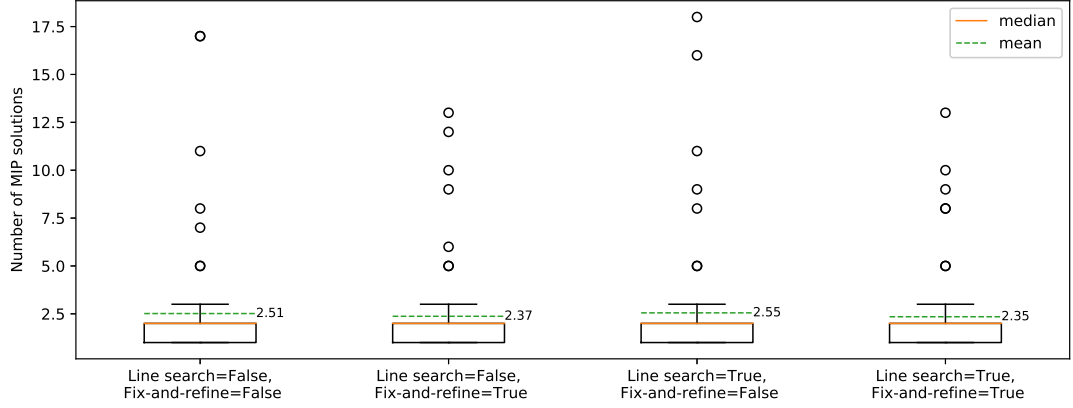


Figure 2.4: The distribution of the number of MIP runs for four variants of Algorithm 2.3 over selected instances.

tight outer approximation and, therefore, to reduce the number of MIP runs. Similar gains in reduction of MIP runs have been achieved in [69]. In contrary to DECOA, [69] proposed to improve the quality of polyhedral OA with extended formulations, which are based on detecting convexity of constraints.

2.4.2 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 optimization problems. For this purpose, we have chosen the branch-and-bound-based SCIP solver 5.0 [44] and Pyomo-based toolbox MindtPy 0.1.0 [10]. 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 and Ipopt 3.12.8 as a MIP solver 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 comparison with both solvers, we use the variant of Algorithm 2.3 without line-search and fix-and-refine. It is the least computationally demanding variant of Algorithm 2.3, as has been shown in Figure 2.5. The test instances were selected from MINLPLib [78] using condition (2.19).

Table 2.1 presents the results for DECOA and SCIP for each instance individually. For each instance, it presents its statistic: problem size N , average blocksize \bar{N}_k after reformulation and measured total solution time T of the DECOA run. Note that the total time T does not include the time spent on the automatic reformulation, described

2. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR CONVEX MINLP

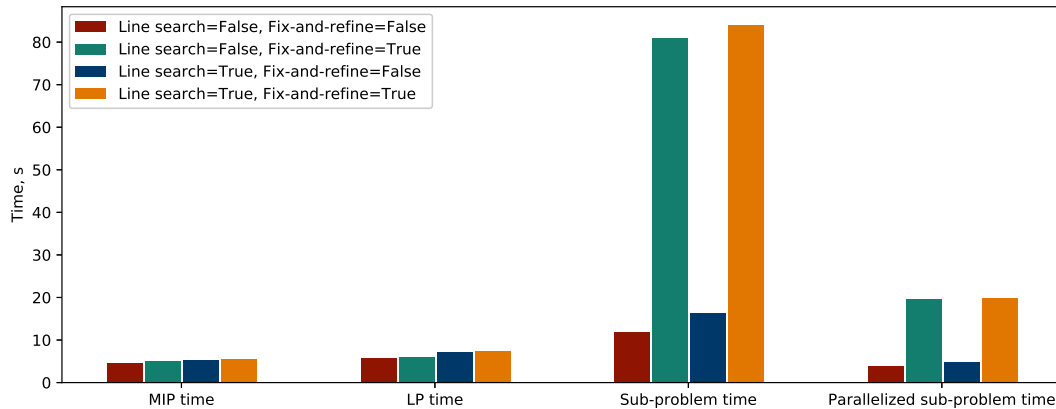


Figure 2.5: The average time spent to solve master problems and sub-problems.

in Section 1.3. 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 (2.5). T_{SCIP} denotes the time spent on solving the original problem with SCIP.

Table 2.1: Performance comparison per instance for variant of Algorithm 2.3 without line-search and fix-and-refine with the SCIP solver

	Instance name	N	\bar{N}_k	T (sec)	T_{MIP} (sec)	N_{MIP}	T_{LP} (sec)	T_{NLP} (sec)	T_{sub} (sec)	T_{SCIP} (sec)
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	batches101006m	278	10.2	31.9	3.4	2	8.6	0.8	17.4	19.5
5	batches121208m	406	12.2	44.6	14.4	3	9.7	1.9	16.0	44.5
6	batches151208m	445	12.4	63.5	22.8	3	16.8	2.3	18.6	56.6
7	batches201210m	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

Table 2.1: (continued)

Instance name	N	\bar{N}_k	T (sec)	T_{MIP} (sec)	N_{MIP}	T_{LP} (sec)	T_{NLP} (sec)	T_{sub} (sec)	T_{SCIP} (sec)
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
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

2. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR CONVEX MINLP

Table 2.1: (continued)

Instance name	N	\overline{N}_k	T (sec)	T_{MIP} (sec)	N_{MIP}	T_{LP} (sec)	T_{NLP} (sec)	T_{sub} (sec)	T_{SCIP} (sec)
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 2.1 compares the solution time of SCIP and DECOA for each instance individually. However, comparing solution time of both solvers can not 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 is that Python is an interpreted language and C is compiled. SCIP also carries out several advanced preprocessing techniques, e.g. constraint analysis, variable bounds tightening,

etc, whereas DECOA does not perform anything like that.

Table 2.1 shows that 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 similar to SCIP, i.e. SCIP time is within 80% of DECOA time. Moreover, for almost all problems, T_{MIP} is small, and T_{sub} is relatively large. Hence, since all sub-problems can be solved in parallel, there is a clear indication that runtime for DECOA can be significantly reduced, see Figure 2.5.

From Table 2.1 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 2.2 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 2.2: Performance comparison per instance for the variant of Algorithm 2.3 without line-search and fix-and-refine with MindtPy using OA strategy.

Instance name		DECOA		OA		
		N_{MIP}	T (sec)	N_{MIP}	T (sec)	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	batches101006m	2	31.9	11	44.4	Converged
5	batches121208m	3	44.6	5	37.9	Converged
6	batches151208m	3	63.5	-	-	Iterations limit
7	batches201210m	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

2. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR CONVEX MINLP

Table 2.2: (continued)

Instance name		DECOA		OA		
		N_{MIP}	T (sec)	N_{MIP}	T (sec)	Status
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
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

Table 2.2: (continued)

Instance name	DECOA		OA		
	N_{MIP}	T (sec)	N_{MIP}	T (sec)	Status
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

Table 2.2 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 2.2 present that for almost all instances, the number of MIP runs N_{MIP} for DECOA is less than the number of

2. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR CONVEX MINLP

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 of OA, i.e. DECOA is more efficient than OA for these problems. This situation is illustrated by instance `c1ay0204h`. 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.

2.5 Conclusions

This chapter presents a multi-tree Decomposition-based Outer Approximation Algorithm for solving convex block-separable MINLP problems (1.2). It iteratively solves and refines an outer approximation problem by generating new supporting hyperplanes. Due to the block-separability of problem (1.2), the sample points for supporting hyperplanes are obtained by solving low-dimensional sub-problems. We presented two versions of DECOA: basic and enhanced. The basic version solves only MIP-OA master problems, whereas the enhanced version solves both LP- and MIP-OA master problems. The enhanced version of algorithm is designed such that the MIP-OA master problems are solved as few times as possible.

One of the question of this chapter was whether the number of MIP instances to be solved to reach convergence can be reduced. The experiments show that in average the algorithm requires only 2 – 3 MIP problems to solve the problem. In order to answer the question whether several additional cut generation methods can reduce further the number of MIP instances to solve, we tested four variants of DECOA on a set of convex MINLP instances. The numerical results show that the average number of MIP runs is reduced further. Moreover, the experiments demonstrate that the average number of MIP runs is independent of the problem size. The algorithm solves a small number of MIP problems, since it generates cuts in the LP phase to obtain a tight OA. However, it might be necessary to solve more MIP problems, if the problem is defined with nonlinear convex constraints that have many (more than one) nonlinear terms. The time measurements illustrate that the time spent on solving sub-problems is larger than the time to solve LP and MIP problems. This demonstrates that the algorithm can be further improved by solving sub-problems in parallel.

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. 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.

This study has been published in [87].

A Decomposition-based Outer Approximation Algorithm for nonconvex MINLP

The aim of this study is to extend the DECOA algorithm, described in Chapter 2, to solving nonconvex MINLP problems. Most of the OA methods for nonconvex problems exploit mathematical structures, as described in Section 1.5. We aim to design a decomposition-based OA approach which solves block-separable nonconvex problems. The proposed method does not make use of mathematical structure of nonconvex constraint functions. We assume that nonconvex nonlinear constraint functions, $g_{kj} : \mathbb{R}^{n_k} \rightarrow \mathbb{R}$, are bounded and twice differentiable within the set $[\underline{x}_k, \bar{x}_k]$.

3.1 Introduction

In Chapter 2, we construct a polyhedral Outer Approximation (OA) by computing valid linearization cuts at sample points. However, when dealing with nonconvex constraint functions, linearization cuts might be invalid. To construct a polyhedral OA of nonconvex feasible set, one typically employs convex under- and overestimators of nonconvex functions. There exist several approaches to define convex under- and overestimators of nonconvex functions [51, 52]. Most of them exploit mathematical structures, e.g. McCormick inequalities for bilinear terms [74]. The disadvantage of these approaches is that they can be utilized only for special classes of functions. One of the research questions of this chapter is how to construct tight convex underestimators for arbitrary functions.

Several approaches solve nonconvex MINLP problems by constructing a piecewise nonconvex OA of nonconvex feasible set. Examples are [15, 88]. However, they rely on mathematical structures, as mentioned before. The challenge of these approaches is an

3. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR NONCONVEX MINLP

effective partitioning of variable domains such that the resulting MIP relaxation is still solvable in a reasonable time. The algorithms proposed by [15, 88] refine the piecewise MIP relaxation by adaptively adding new partition points. Our next research question is how to add efficiently break-points, in order to define a tight piecewise nonconvex OA, which eventually keeps a MIP problem solvable in a reasonable time.

In this chapter, we extend the DECOA algorithm for nonconvex block-separable MINLP problems. For this approach, we assume that nonconvex constraint functions are twice differentiable. These functions are approximated by piecewise polyhedral underestimators. Like DECOA for convex problems, the presented approach is a multi-tree two-phase approach and generates cutting planes and partition points by solving low-dimensional projection sub-problems. It reduces the number of partition points by employing an Optimization-based Bound Tightening (OBBT), a technique to reduce variable bounds [88]. Note that in Algorithm 3 of [93], a variant of DECOA has been presented, which solves nonconvex MINLP problems by adapting partition points without using projection.

This chapter is organized as follows. Section 3.2 explains piecewise nonconvex underestimators of nonconvex functions. Section 3.3 presents OA initialization. Section 3.4 shows the procedure for computing solution candidates. Section 3.5 describes the full DECOA algorithm. Section 3.6 demonstrates a numerical experience with DECOA. Finally, Section 3.7 discusses conclusions.

3.2 Piecewise DC Outer Approximation

Consider a DC (Difference of Convex Functions) formulation of the nonconvex twice differentiable function g_{kj} [103]

$$g_{kj}(x) = h_{kj}(x) - q_{kj}(x), \quad j \in [m_k], \quad k \in K, \quad (3.1)$$

where a convexified nonlinear function h_{kj} and quadratic function q_{kj} are defined as follows

$$\begin{aligned} h_{kj}(x) &:= g_{kj}(x) + q_{kj}(x), \\ q_{kj}(x) &:= \sigma_{kj} \sum_{i \in \mathcal{I}_{kj}} \varphi_{ki}(x_i), \\ \varphi_{ki}(x_i) &:= (x_i - \underline{x}_{ki})(\bar{x}_{ki} - x_i). \end{aligned} \quad (3.2)$$

The set $\mathcal{I}_{kj} = \{i : \frac{\partial g_{kj}}{\partial x_i} \neq 0\}$ denotes an index set of nonlinear variables of constraint function g_{kj} and $\sigma_{kj} \geq 0$ denotes a convexification parameter for the constraint function g_{kj} .

Let $H_{kj} = \nabla^2 g_{kj}$ be the Hessian matrix of constraint function g_{kj} . If matrix H_{kj} is a negative-definite, i.e. all of its eigenvalues are negative, then function g_{kj} is concave. If matrix H_{kj} is a positive semi-definite, i.e. all of its eigenvalues are positive, then function g_{kj} is convex. Based on this information, we set a parameter σ_{kj} , such that function h_{kj} is convex

$$\sigma_{kj} = \max\{0, -v_{kj}\}, \quad (3.3)$$

where v_{kj} is an optimal value of the following nonlinear eigenvalue problem

$$\begin{aligned} v_{kj} &= \min y^T H_{kj}(x)y \\ \text{s. t. } &x \in [\underline{x}_k, \bar{x}_k], \quad y \in \mathbb{R}^{n_k}, \quad \|y\|^2 = 1. \end{aligned} \quad (3.4)$$

According to the min-max theorem, the optimal value of problem (3.4) yields a minimum eigenvalue of matrix H_{kj} . If $v_{kj} < 0$, then function g_{kj} is nonconvex, otherwise, it is convex. Note that parameter σ_{kj} can be computed in the similar way as in the α BB method [3, 34].

Similar to (2.2), we define a convex polyhedral underestimator of convex function h_{kj} as follows

$$\check{h}_{kj}(x) = \max_{\hat{y} \in T_k} \bar{h}_{kj}(x, \hat{y}), \quad j \in [m_k], \quad (3.5)$$

where

$$\bar{h}_{kj}(x, \hat{y}) := h_{kj}(\hat{y}) + \nabla h_{kj}(\hat{y})^T (x - \hat{y}) \quad (3.6)$$

denotes the linearization of h_{kj} at the sample point $\hat{y} \in T_k \subset \mathbb{R}^{n_k}$, similar to (2.3).

Let denote $\mathcal{B}_{ki} := \{p_1, p_2, \dots, p_{|\mathcal{B}_{ki}|}\}$ a set of break-points of a nonconvex variable x_{ki} with the following properties

$$\begin{aligned} p_1 &:= \underline{x}_{ki}, \quad p_{|\mathcal{B}_{ki}|} := \bar{x}_{ki}, \\ p_1 &< p_2 < \dots < p_{|\mathcal{B}_{ki}|}. \end{aligned} \quad (3.7)$$

We replace quadratic function $\varphi_{ki}(x_i)$ by a piecewise linear overestimator $\check{\varphi}_{ki}(x_i)$ as follows

$$\check{\varphi}_{ki}(x_i) := \varphi_{ki}(p_\ell) \frac{p_{\ell+1} - x_i}{p_{\ell+1} - p_\ell} + \varphi_{ki}(p_{\ell+1}) \frac{x_i - p_\ell}{p_{\ell+1} - p_\ell}, \quad (3.8)$$

where $x_i \in [p_\ell, p_{\ell+1}]$, $\ell \in [|\mathcal{B}_{ki}| - 1]$, $p_\ell, p_{\ell+1} \in \mathcal{B}_{ki}$. Using definition (3.8), we denote $\check{q}_{kj}(x)$ as an overestimator of $q_{kj}(x)$

$$\check{q}_{kj}(x) = \sigma_{kj} \sum_{i \in \mathcal{I}_{kj}} \check{\varphi}_{ki}(x_i). \quad (3.9)$$

A DC polyhedral underestimator \check{g}_{kj} of nonconvex constraint function g_{kj} is given by

$$\check{g}_{kj}(x) := \check{h}_{kj}(x) - \check{q}_{kj}(x). \quad (3.10)$$

3. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR NONCONVEX MINLP

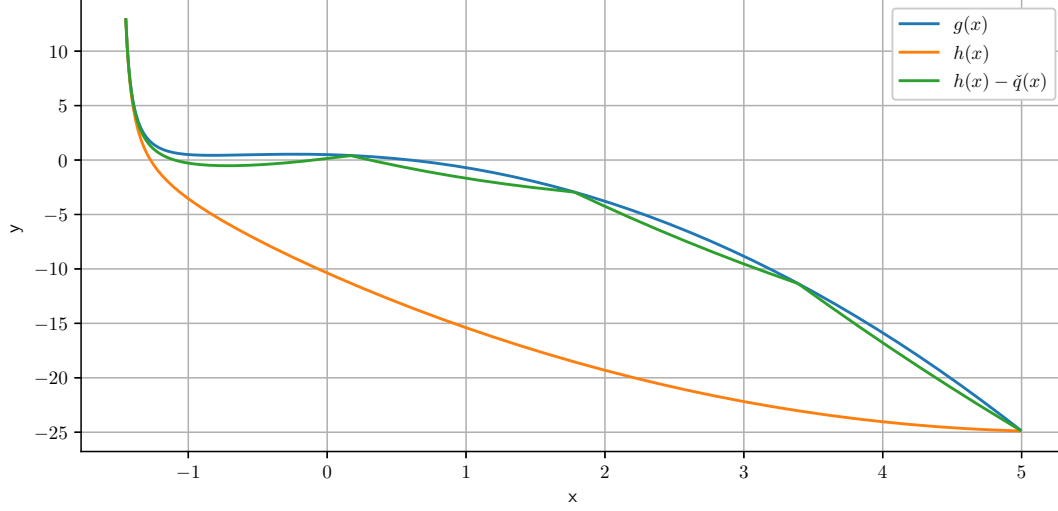


Figure 3.1: Comparison of original function $g(x)$, convexified function $h(x)$ and polyhedral DC underestimator $\check{g}(x)$ for function $g(x) = 3/(4x + 6) - x^2$.

Figure 3.1 shows the comparison between functions $g(x)$, $h(x)$ and $h(x) - \check{q}(x)$. Piecewise nonconvex underestimator $g(x)$ improves a lot over the convexified function $h(x)$.

We denote an outer approximation \hat{G}_k of G_k by

$$\hat{G}_k = \hat{C}_k \cap \hat{Q}_k, \quad (3.11)$$

where

$$\begin{aligned} \hat{C}_k &:= \{y \in [\underline{x}_k, \bar{x}_k], \ r_k \in \mathbb{R}^{n_k} : \check{h}_{kj}(y) - \sigma_{kj} \sum_{i \in \mathcal{I}_{kj}} r_{ki} \leq 0, \ j \in [m_k]\}, \\ \hat{Q}_k &:= \{y \in [\underline{x}_k, \bar{x}_k], \ r_k \in \mathbb{R}^{n_k} : r_k - \check{\varphi}_k(y) \leq 0\}. \end{aligned} \quad (3.12)$$

Moreover, we define combined OA \hat{G} as follows

$$\hat{G} := \prod_{k \in K} \hat{G}_k, \quad \hat{C} := \prod_{k \in K} \hat{C}_k, \quad \hat{Q} := \prod_{k \in K} \hat{Q}_k. \quad (3.13)$$

Polytope \hat{C}_k is defined by *linearization cuts* as in (3.5) and set \hat{Q}_k is defined by break-points \mathcal{B}_k as in (3.8). Note that we use the same definition of \hat{X}_k as in (2.4). To formulate an MIP-OA master problem as in (2.1), set \hat{Q}_k is modelled using Special Ordered Set of type 2, i.e. SOS2 constraints [6]. Typically, SOS2 are modelled with additional binary variables. The number of these additional binary variables corresponds to the size of break-point set \mathcal{B} . Therefore, the complexity of MIP-OA problem strongly depends on the number of break-points in set \mathcal{B} .

3.3 OA initialization

Similar to Algorithm 2.3, we aim to design a two-phase algorithm for nonconvex MINLP problems. In comparison to Algorithm 2.3, we can not perform an initial procedure using the LP-OA master problem as described in Algorithm 2.3. This is due to the fact that OA master problem (3.11) is modelled only by a MIP problem. Therefore, it is important to use as few break-points as possible for keeping the OA master problem relatively easy to solve. The purpose of the start heuristic procedure is to compute an initial set of break-points such that they define a tight initial OA. First, we describe all ingredients for the initialization phase, and then, we present the procedure itself.

Like in Algorithm 2.1, set \widehat{G} is refined by adding cuts. Moreover, the OA is refined by adding new break-points. Algorithm 3.1 describes the procedure of adding the linearization cuts to set \widehat{C} and break-points to set \mathcal{B} . Points \hat{x} and \hat{y} are used in procedure `ADDACTIVELINCUT`(\hat{y}_k, \hat{x}_k) to add linearization cuts (3.5) at \hat{y}_k for all nonlinear constraints $g_{kj}, j \in [m_k]$, which are active at \hat{y}_k and violated at \hat{x}_k .

In the next step, the algorithm adds the break-points for the nonconvex variables. Consider

$$\hat{x}_{ki} \in [p_\ell, p_{\ell+1}], \quad p_\ell, p_{\ell+1} \in \mathcal{B}_{ki}, \quad i \in \mathcal{I}_{kj}, \quad j \in [m_k], \quad k \in K. \quad (3.14)$$

In other words, $[p_\ell, p_{\ell+1}]$ is a break-point interval containing the point \hat{x}_{ki} . Procedure `ADDFLEXIBLELINCUTSANDPOINTS` adds adaptively the following two break-points around point \hat{x}_{ki} (see Figure 3.2), as proposed in [88],

$$\hat{x}_{ki} + \xi, \quad \hat{x}_{ki} - \xi, \quad (3.15)$$

where $\xi = (p_{\ell+1} - p_\ell)/\Delta, \Delta \geq 2, \Delta \in \mathbb{N}$. Using (3.15), interval $[p_\ell, p_{\ell+1}]$ containing \hat{x}_{ki} is partitioned into three new intervals, i.e. $[p_\ell, \hat{x}_{ki} - \xi], [\hat{x}_{ki} - \xi, \hat{x}_{ki} + \xi]$ and $[\hat{x}_{ki} + \xi, p_{\ell+1}]$, see Figure 3.2. Moreover, the procedure adds the cuts for all nonconvex constraints g_{kj} at new break-points (3.15).

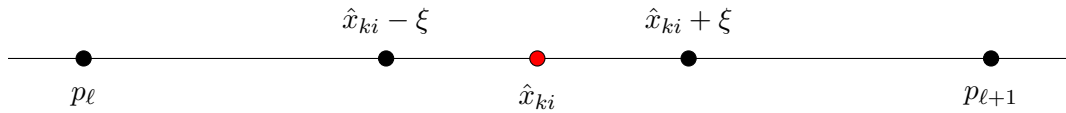


Figure 3.2: Adaptive partition of interval $[p_\ell, p_{\ell+1}]$, defined by (3.15).

In order to initialize the set of break-points, we compute a feasible solution for the following MINLP sub-problem

$$\hat{x}_k = \operatorname{argmin} d_k^T x_k \quad \text{s. t. } x_k \in X_k, \quad (3.16)$$

3. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR NONCONVEX MINLP

Algorithm 3.1 Cut and break-point generation

```

1: function ADDCUTSANDPOINTS( $\hat{x}, \hat{y}, \mathcal{B}$ )
2:   for  $k \in K$  do
3:      $\hat{C}_k \leftarrow \text{ADDACTIVELINCUT}(\hat{y}_k, \hat{x}_k)$ 
4:      $(\hat{C}_k, \mathcal{B}_k) \leftarrow \text{ADDNONCONVEXLINCUTSANDPOINTS}(\hat{x}_k, \mathcal{B}_k)$ 
5:   return  $(\hat{C}, \mathcal{B})$ 

```

where $d_k \in \mathbb{R}^{n_k}$ is a search direction. It is solved by an OA approach with procedure `SOLVESUBPROBLEMWITHOA`(d_k). This procedure is described in Algorithm 3.2. It starts with initializing a temporary set of break-points \mathcal{D}_k . Then, the algorithm solves the following local OA sub-problem

$$\min d_k^T x_k \quad \text{s. t. } x_k \in \hat{X}_k. \quad (3.17)$$

Procedure `SOLVEOASUBPROBLEM` solves OA sub-problem (3.17) and returns a solution point \hat{x}_k . Similar to Algorithm 2.1, the algorithm iteratively projects the solution point \hat{x}_k , uses the projection point for generating cuts and break-points and solves the OA sub-problem again. The projection sub-problem is similar to (2.5), but the integer variables are fixed. Procedure `PROJECT` solves the following projection sub-problem with the fixed integers

$$\begin{aligned} \hat{y}_k &= \operatorname{argmin} \|x_k - \hat{x}_k\|^2 \\ \text{s. t. } x_k &\in G_k \cap L_k, \\ x_{ki} &= \hat{x}_{ki}, \quad i \in I_k, \end{aligned} \quad (3.18)$$

where \hat{x}_k is a solution point of (3.17).

After performing refinement of local OA sub-problem (3.17), the algorithm calls procedure `SOLVEFIXEDSUBNLP`. This procedure solves the following local NLP problem with *fixed integer variables*

$$\begin{aligned} \min c_k^T x_k \\ \text{s. t. } x_k &\in L_k \cap G_k, \\ x_{ki} &= \hat{x}_{ki}, \quad i \in I_k, \end{aligned} \quad (3.19)$$

where point \hat{x}_k is the solution of (3.17). The procedure returns a point x_k^* . If x_k^* is feasible, i.e. $x_k^* \in X_k$, then it is saved in the temporary set S_k . Set S_k is later used for initialization of the global list of break-points \mathcal{B} .

Algorithm 3.3 presents procedure `INITOA`, which computes initial OA \hat{C} and initial break-point set \mathcal{B} . First, the algorithm solves for each block three different sub-problems

Algorithm 3.2 Solving sub-problems using OA

```

1: function SOLVESUBPROBLEMWITHOA( $d_k, S_k, \hat{C}_k$ )
2:    $\mathcal{D}_k \leftarrow \{\underline{x}_k, \bar{x}_k\}$ 
3:    $\hat{x}_k \leftarrow \text{SOLVEOASUBPROBLEM}(\hat{C}_k, \mathcal{D}_k)$ 
4:   repeat
5:      $\hat{y}_k \leftarrow \text{PROJECT}(\hat{x}_k, G_k, L_k)$ 
6:      $(\hat{C}_k, \mathcal{D}_k) \leftarrow \text{ADDCUTSANDPOINTS}(\hat{x}_k, \hat{y}_k, \mathcal{D}_k)$ 
7:      $\hat{x}_k \leftarrow \text{SOLVEOASUBPROBLEM}(\hat{C}_k, \mathcal{D}_k)$ 
8:   until stopping criterion
9:    $x_k^* \leftarrow \text{SOLVEFIXEDSUBNLP}(\hat{x}_k)$ 
10:  if  $x_k^* \in X_k$  then
11:     $S_k \leftarrow S_k \cup \{x_k^*\}$ 
12:  return  $(\hat{C}_k, S_k)$ 

```

regarding different directions by calling procedure SOLVESUBPROBLEMWITHOA, depicted in Algorithm 3.2. These directions are: c_k , $\mathbf{1}$ and $-\mathbf{1}$, where $\mathbf{1}$ denotes a vector of ones. Then, Algorithm 3.3 uses a temporary set S_k of feasible points regarding non-linear feasible set X_k for initializing break-point set \mathcal{B} . For this purpose, procedure BOX(S_k) computes the smallest interval $[\alpha_k, \beta_k]$ containing set S_k .

Algorithm 3.3 OA initialization

```

1: function INITOA
2:  for  $k \in K$  do
3:     $\hat{C}_k \leftarrow \mathbb{R}^{n_k}$ 
4:     $S_k \leftarrow \emptyset$ 
5:    for  $d_k \in \{c_k, \mathbf{1}, -\mathbf{1}\}$  do
6:       $(\hat{C}_k, S_k) \leftarrow \text{SOLVESUBPROBLEMWITHOA}(d_k, S_k, \hat{C}_k)$ 
7:      if  $S_k \neq \emptyset$  then
8:         $[\alpha_k, \beta_k] \leftarrow \text{BOX}(S_k)$ 
9:         $\mathcal{B}_k \leftarrow \{\underline{x}_k, \alpha_k, \beta_k, \bar{x}_k\}$ 
10:     else
11:        $\mathcal{B}_k \leftarrow \{\underline{x}_k, \bar{x}_k\}$ 
12:  return  $(\hat{C}, \mathcal{B})$ 

```

3. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR NONCONVEX MINLP

Algorithm 3.4 OA-based local search

```

1: function OALocalSearch( $\hat{x}, \hat{C}, \mathcal{B}$ )
2:    $(\alpha, \beta) \leftarrow \text{GETADJACENTINTERVAL}(\hat{x}, \mathcal{B})$ 
3:    $\mathcal{D} \leftarrow \{\alpha, \beta\}$ 
4:   repeat
5:     for  $k \in K$  do
6:        $\hat{y}_k \leftarrow \text{PROJECT}(\hat{x}_k, G_k, L_k)$ 
7:        $(\hat{C}, \mathcal{D}) \leftarrow \text{ADDCUTSANDPOINTS}(\hat{x}, \hat{y}, \mathcal{D})$ 
8:        $\hat{x} \leftarrow \text{SOLVERESTRICTEDOA}([\alpha, \beta], \hat{C}, \mathcal{D})$ 
9:   until stopping criterion
10:   $\tilde{x} \leftarrow \text{SOLVEFIXEDNLP}(\hat{x})$ 
11:   $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{D}$ 
12:  return  $(\tilde{x}, \hat{C}, \mathcal{B})$ 

```

3.4 The local search

In this section, we present the algorithm for computing solution candidates of (1.2). Similar to Algorithm 2.3, it is based on solving an NLP master problem with fixed integer variables. The fixation point is provided by the solution point of the MIP-OA master problem. The algorithm operates with a limited set of break-points, in order to reduce the number of binary variables of MIP-OA (2.1).

Algorithm 3.4 presents the procedure for computing a solution candidate \tilde{x} of problem (1.2). As input, the procedure gets the solution point \hat{x} of OA master problem (2.1). In the beginning, procedure $\text{GETADJACENTINTERVAL}(\hat{x}, \mathcal{B})$ computes the smallest interval $[\alpha, \beta]$ containing \hat{x} using break-point set \mathcal{B} . Using this interval, the algorithm initializes a temporary break-point set \mathcal{D} , which is used later.

The algorithm iteratively projects a solution point \hat{x} by calling procedure PROJECT , which solves problem (3.18). Then, procedure ADDCUTSANDPOINTS adds cuts and break-points (Algorithm 3.1). At the end of each iteration, the algorithm solves the following restricted MIP-OA master problem

$$\begin{aligned}
 \hat{x} &= \operatorname{argmin} c^T x \\
 \text{s. t. } x &\in P \cap \hat{X} \cap [\alpha, \beta],
 \end{aligned} \tag{3.20}$$

where $[\alpha, \beta]$ is the interval computed by procedure $\text{GETADJACENTINTERVAL}$ at the beginning of the algorithm. We include the interval $[\alpha, \beta]$ in restricted MIP-OA master problem (3.20) to reduce the complexity of MIP model and to refine the OA within the

given target interval.

After performing iterations with restricted MIP-OA problem (3.20), procedure SOLVEFIXEDNLP computes a solution candidate \tilde{x} of problem (1.2) by solving NLP master problem (2.9) with fixed integer variables. At the end, temporary break-point set \mathcal{D} is added to the global set of break-points \mathcal{B} .

3.5 The main algorithm

In this section, we present the DECOA algorithm for solving original problem (1.2). The algorithm is similar to Algorithm 2.3, which is designed for convex MINLP problems.

Algorithm 3.5 presents the DECOA procedure to solve nonconvex MINLP problems. The algorithm starts by computing initial outer approximation \hat{C} and an initial break-point set \mathcal{B} by calling procedure INITOA, depicted in Algorithm 3.3. Also, it initializes an upper bound \bar{v} of the optimal value of (1.2). Before entering into the main loop, procedure SOLVEOA computes the first OA solution point \hat{x} by solving MIP-OA master problem (2.1).

In the main loop, the algorithm iteratively computes a solution candidate of problem (1.2) and refines the OA by adding cuts and break-points. A solution candidate \tilde{x} is computed by calling procedure OALOCALSEARCH, described in Algorithm 3.4. Using \tilde{x} , outer approximation \hat{C} is refined by calling procedure FIXANDREFINE, described in Algorithm 3.6. If solution point \tilde{x} of problem (2.9) improves the primal solution point, i.e. $\tilde{x} \in X$ and $c^T \tilde{x} < \bar{v}$, point \tilde{x} is a new primal solution of problem (1.2) and it is assigned to x^* . Moreover, the algorithm updates primal bound \bar{v} to $c^T x^*$. If the primal bound has been improved, the algorithm performs the procedure Optimization-based Bound Tightening (OBBT) [88] to reduce variable bounds. This strategy is based on minimizing and maximizing each single variable over the OA. Moreover, the feasible set of that problem also includes the optimality cut defined by primal bound \bar{v} . The OBBT MIP-OA problems are defined as follows

$$\begin{aligned} \min \quad & \pm x_{ki}, \\ & c^T x \leq \bar{v}, \\ & x \in \hat{X}, \quad i \in [n_k], \quad k \in K. \end{aligned} \tag{3.21}$$

Procedure TIGHTENBOUNDS solves $2n$ MIP-OA master problems (3.21), where n is the number of variables. In addition to it, it sets new upper and lower bounds for the variables, i.e. new lower bound \underline{x}_{ki} is set to the solution of (3.21) with ‘+’ in the objective and new upper bound \bar{x}_{ki} is set to the solution of (3.21) with ‘−’ in the

3. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR NONCONVEX MINLP

Algorithm 3.5 DECOA for nonconvex MINLP problems

```

1: function OASOLVE
2:    $(\widehat{C}, \mathcal{B}) \leftarrow \text{INITOA}$ 
3:    $\bar{v} \leftarrow \infty$ 
4:    $\hat{x} \leftarrow \text{SOLVEOA}(\widehat{C}, \mathcal{B})$ 
5:   repeat
6:      $(\tilde{x}, \widehat{C}, \mathcal{B}) \leftarrow \text{OALOCALSEARCH}(\hat{x}, \widehat{C}, \mathcal{B})$ 
7:      $(\widehat{C}, \mathcal{B}) \leftarrow \text{FIXANDREFINE}(\tilde{x}, \widehat{C}, \mathcal{B})$ 
8:     if  $\tilde{x} \in X$  and  $c^T \tilde{x} < \bar{v}$  then
9:        $x^* \leftarrow \tilde{x}$ ,  $\bar{v} \leftarrow c^T x^*$ 
10:      if  $\bar{v} - c^T \hat{x} < \epsilon$  then
11:        return  $(\hat{x}, x^*)$ 
12:       $(\underline{x}, \bar{x}) \leftarrow \text{TIGHTENBOUNDS}(x^*)$ 
13:       $\hat{x} \leftarrow \text{SOLVEOA}(\widehat{C}, \mathcal{B})$ 
14:      for  $k \in K$  do
15:         $\hat{y}_k \leftarrow \text{PROJECT}(\hat{x}_k)$ 
16:         $(\widehat{C}, \mathcal{B}) \leftarrow \text{ADDCUTSANDPOINTS}(\hat{x}, \hat{y}, \mathcal{B})$ 
17:    until  $\bar{v} - c^T \hat{x} < \epsilon$ 
18:    return  $(\hat{x}, x^*)$ 

```

objective. MIP problems (3.21) are difficult to solve, but they help to tighten variable bounds and, therefore, reduce set of break-points \mathcal{B} . By reducing break-point set \mathcal{B} , we reduce of binary variables in MIP-OA problem (2.1) and make it easier to solve.

In the end of each iteration of Algorithm 3.5, procedure SOLVEOA solves MIP-OA master problem (2.1). Using solution point \hat{x} of problem (2.1), procedure PROJECT solves problem (3.18) and returns \hat{y} as a solution point. Then, procedure ADDCUTSANDPOINTS, described in Algorithm 3.1, refines OA by generating cuts and break-points using points \hat{x} and \hat{y} . The algorithm terminates when a stopping criterion is fulfilled.

Procedure FIXANDREFINE, described in Algorithm 3.6, is similar to Algorithm 2.4 used in the DECOA algorithm for convex problems. As input, it gets a solution point \tilde{x} , which is a possible solution candidate of problem (1.2). At the beginning, the algorithm solves partly-fixed MIP-OA problem, defined in (2.10), where variables of all blocks are fixed, except for one block. Procedure SOLVEFIXOA solves the partly-fixed MIP-OA problem and returns \hat{x}_k as a result. Points \hat{x} and \tilde{x} are used to improve the OA by proce-

Algorithm 3.6 Fixation-based cut and break-point generation

```

1: function FIXANDREFINE( $\tilde{x}, \hat{C}, \mathcal{B}$ )
2:   for  $k \in K$  do
3:      $\hat{x}_k \leftarrow \text{SOLVEFIXOA}(\tilde{x}_k, \hat{C}_k, \mathcal{B}_k)$ 
4:      $(\hat{C}, \mathcal{B}) \leftarrow \text{ADDCUTSANDPOINTS}(\hat{x}, \tilde{x}, \mathcal{B})$ 
5:   repeat
6:     for  $k \in K$  do
7:        $\hat{x}_k \leftarrow \text{SOLVEFIXOA}(\tilde{x}_k, \hat{C}_k, \mathcal{B}_k)$ 
8:        $\hat{y}_k \leftarrow \text{PROJECT}(\hat{x}_k, G_k, L_k)$ 
9:        $(\hat{C}, \mathcal{B}) \leftarrow \text{ADDCUTSANDPOINTS}(\hat{x}, \hat{y}, \mathcal{B})$ 
10:  until stopping criterion
11:  return  $(\hat{C}, \mathcal{B})$ 

```

cedure `ADDCUTSANDPOINTS`, depicted in Algorithm 3.1. Then, the algorithm iteratively solves partly-fixed MIP-OA problem (2.10) and projection sub-problem (3.18). These operations are performed by procedures `SOLVEFIXOA` and `PROJECT`, respectively. At the end of each iteration, the algorithm calls procedure `ADDCUTSANDPOINTS` to refine the OA further. The algorithm stops when a stopping criterion is satisfied.

The convergence of Algorithm 3.5 has been proven in [93]. The objective value $c^T \hat{x}$, provided by the MIP-OA master problem in line 13 of Algorithm 3.5, converges to approximate optimal global solution v^* if finitely many break-points added to the set \mathcal{B} (see Theorem 1 in [93] for more details).

3.6 Numerical illustration

Algorithm 3.5 was implemented with Pyomo [48], an algebraic modelling language in Python, as a part of the MINLP solver DECOGO. Note that the sub-problems are not solved in parallel. For the experiments, we utilized Gurobi 8.0.1 [47] for solving MIP problems, which is free for an academic use. For solving LP and NLP problems, we used the open-source solver IPOPT 3.12.8 [106]. We present an experiment with one small-scale instance from MINLPLib [78] and compare the result with the BB solver SCIP 5.0 [44]. The experiment was performed using a computer with Intel Core i7-7820HQ 2.9 GHz CPU and 16 GB RAM. The automatic reformulation of MINLP into block-separable was performed using natural block structure identification, described in Section 1.3. For adding the break-points in procedure `ADDCUTSANDPOINTS`, we set

3. A DECOMPOSITION-BASED OUTER APPROXIMATION ALGORITHM FOR NONCONVEX MINLP

the parameter $\Delta = 4$.

The purpose of the experiment was to analyse the convergence speed of the objective value of MIP-OA problem (2.1), computed by Algorithm 3.5, to the primal bound \bar{v} . In this regard, we selected small-scale instance *ex3_1_1* from MINLPLib [78]. This instance consists of 8 variables and its global optimum value is 7049.2479. SCIP 5.0 [44] solved this instance to global optimality within 6.29 seconds. The experiment with Algorithm 3.5 has shown that the primal bound which corresponds to the global optimum can be computed already in the first iteration. However, the quality of OA is poor. The initial objective value of MIP-OA problem (2.1) was 2690.8968. Within 300 seconds and after 10 iterations of Algorithm 3.5, the objective value of MIP-OA problem (2.1), computed in line 13 of Algorithm 3.5 was 5987.361. Moreover, the MIP solver took a major part of the runtime of the entire algorithm, i.e. 213.17 seconds. This is due to the fact that the algorithm generated a lot of break-points for defining MIP-OA problem (2.1). After 10 iterations, the algorithm generated in total 38 break-points per variable, which corresponds to approximately 300 binary variables in MIP-OA problem (2.1). In further iterations, the MIP-OA problem becomes even more difficult to solve. In the same time, the objective value of MIP-OA problem had a little improvement.

For larger instances, the algorithm took even more time for solving MIP-OA problem (2.1). Clearly, due to the larger number of the nonconvex variables in larger instances, MIP-OA problem (2.1) already contained a lot of binary variables. The number of break-points could not be reduced, since, if there are not sufficient break-points, the quality of OA is unsatisfactory.

3.7 Conclusions

This chapter presents a multi-tree Decomposition-based Outer Approximation Algorithm for solving nonconvex block-separable MINLP problems (1.2). A piecewise nonconvex OA of the nonconvex feasible set is defined by piecewise DC underestimators of nonconvex functions. The presented approach to define piecewise nonconvex underestimators can be applied to any twice differentiable nonconvex function. The algorithm generates supporting hyperplanes for convexified functions and break-points to approximate quadratic functions by piecewise linear functions. Like in DECOA for convex problems, the supporting hyperplanes and break-points are generated by solving small sub-problems.

We focused on the question how to add partition points such that they define a tight OA and corresponding MIP-OA problem can be solved in a reasonable time.

An experiment with the small-scale example shows that the convergence speed of the algorithm is poor. The main reason of slow convergence speed is a computationally demanding MIP-OA master problem. This is due to the fact that, in order to define a tight piecewise nonconvex OA, one has to use a large number of break-points. When the MIP-OA master problem is defined with many break-points, then the complexity of this problem increases. When we use too few break-points, the quality of OA is insufficient.

The investigation of this chapter has been published in [84].

A Decomposition-based Inner and Outer Refinement Algorithm for nonconvex MINLP

In this chapter, we introduce a multi-tree Decomposition-based Inner and Outer Refinement (DIOR) Algorithm for solving nonconvex MINLP problems. The method is based on the so-called resource-constrained reformulation of problem (1.2), presented in Section 1.6. Using this approach, we compute an inner and outer approximation of convex relaxation (1.30) in the resource space by Column Generation (CG). Our research questions are: whether OA is suitable for solving nonconvex problems with many coupling constraints; what solution quality can be achieved when using IA.

4.1 Introduction

Over the last years, Column Generation (CG) has emerged as an efficient way to solve large-scale optimization problems [9, 26, 41]. For instance, Rapid Branching solves heuristically large-scale transport planning problems using CG [14]. For MINLP problems with a small duality gap, CG-based methods can be used to compute near-optimal solutions of problems with millions of variables [92]. However, we also consider MINLP problems with a large duality gap.

Like in previous chapters, we investigate the potential of decomposition in contrast to applying the BB algorithm. To do so, we develop two multi-tree Decomposition-based Inner and Outer Refinement (DIOR) algorithms. In both approaches, we reduce the dimension of the original problem using the concept of a resource-constrained program introduced in Section 1.6. Like DECOA, DIOR is a two-phase approach. In the first stage, both algorithms compute an *LP approximation* of the RCP regarding nondominated columns using two methods: (i) Subgradient method, (ii) Column Gen-

4. A DECOMPOSITION-BASED INNER AND OUTER REFINEMENT ALGORITHM FOR NONCONVEX MINLP

eration (CG). Our research question here is whether convergence of the CG procedure can be improved. In particular, we generate feasible points instead of optimal solution points of MINLP sub-problems.

In the second stage, both algorithms compute a *MIP approximation* of the RCP by adding disjunctive cuts. The approaches differ in the definition of a MIP approximation. The first algorithm uses a MIP outer approximation. It iteratively improves the OA by eliminating nondominated regions using a multi-objective-based line-search. Here, we focus on the question whether such approach can be applied to large-scale problems, in particular with many coupling constraints. The second algorithm uses a MIP inner approximation. To eliminate parts of possibly infeasible region, it adds disjunctive cuts by utilizing RCP sub-problem (1.16). The research question is whether the algorithm can compute high-quality solutions of the original problem given MIP inner approximation. Moreover, we answer the question what performance gains this algorithm may provide compared to other existing methods.

This chapter is organized as follows. Section 4.2 describes a Column Generation algorithm for computing initial inner and outer approximations. Section 4.3 presents a Decomposition-based Inner and Outer Refinement (DIOR) algorithm for computing a MIP outer approximation and its convergence proof. Section 4.4 explains a faster heuristic DIOR algorithm for computing a MIP inner approximation. Section 4.5 contains the numerical evaluation and shows the potential of the decomposition-based approach. Section 4.6 summarizes the findings of this chapter.

4.2 Column Generation

Column Generation (CG) is a decomposition approach for solving convex relaxation (1.31). The idea is to use the resource-constrained formulation of inner approximation (IA) defined by (1.35). Recall that IA problem (1.35) is defined by a set of columns $R_k, k \in K$. CG generates these columns by alternately computing the solution of LP-IA problem (1.36) and MINLP sub-problems over nonlinear feasible set X_k . Along with computation of the inner approximation, CG computes an outer approximation. More details on the CG algorithm are given in the next sections.

4.2.1 Initialization of LP-IA

Algorithm 4.1 computes initial columns $R_k, k \in K$ using a *subgradient method* [98] by maximizing the dual function $\mathcal{L}(\mu)$ of problem (1.2) regarding the global constraints:

$$\mathcal{L}(\mu) := \sum_{k \in K} \min_{y_k \in X_k} (1, \mu^T) A_k y_k - \mu^T b. \quad (4.1)$$

We compute a step length α^p by comparing the values of function $\mathcal{L}(\mu)$ defined in (4.1) at different iterations p of Algorithm 4.1 [98]:

$$\alpha^{p+1} = \begin{cases} 0.5\alpha^p & : \mathcal{L}(\mu^p) \leq \mathcal{L}(\mu^{p-1}), \\ 2\alpha^p & : \text{otherwise.} \end{cases} \quad (4.2)$$

The step size α^{p+1} at iteration $p + 1$ depends on the value of dual function $\mathcal{L}(\mu)$ in the preceding iterations. If the value of $\mathcal{L}(\mu^p)$ at iteration p is smaller than the value of $\mathcal{L}(\mu^{p-1})$ at iteration $p - 1$, then the step size α^{p+1} is decreased. In this situation, reduction of step size α is necessary, since we maximize dual function $\mathcal{L}(\mu)$. If the value of dual function $\mathcal{L}(\mu)$ increases, then step size α is enlarged. In this way, we try to increase the speed of convergence to the optimal value of dual function $\mathcal{L}(\mu)$. Procedure SOLVESUBPROBLEM(d) solves the following MINLP sub-problem for a given search direction $d \in \mathbb{R}^{m+1}$

$$\begin{aligned} y_k &= \operatorname{argmin} d^T A_k x \\ \text{s. t. } x &\in X_k \end{aligned} \quad (4.3)$$

and computes a reduced cost δ_k of the new point y_k . The columns $w_k = A_k y_k$ are added to the column set $R_k, k \in K$. Reduced cost δ_k is computed by taking the difference between the cost of new column w_k regarding direction d and the minimum cost of existing columns R_k regarding direction d , i.e.

$$\delta_k = d^T w_k - \min_{r_k \in R_k} d^T r_k. \quad (4.4)$$

Reduced cost δ_k is used to measure the impact of the procedure. If $\delta_k < 0$ for some $k \in K$, then column $A_k y_k$ may improve the objective value of (1.36). In the other case, if $\delta_k = 0, \forall k \in K$, the objective value of (1.36) cannot be changed [91] and the column generation procedure can be terminated. After computing minimizers y_k of (4.3) for $d = (1, \mu^T)$, one can easily compute the value of dual function (4.1). Note that if y_k is a global minimizer and d_k is a non-negative search direction, then $w_k = A_k y_k$ is a supported NDP (Section 1.6.4).

4. A DECOMPOSITION-BASED INNER AND OUTER REFINEMENT ALGORITHM FOR NONCONVEX MINLP

Algorithm 4.1 Initialization of LP-IA

```

1: function INITIA
2:    $R \leftarrow \emptyset, p \leftarrow 0, \mu^p \leftarrow 0, \alpha^p = 1$ 
3:   for  $k \in K$  do
4:      $(y_k, \delta_k) \leftarrow \text{SOLVESUBPROBLEM}(1, \vec{0}^T),$ 
5:      $R_k \leftarrow R_k \cup \{A_k y_k\}$ 
6:   repeat
7:      $p \leftarrow p + 1, \mu^p \leftarrow \mu^{p-1} + \alpha^p(Ay - b)$ 
8:     for  $k \in K$  do
9:        $(y_k, \delta_k) \leftarrow \text{SOLVESUBPROBLEM}(1, (\mu^p)^T)$ 
10:       $R_k \leftarrow R_k \cup \{A_k y_k\}$ 
11:   until  $p = p_{\max}$ 
12:   return  $R$ 

```

4.2.2 A Column Generation algorithm

Algorithm 4.2 describes a Column Generation algorithm for computing LP-IA (1.36). At the beginning of the algorithm, the feasible set of LP-IA (1.36) may be empty. Therefore, we define the LP-IA master problem with slack variables as follows

$$\begin{aligned}
& \min \sum_{k \in K} w_{k0}(z_k) + \theta \sum_{i \in M_1 \cup M_2} s_i^+ + s_i^- \\
& \text{s. t. } \sum_{k \in K} w_{ki}(z_k) \leq b_i + s_i^+, \quad i \in M_1, \\
& \quad \sum_{k \in K} w_{ki}(z_k) = b_i + s_i^+ - s_i^-, \quad i \in M_2, \\
& \quad z_k \in \Delta_{|R_k|}, \quad k \in K, \\
& \quad s_i^+, s_i^- \geq 0, \quad i \in M_1 \cup M_2,
\end{aligned} \tag{4.5}$$

where a penalty weight $\theta > 0$ is sufficiently large. Consider a vector s as all slacks together, i.e. $s = (s^+, s^-)$. Procedure $\text{SOLVESLACKMASTERPROBLEM}(R)$ solves problem (4.5). If the slack variables are nonzero, i.e. $s \neq 0$, procedure $\text{GETSLACKDIRECTIONS}$ computes a new search direction $d \in \mathbb{R}^m$ in the following way

$$d := \sum_{i \in \mathcal{J}} e_i, \quad \mathcal{J} = \{i \in M_1 \cup M_2 : \max(s_i^+, s_i^-) > 0.1 \max(s)\} \tag{4.6}$$

with $e_i \in \mathbb{R}^m$ the coordinate i unit vector. To eliminate the nonzero slack variables, sub-problems (4.3) are solved regarding the direction given by (4.6). This direction is

4.3 A DIOR algorithm for computing a MIP outer approximation

defined by the sum of the vectors that define left-hand side of global constraints with a relative large violation, i.e. corresponding slack value $s_i^+, s_i^-, i \in M_1 \cup M_2$ are larger than the 10 % of maximum slack value. The algorithm terminates when reduced cost δ_k (4.4) is nonnegative.

Algorithm 4.2 Column Generation

```

1: function COLGEN
2:    $R \leftarrow \text{INITIA}, \mathcal{M} \leftarrow \emptyset$ 
3:   repeat
4:      $(z, \mu, s) \leftarrow \text{SOLVESLACKMASTERPROBLEM}(R)$ 
5:      $\mathcal{M} \leftarrow \mathcal{M} \cup \mu$ 
6:     if  $s > 0$  then
7:        $d \leftarrow \text{GETSLACKDIRECTIONS}(s)$ 
8:       for  $k \in K$  do
9:          $(y_k, \delta_k) \leftarrow \text{SOLVESUBPROBLEM}(0, d^T)$ 
10:         $R_k \leftarrow R_k \cup \{A_k y_k\}$ 
11:       for  $k \in K$  do
12:          $(y_k, \delta_k) \leftarrow \text{SOLVESUBPROBLEM}(1, \mu^T)$ 
13:          $R_k \leftarrow R_k \cup \{A_k y_k\}$ 
14:   until  $\forall \delta_k \geq 0$ 
15:   return  $(z, R, \mathcal{M})$ 

```

4.3 A DIOR algorithm for computing a MIP outer approximation

In this section, we present a Decomposition-based Inner and Outer Refinement (DIOR) algorithm for computing an exact MIP outer approximation of resource-constrained problem (1.14). It consists of an LP phase and a MIP phase. The LP phase generates supported NDPs and computes convex relaxation (1.30). The MIP phase generates also non-supported NDPs. For the sake of simplicity, we consider only global inequality constraints, i.e. $M_2 = \emptyset$.

4. A DECOMPOSITION-BASED INNER AND OUTER REFINEMENT ALGORITHM FOR NONCONVEX MINLP

4.3.1 An LP outer approximation

An LP outer approximation (LP-OA) of problem (1.14) is defined by

$$\min \sum_{k \in K} w_{k0} \quad \text{s. t. } w \in H, \quad w_k \in P_k, \quad k \in K \quad (4.7)$$

with

$$P_k := \{w_k \in \mathbb{R}^{m+1} : (1, \mu^T) A_k y_k \leq (1, \mu^T) w_k, \quad \forall \mu \in \mathcal{M}\}, \quad (4.8)$$

where y_k is a solution of sub-problem (4.3) regarding a dual point μ and \mathcal{M} is a set of dual solution points computed by Algorithm 4.2. Note that P_k consists of a set of supporting hyperplanes of set W_k . In other words, set P_k is defined by valid linear constraints, since they are constructed using optimal solution point y_k of sub-problem (4.3) regarding dual direction $\mu \in \mathcal{M}$. Therefore,

$$W_k \subset P_k, \quad k \in K. \quad (4.9)$$

4.3.2 A MIP outer approximation

We construct a nonconvex outer approximation (OA) of W_k defined by polyhedral subdivision elements (cells) D_{ku} as follows

$$D_k := \bigcup_{u \in U_k} D_{ku} \supset W_k, \quad (4.10)$$

where U_k is an index set of subdivision elements. We define a cell D_{ku} , $u \in U_k$, $k \in K$ in the following way

$$D_{ku} = \{w \in \mathbb{R}^{m+1} : d_{kj}^T w \leq \beta_{kj}, \quad j \in J_{ku}\}, \quad (4.11)$$

where J_{ku} denotes an index set of constraints defining the u -th cell and $d_{kj} \in \mathbb{R}^{m+1}$, $\beta_{kj} \in \mathbb{R}$ are given.

A nonconvex MIP outer approximation (MIP-OA) problem of (1.14) is given by

$$\begin{aligned} \min \quad & \sum_{k \in K} w_{k0} \\ \text{s. t. } \quad & w \in H, \\ & w_k \in D_k, \quad w_k \in P_k, \quad k \in K. \end{aligned} \quad (4.12)$$

Note that (4.12) does not consider the integer constraints of original problem (1.2), since it is defined in the transformed feasible set, as in (1.12). Implicitly, problem

4.3 A DIOR algorithm for computing a MIP outer approximation

(4.12) contains only binary constraints which indicate whether cell $D_{ku}, u \in U_k$ is active or inactive. A MIP formulation of (4.12) is given by

$$\begin{aligned} \min \quad & \sum_{k \in K} w_{k0} \\ \text{s. t. } \quad & w \in H, \quad w_k \in P_k, \\ & d_{kj}^T w_k \leq M(1 - t_u) + \beta_{kj}, \quad j \in J_{ku}, \quad u \in U_k, \\ & t \in \{0, 1\}^{|U_k|} \cap \Delta_{|U_k|}, \quad k \in K, \end{aligned} \quad (4.13)$$

where d_{kj} and $\beta_{kj}, j \in J_{ku}$ describe the polyhedral set corresponding to cell D_{ku} described by (4.11). Cell D_{ku} is selected when corresponding binary variable $t_u = 1$. The “big” $M > 0$ should be sufficiently large.

4.3.3 Disjunctive cuts

A *p-disjunctive cut* removes a polyhedral set C_k from a cell D_{ku} , which is defined by p linear inequalities as follows

$$C_k = \{w \in \mathbb{R}^{m+1} : d_j^T w < \beta_j, \quad j \in [p]\}. \quad (4.14)$$

We generate a *cone cut* with respect to a vertex $v \in \mathbb{R}^{m+1}$, which we call an $|M_{1k}|$ -disjunctive cut, by removing the cone of dominated area

$$C_k(v_k) := \{w \in \mathbb{R}^{m+1} : \exists i \in M_{1k}, \quad w_i < v_i\}. \quad (4.15)$$

In the outer approximation, we use this concept to remove a cone of dominated area given an NDP v_k . MIP approximation (4.13) is refined by cutting off parts of solution \hat{w} by removing cones $C_k(v_k)$ that contain $\hat{w}_k \in C_k(v_k)$ for those blocks $k \in K$, where $\hat{w}_k \notin W_k$. In order to remove cone $C_k(v_k)$, cell D_{ku} is divided into $|M_{1k}|$ new overlapping cells

$$D_{ku_i} = \{w \in D_{ku} : w_i \geq v_i\}, \quad i \in M_{1k}. \quad (4.16)$$

4.3.4 Pareto line-search

We describe a line-search procedure that constructs a disjunctive cut in order to eliminate a solution (u, \hat{w}) of MIP-OA (4.13) by removing cone $C_k(v_k)$ defined in (4.15). Similar to (1.21), we consider the ideal point \underline{w}_k regarding the set of relevant resources M_{1k}

$$\underline{w}_{ki} = \min_{x_k \in X_k} A_{ki} x_k, \quad i \in M_{1k}. \quad (4.17)$$

4. A DECOMPOSITION-BASED INNER AND OUTER REFINEMENT ALGORITHM FOR NONCONVEX MINLP

Algorithm 4.3 Initialize DIOR

```

1: function INITDIOR
2:    $(\hat{z}, R, \mathcal{M}) \leftarrow \text{COLGEN}$                                 # inner refine
3:    $(u, D) \leftarrow \text{INITOA}(R, \mathcal{M})$                             # init cells
4:   return  $(\hat{z}, u, D, R)$ 

```

We compute a disjunctive cut by removing the largest cone $C_k(v_k)$, such that $C_k(v_k) \cap W_k = \emptyset$. v_k lies on the line connecting ideal point \underline{w}_k and solution point \hat{w}_k of MIP-OA problem (4.12). Given a line

$$v_i(\lambda) := \underline{w}_{ki} + \lambda_k(\hat{w}_{ki} - \underline{w}_{ki}), \quad i \in M_{1k}, \quad \lambda \in \mathbb{R}, \quad (4.18)$$

a line-search step size λ_k is computed by solving the following nonconvex *Pareto line-search sub-problem*

$$\begin{aligned}
\lambda_k = \min \quad & \lambda \\
\text{s. t.} \quad & A_{ki}x \leq v_i(\lambda), \quad i \in M_{1k}, \\
& x \in X_k, \quad \lambda \in \mathbb{R}.
\end{aligned} \quad (4.19)$$

Then, the cone tip of $C_k(v_k)$ is given componentwise by $v_{ki} = v_i(\lambda_k), i \in M_{1k}$.

4.3.5 DIOR using Pareto line-search

Algorithm 4.4 presents a DIOR algorithm for solving (1.2). It iteratively adds disjunctive cone cuts, computed by Pareto line-search sub-problem (4.19), to improve MIP-OA problem (4.13). The algorithm maintains a set of columns R_k and cells D_k on the local level and a set \mathcal{M} of dual vectors on the global level.

Algorithm 4.4 starts with calling procedure INITDIOR, depicted in Algorithm 4.3. The main goal of Algorithm 4.3 is to initialize an inner and outer approximation by performing CG procedure COLGEN, described in Algorithm 4.2. Algorithm 4.2 returns solution point \hat{z} of problem (1.36), set of columns R and set \mathcal{M} , which contains dual solutions of problem (1.36). In Algorithm 4.3, procedure INITOA(R, \mathcal{M}) initializes the cells D by $D_{k1} = P_k, k \in K$, where P_k is defined by local constraints of LP-OA problem (4.7) and returns the initial cell index u .

After performing initialization, in Algorithm 4.4, procedure IDEALPOINT computes an ideal point \underline{w}_k , defined in (4.17). Using it, the algorithm iteratively computes a cone tip (NDP) v_k and refines the OA by eliminating cones $C_k(v_k)$. Procedure PARETOLINESEARCH($\underline{w}_k, \hat{w}_k$) computes a possibly nonsupported NDP v_k (see Section 1.6.4) by solving problem (4.19). If $\lambda_k > 1$, then the algorithm calls procedure

4.3 A DIOR algorithm for computing a MIP outer approximation

Algorithm 4.4 DIOR for computing a MIP outer approximation

```

1: function DIOR1
2:    $(\hat{z}, u, D, R) \leftarrow \text{INITDIOR}, \hat{w} \leftarrow w(\hat{z})$  # LP-IA refine
3:   for  $k \in K$  do
4:      $\underline{w}_k \leftarrow \text{IDEALPOINT}$ , add cuts  $w_k \geq \underline{w}_k$  to  $D_k$ 
5:      $\lambda_k \leftarrow 1$ 
6:   repeat # MIP-OA refine
7:      $\tilde{w} \leftarrow \hat{w}$ 
8:     for  $k \in K$  do
9:        $(v_k, \lambda_k) \leftarrow \text{PARETOLINESEARCH}(\underline{w}_k, \hat{w}_k)$ 
10:      if  $\lambda_k > 1$  then
11:         $D_k \leftarrow \text{CONESUBDIV}(u_k, v_k, D_k)$  #  $D_k \leftarrow D_k \setminus C_k$ 
12:       $(u, \hat{w}) \leftarrow \text{SOLVEOUTERMIP}(D)$  # MIP-OA solution
13:    until  $\hat{w} - \tilde{w} \leq \epsilon$  or  $\lambda_k = 1, \forall k \in K$ 
14:  return  $\sum_{k \in K} w_{k0}$ 

```

$\text{CONESUBDIV}(u_k, v_k, D_k)$, in order to remove the cone $C_k(v_k)$ from set D_k by dividing D_k into new overlapping cells, defined in (4.16). If $\lambda_k = 1$, then solution \hat{w}_k is an NDP. In this case, subdivision is not performed. Procedure $\text{SOLVEOUTERMIP}(D)$ computes a point \hat{w}_k by solving MIP-OA problem (4.12). If, at some iteration, the algorithm does not remove the cone for all blocks, i.e. $\lambda_k = 1, \forall k \in K$, then the algorithm stops. The algorithm also terminates if an ϵ -tolerance on improvement of MIP-OA objective value is fulfilled, i.e. $\hat{w} - \tilde{w} \leq \epsilon$, where \tilde{w} is the solution of MIP-OA problem (4.12) in the previous iteration.

A reduced version of Algorithm 4.4 is illustrated with instances having one global constraint in [85]. In Section 4.5, we will provide a sketch of the idea of the cone points, outer solutions and outer approximation.

4.3.6 Proof of convergence

In this section, we prove that Algorithm 4.4 computes an ϵ -global optimum of problem (1.2) in finitely many iterations. Note that we assume $M_2 = \emptyset$, i.e. $m = |M_1|$.

Let

$$f(w) := \sum_{k \in K} w_{k0}. \quad (4.20)$$

Denote by $\hat{w}^p, \lambda^p, v^p$ a solution of MIP-OA (4.12), solution of Pareto line-search sub-

4. A DECOMPOSITION-BASED INNER AND OUTER REFINEMENT ALGORITHM FOR NONCONVEX MINLP

problem (4.19) and its corresponding cone tip, defined in (4.18), in iteration p , respectively. Furthermore, denote by $D_k^p \supset W_k, k \in K$ the outer approximation, which is refined by Algorithm 4.4 in iteration p by adding the cone cuts. In particular, we have

$$D_k^{p+1} = \begin{cases} D_k^p \setminus C_k(v_k^p), & \text{if } \lambda_k^p > 1, \\ D_k^p, & \text{if } \lambda_k^p = 1. \end{cases} \quad (4.21)$$

This process creates a sequence of enclosure sets

$$\widehat{W}^p := \prod_{k \in K} D_k^p \quad (4.22)$$

with the following property

$$\widehat{W}^0 \supset \dots \supset \widehat{W}^{p-1} \supset \widehat{W}^p \supset W. \quad (4.23)$$

In fact, it is sufficient to have \widehat{W}^p enclose W^* as Proposition 1.3 details. In order to prove the main convergence result, we present intermediate results in Lemmas 4.1–4.7. It is assumed that MIP-OA master problem (4.12) and line-search sub-problems (4.19) are solved to global optimality.

Lemma 4.1. Let \widehat{W}_k^{*p} be the Pareto front of \widehat{W}_k^p , i.e.

$$\widehat{W}_k^{*p} := \{w \in \widehat{W}_k^p : w \text{ is a NDP of } \min v \text{ s.t. } v \in \widehat{W}_k^p\}. \quad (4.24)$$

MIP-OA problem (4.12) is equivalent to

$$\min f(w) \quad \text{s.t. } w \in H, \quad w_k \in \widehat{W}_k^{*p}. \quad (4.25)$$

Proof. This can be proved exactly as Proposition 1.3. \square

For the sequel of the proof, we introduce the extended resource set as a complement of the dominated area

$$\overline{W}_k = \mathbb{R}^{m+1} \setminus \{w \in \mathbb{R}^{m+1} : \exists v \in W_k^*, \exists i \in M_{1k}, w_i < v_i\}. \quad (4.26)$$

The *extended Pareto front* is defined as

$$\overline{W}_k^* := \{w \in \overline{W}_k : w \text{ is a NDP of } \min v \text{ s.t. } v \in \overline{W}_k\}. \quad (4.27)$$

Notice that \overline{W}_k^* not only includes the Pareto front W_k^* , but also covers the gaps in the Pareto front, which are also sketched in Figure 1.5.

4.3 A DIOR algorithm for computing a MIP outer approximation

Lemma 4.2. The solution of problem

$$\min f(w) \quad \text{s.t. } w \in H, \quad w_k \in W_k, \quad k \in K \quad (4.28)$$

is attained at $w^* \in \overline{W}^*$, i.e. w^* is the solution of the following problem

$$\min f(w) \quad \text{s.t. } w \in H, \quad w_k \in \overline{W}_k^*, \quad k \in K. \quad (4.29)$$

Proof. This can be proven as in Proposition 1.3. Assume that part \hat{w}_k^* of optimal solution w^* does not belong to the extended Pareto front, i.e. $\hat{w}_k^* \notin \overline{W}_k^*$. This means $\exists \hat{w}_k \in W_k^*$ that dominates w_k^* , i.e. $\hat{w}_{ki} \leq w_{ki}^*$ for $i \in \{0\} \cup M_1$. Consider \hat{w} the corresponding solution, where in w^* the parts w_k^* are replaced by \hat{w}_k . As in the proof of Proposition 1.3, it follows that the optimum is attained at a NDP point $\hat{w} \in W^* \subseteq \overline{W}^*$. \square

Considering \overline{W}_k is relevant when we have a look at line-search (4.19). Now, focusing on this step, notice that if λ takes a value of 1, then the outer approximation sub-solution \hat{w}_k is feasible and optimal for this part of the master problem. So, if for all sub-problems $\lambda_k = 1$, then the algorithm converges.

Lemma 4.3. Let \hat{w}^p be an optimal solution of MIP-OA master problem (4.12). If after $p < \infty$ iterations of Algorithm 4.4, $\lambda_k^p = 1$ for all $k \in K$, then \hat{w}^p is an optimal solution of problem (1.14).

Proof. Since \hat{w}^p is an optimal solution of MIP-OA master problem (4.12), it is in $H \cap \widehat{W}^p$. From property (4.23), \widehat{W}^p includes W . Since $\hat{w}_k^p \in \widehat{W}_k^{*p}$ from Lemma 4.1, it follows $\text{int}[\underline{w}_k, \hat{w}_k^p] \cap \widehat{W}_k^p = \emptyset$, and hence $\lambda_k \geq 1$ for all $k \in K$. If $\lambda_k^p = 1$, then $v_k \in [\underline{w}_k, \hat{w}_k^p]$. Therefore, no cone $C_k(v_k)$ with $v_{ki} > \hat{w}_{ki}^p$ for $i \in \{0, \dots, m\}$ and $\hat{w}_{ki}^p > \underline{w}_{ki}$ has to be removed from \widehat{W}_k^p . Hence, $\hat{w}_k^p \in \overline{W}_k^*$ for all $k \in K$. From Lemma 4.2 it follows that \hat{w}^p minimizes the objective function within $H \cap W$. Since $\hat{w}^p \in H$, it follows that it is also an optimal solution of (1.14). \square

Lemma 4.4. If $\lambda_k^p \neq 1$ for some $k \in K$, Algorithm 4.4 excludes \hat{w}^p from set \widehat{W}^{p+1} , i.e. $\hat{w}^p \notin \widehat{W}^{p+1}$.

Proof. If $\lambda_k^p \neq 1$, then $\lambda_k^p > 1$ from Lemma 4.1 and $C_k(v_k^p)$ is removed from \widehat{W}^p . Since $\lambda_k^p > 1$, then $\exists i \in \{0, \dots, m\}$ with $v_{ki}^p > \hat{w}_{ki}^p$. Hence, $\hat{w}_k^p \in C_k(v_k^p)$ and $\hat{w}^p \notin \widehat{W}^{p+1}$. \square

In Lemma 4.5, we show that if Algorithm 4.4 does not stop in a finite number of iterations, the sequence of primal solution points contains at least one convergent subsequence $\{\hat{w}^{p_j}\}_{j=1}^\infty$, where

$$\{p_1, p_2, \dots\} \subseteq \{1, 2, \dots\} \quad \text{and} \quad \{\hat{w}^{p_j}\}_{j=1}^\infty \subseteq \{\hat{w}^p\}_{p=1}^\infty.$$

4. A DECOMPOSITION-BASED INNER AND OUTER REFINEMENT ALGORITHM FOR NONCONVEX MINLP

Since subsequence $\{\hat{w}^{p_j}\}_{j=1}^\infty$ is convergent, there exists a limit $\lim_{j \rightarrow \infty} \hat{w}^{p_j} = w^*$. In Lemmas 4.6 and 4.7, we show that w^* is in the extended Pareto front \overline{W}^* and therefore an optimal solution of (1.14), where $\overline{W}^* := \prod_{k \in K} \overline{W}_k^*$.

Lemma 4.5. If Algorithm 4.4 does not stop in a finite number of iterations, it generates a convergent subsequence $\{\hat{w}^{p_j}\}_{j=1}^\infty$.

Proof. Since the algorithm has not terminated, for all $p = 1, 2, \dots$ there exists $k \in K$ such that $\lambda_k^p > 1$. Therefore, all points in the sequence $\{\hat{w}^p\}_{p=1}^\infty$ are distinct as shown in Lemma 4.4. Recall that the Bolzano-Weierstrass Theorem states that each bounded sequence in \mathbb{R}^n has a convergent subsequence. Since $\{\hat{w}^p\}_{p=1}^\infty$ contains an infinite number of different points, and all are in a compact set and MIP-OA is bounded, according to the Bolzano-Weierstrass Theorem, the sequence contains a convergent subsequence. \square

Lemma 4.6. The limit w_k^* for any convergent subsequence $\{\hat{w}^{p_j}\}_{j=1}^\infty$ generated in Algorithm 4.4 belongs to \overline{W}_k^* .

Proof. Let $\hat{w}_k^{p_j}$ and $\hat{w}_k^{p_{j+1}}$ be points from sequence $\{\hat{w}_k^{p_j}\}_{j=1}^\infty$. From Lemma 4.4 follows that in each iteration a cone $C_k(v_k^p)$ with $v_k^p = \underline{w}_k + \lambda_k^p(\hat{w}_k^p - \underline{w}_k)$ for some $k \in K$ is removed. The cone is also removed for next iterates p_j , i.e. $\exists i, \hat{w}_{ki}^p \geq v_{ki}^{p_j}$. This means

$$\exists i \in \{0, \dots, m\}, \hat{w}_{ki}^{p_{j+1}} - \underline{w}_i \geq \lambda_k^{p_j}(\hat{w}_{ki}^{p_j} - \underline{w}_i). \quad (4.30)$$

Assume that $\lambda_k^{p_j}$ does not converge to 1 and there is a value $\tau > 1$, such that in each iterate $\lambda_k^{p_j} > \tau > 1$. This leads to a contradiction, because the iterates v_k^p are in the bounded set D_k^0 . From the proof of Lemma 4.3, $\lambda_k^{p_j} \geq 1$. Hence, $\lambda_k^{p_j} \rightarrow 1$ and $|\hat{w}_k^{p_j} - v_k^{p_j}| \rightarrow 0$. This implies $\lim_{j \rightarrow \infty} \hat{w}_k^{p_j} \in \overline{W}_k^*$. \square

Lemma 4.7. The limit point of a convergent subsequence generated in Algorithm 4.4 is a global minimum point of (1.14).

Proof. Because each set \widehat{W}^p is an outer approximation of the feasible set W , $f(\hat{w}^{p_j})$ gives a lower bound on the optimal value of the objective function. Since sequence $\{f(\hat{w}^{p_j})\}_{j=1}^\infty$ is nondecreasing due to property (4.23) and the objective function is continuous, we get $\lim_{j \rightarrow \infty} f(\hat{w}^{p_j}) = f(w^*)$. According to Lemma 4.6, limit point w_k^* is within the set \overline{W}^* . From Lemma 4.2 follows that w^* minimizes the objective function within $H \cap W$. Because $w^* \in H$, it is also an optimal solution of (1.14). \square

4.4 A DIOR algorithm for computing a MIP inner approximation

Since Lemmas 4.6 and 4.7 apply to all convergent subsequences generated by MIP-OA master problems (4.12), any limit point of such sequence is a global optimum. We summarize the convergence results in the following theorem.

Theorem 4.8. *Algorithm 4.4 either finds a global optimum of (1.14) in a finite number of iterations or generates a sequence $\{\hat{w}^{p_j}\}_{j=1}^{\infty}$ converging to a global optimum.*

Proof. Suppose the algorithm stops in a finite number of iterations. Then, the last solution of MIP-OA master problem (4.12) satisfies all constraints and according to Lemma 4.3 it is a global optimum of (1.14). In case the algorithm does not stop in a finite number of iterations, it generates a sequence converging to a global optimum of (1.14) according to Lemmas 4.5 and 4.7. \square

4.4 A DIOR algorithm for computing a MIP inner approximation

Motivated by the Rapid Branching approach [14], we present in this section a heuristic DIOR algorithm for computing a MIP inner approximation of resource-constrained problem (1.14). The goal of the algorithm is to compute a high-quality primal solution point using the resources obtained by the MIP inner approximation. The idea is to cut off iteratively low-dimensional faces of $\text{conv}(R \cap H)$ containing the solution \hat{w} of a MIP master problem.

4.4.1 MIP inner approximation

Consider a partition of resource space $[\underline{w}, \bar{w}]$ using polyhedral partition elements (cells) D_{ku} , i.e.

$$[\underline{w}, \bar{w}] = \bigcup_{u \in U_k} D_{ku}, \quad \text{int}(D_{ku}) \cap \text{int}(D_{k\nu}) = \emptyset, \quad \forall u, \nu \in U_k. \quad (4.31)$$

4. A DECOMPOSITION-BASED INNER AND OUTER REFINEMENT ALGORITHM FOR NONCONVEX MINLP

Implicitly, this partition also divides set of columns R_k into subsets $R_{ku} := R_k \cap D_{ku}$. A MIP inner approximation (MIP-IA) with slacks is defined by

$$\begin{aligned}
& \min \sum_{k \in K} w_{k0} + \theta \sum_{i \in M_1 \cup M_2} s_i^+ + s_i^- \\
& \text{s. t. } \sum_{k \in K} w_{ki} \leq b_i + s_i^+, \quad i \in M_1, \\
& \quad \sum_{k \in K} w_{ki} = b_i + s_i^+ - s_i^-, \quad i \in M_2, \\
& \quad w_k \in \bigcup_{u \in U_k} \text{conv}(R_{ku}), \quad k \in K, \\
& \quad s_i^+, s_i^- \geq 0, \quad i \in M_1 \cup M_2.
\end{aligned} \tag{4.32}$$

A MIP formulation of (4.32) is given by

$$\begin{aligned}
& \min \sum_{k \in K} w_{k0}(z_k) + \theta \sum_{i \in M_1 \cup M_2} s_i^+ + s_i^- \\
& \text{s. t. } \sum_{k \in K} w_{ki}(z_k) \leq b_i + s_i^+, \quad i \in M_1, \\
& \quad \sum_{k \in K} w_{ki}(z_k) = b_i + s_i^+ - s_i^-, \quad i \in M_2, \\
& \quad z_k \in \Delta_{|R_k|}, \quad t \in \Delta_{|U_k|} \cap \{0, 1\}^{|U_k|}, \\
& \quad \sum_{j \notin [R_{ku}]} z_{kj} \leq 1 - t_u, \quad u \in U_k, \quad k \in K, \\
& \quad s_i^+, s_i^- \geq 0, \quad i \in M_1 \cup M_2,
\end{aligned} \tag{4.33}$$

where $[R_{ku}] \subset [R_k]$ denotes indices of columns R_{ku} and $U_k, k \in K$ denotes an index set for cells. Note that replacing $\text{conv}(R_{ku})$ in (4.32) by $\text{conv}(W_k \cap D_{ku_k})$ defines a lower bounding program of MINLP (1.2). By performing Column Generation regarding cells D_{ku} , the optimum value of (4.32) converges to the optimum value of this lower bounding program.

4.4.2 Refinement of MIP-IA

We refine MIP-IA (4.33) by adding an *inner disjunctive cut* defined by subdividing a cell D_{ku_k} into sub-cells D_{kv} , $v \in V_k(u_k)$ such that

$$D_{ku_k} = \bigcup_{v \in V_k(u_k)} D_{kv}, \quad \text{int}(D_{kv}) \cap \text{int}(D_{kw}) = \emptyset, \quad \forall v, w \in V_k(u_k) \tag{4.34}$$

4.4 A DIOR algorithm for computing a MIP inner approximation

and replacing $\text{conv}(R_{ku_k})$ by $\bigcup_{v \in V_k(u_k)} \text{conv}(R_{ku_k} \cap D_{kv})$. In order to increase the optimum value of (4.32), it is necessary to cut off $w_k(\hat{z}_k)$ for some $k \in K$, where \hat{z} is a solution of MIP-IA (4.33). This is equivalent to

$$w_k(\hat{z}_k) \notin \text{conv}(R_{ku_k} \cap D_{kv}), \quad \forall v \in V_k(u_k). \quad (4.35)$$

Denote by $\hat{R}_k \subseteq R_k$ a set of supporting columns with $w_k(\hat{z}_k) \in \text{int}(\text{conv}(\hat{R}_k))$. We define the sub-cell D_{kv} such that it eliminates one supporting column from set \hat{R}_k , i.e. $\hat{R}_k \not\subseteq D_{kv}, \forall v \in V_k(u_k)$. For that, we set point $w_k(\hat{z})$ to be a vertex of $D_{kv}, \forall v \in V_k(u_k)$. Since $w_k(\hat{z}_k) \in \text{int}(\text{conv}(\hat{R}_k))$ and $w_k(\hat{z}_k) \in \text{vert}(D_{kv}), \forall v \in V_k(u_k)$, it cannot be expressed as a convex combination of points in $\hat{R}_k \cap D_{kv}$, i.e. (4.35) holds.

Algorithm 4.8 presents a procedure for refining MIP-IA by adding disjunctive cuts. It splits iteratively active cell D_{ku_k} with index u_k into sub-cells $D_{kv}, v \in V_k(u_k)$. Note that the indices of active cells u are provided by MIP-IA problem (4.33). In order to prevent, that cell D_{ku_k} is subdivided several times, the algorithm stores the index set of new cells $V_k = V_k(u_k)$. The algorithm subdivides the cell D_{ku_k} only if $V_k = \emptyset$.

Denote a set of refined blocks by $\tilde{K} \subset K$. The algorithm creates *paths* for the refined sub-cells $D_{ku_k}, k \in \tilde{K}$. The triple (u, \hat{z}, \tilde{K}) represents the path and it is saved in set \mathcal{P} . The elements of the triple are:

1. Index u containing indices of cells $D_{ku_k}, k \in K$. It is obtained by solving MIP-IA problem (4.33) or after performing partition of a cell.
2. Solution point \hat{z} corresponding to cells with index u . It is computed by MIP-IA problem (4.33) or by restricted LP-IA problem (4.43) with the fixed cells.
3. Set of refined blocks $\tilde{K} \subset K$.

As input arguments, Algorithm 4.8 obtains the indices of active cells u and corresponding solution point \hat{z} , which are computed by MIP-IA (4.33). The algorithm initializes set of paths \mathcal{P} using input arguments u and \hat{z} and setting $\tilde{K} = \emptyset$. In the beginning of each iteration, the algorithm selects the path with a minimum lower bound $\underline{\nu}(\hat{z})$ regarding solution point \hat{z} contained in the path. The lower bound is defined by

$$\underline{\nu}(\hat{z}) := \sum_{k \in K} w_{k0}(\hat{z}_k). \quad (4.36)$$

After selection of the particular path, it is removed from set \mathcal{P} . Then, the algorithm performs three steps:

1. Selects block k , where the cell u_k to be refined. u are indices contained in the selected path.

4. A DECOMPOSITION-BASED INNER AND OUTER REFINEMENT ALGORITHM FOR NONCONVEX MINLP

Algorithm 4.5 Select block for refinement

```

1: function SELECTBLOCK( $u, \hat{z}, D, R, V, \tilde{K}$ )
2:   for  $k \in K \setminus \tilde{K}$  do
3:      $\hat{R}_k \leftarrow \text{GETSUPPORTCOLUMNS}(u_k, \hat{z}_k, D_k, R_k)$ 
4:      $\hat{K} \leftarrow \{k \in K \setminus \tilde{K} : |\hat{R}_k| \geq 2 \wedge V_k = \emptyset\}$            # selects unrefined block
5:     if  $\hat{K} \neq \emptyset$  then
6:       repeat
7:          $k \leftarrow \text{argmin}\{|\max_{j \in [\hat{R}_k]} \hat{z}_{\ell j} - 0.5| : \ell \in \hat{K}\}$ 
8:          $(\tilde{y}_k, \tilde{s}_k) \leftarrow \text{SOLVERESPROJECTSUBPROBLEM}(u_k, \hat{z}_k)$ 
9:          $R_k \leftarrow R_k \cup \{A_k \tilde{y}_k\}$ 
10:        if  $\tilde{s}_k = 0$  then
11:           $\hat{K} \leftarrow \hat{K} \setminus \{k\}$ 
12:        else
13:          return  $(k, A_k \tilde{y}_k + \tilde{s}_k, \hat{R}_k)$            #  $s_k > 0$ , stop
14:        until  $\hat{K} = \emptyset$ 
15:   return  $(\emptyset, \emptyset, \emptyset)$            # no block to refine

```

2. Refines cell D_{ku_k} .
3. Generates columns and paths regarding new cells $D_{kv}, v \in V_k(u_k)$.

Algorithm 4.5 presents the first step, where it determines a block $k \in K \setminus \tilde{K}$ for subdivision. Procedure GETSUPPORTCOLUMNS(u_k, \hat{z}_k, D_k, R_k) computes set of supporting columns \hat{R}_k for each block, except of blocks in \tilde{K} , finds p largest positive values of \hat{z}_k , i.e. $\hat{z}_{k1} \geq \hat{z}_{k2} \geq \dots \geq \hat{z}_{kp}$ and sets

$$\hat{R}_k = \{r_{kj} : j \in [p], \hat{z}_{kj} > 0, r_{kj} \in R_k\}. \quad (4.37)$$

Moreover, the procedure eliminates redundant columns $r_{kj} \in \hat{R}_k$, which are convex combinations of other columns of \hat{R}_k , i.e. $\text{conv}(\hat{R}_k) = \text{conv}(\hat{R}_k \setminus \{r_{kj}\})$ and $|\hat{R}_k| \leq |M_{1k}| + |M_{2k}|$. Then, using the set of new cells V , Algorithm 4.5 computes a set of unrefined blocks \hat{K} that used later to select a block for subdivision. If $\hat{K} = \emptyset$, then there is no block to select. In this case, Algorithm 4.5 terminates.

If $\hat{K} \neq \emptyset$, then the algorithm performs an iterative procedure in order to select a block for refinement. Since the relative distance of $w_k(\hat{z}_k)$ to a column $r_{kj} \in \hat{R}_k$ is related to $1 - \hat{z}_{kj}$, a block with small $|\max_{j \in [\hat{R}_k]} \hat{z}_{\ell j} - 0.5|$ is selected. In order to check whether $w_k(\hat{z}_k)$ is infeasible, the algorithm solves the following resource-constrained

4.4 A DIOR algorithm for computing a MIP inner approximation

Algorithm 4.6 Refinement of the cell D_{ku_k}

- 1: **function** REFINE($u_k, \tilde{w}_k, \hat{z}_k, D_k, \hat{R}_k, \tilde{K}$)
 - 2: $\tilde{K} \leftarrow \tilde{K} \cup \{k\}$
 - 3: $\eta_k \leftarrow \text{COMPUTEDISJCUTS}(\hat{z}_k, \tilde{w}_k, \hat{R}_k)$
 - 4: $(V_k, D_k) \leftarrow \text{SUBDIVIDE}(u_k, \eta_k, \hat{z}_k, D_k)$
 - 5: **return** (V_k, D_k, \tilde{K})
-

projection sub-problem with slacks

$$\begin{aligned}
 (\tilde{y}_k, \tilde{s}_k) = \operatorname{argmin} \quad & \sum_{i \in M_{1k} \cup M_{2k}} s_{ki}^+ + s_{ki}^- \\
 \text{s. t.} \quad & A_{ki}x_k \leq w_{ki}(\hat{z}_k) + s_{ki}^+, \quad i \in M_{1k}, \\
 & A_{ki}x_k = w_{ki}(\hat{z}_k) + s_{ki}^+ - s_{ki}^-, \quad i \in M_{2k}, \\
 & s_{ki}^+, s_{ki}^- \geq 0, \quad i \in M_{1k} \cup M_{2k}, \\
 & x_k \in X_k, \quad A_k x_k \in D_{ku_k},
 \end{aligned} \tag{4.38}$$

where $\tilde{s}_k = (\tilde{s}_k^+, \tilde{s}_k^-)$. Procedure SOLVERESPROJECTSUBPROBLEM(u_k, \hat{z}_k) solves sub-problem (4.38). Projection sub-problem (4.38) is similar to RCP sub-problem (1.16). If $\tilde{s}_k \neq 0$, then $w_k(\hat{z}_k) \notin W_k$. In this case, the algorithm stops and returns corresponding block index k as a result. If $\tilde{s}_k = 0$, then the algorithm removes selected block index k from set \hat{K} and starts a new iteration. After removing block index k from set \hat{K} , this set might be empty. In this case, the algorithm also terminates and no result is returned.

Algorithm 4.6 describes the second step of Algorithm 4.8, i.e. the refinement step. At this stage, the algorithm divides the cell D_{ku_k} into new overlapping sub-cells

$$D_{ku_j} = \{w \in D_{ku_k} : \eta_{kji}^T(w - w_k(\hat{z}_k)) \geq 0, \quad i \in [\hat{R}_k] \setminus \{j\}\}, \quad u_j \in V_k(u_k). \tag{4.39}$$

New cells D_{ku_j} are defined by $|\hat{R}_k| - 1$ cut directions η_{kji} . These cuts separate column $r_{kj} \in \hat{R}_k$ and fulfill the condition (4.35).

At the beginning, Algorithm 4.6 adds selected block k to set of refined blocks \tilde{K} . In the next step, procedure COMPUTEDISJCUTS($\hat{z}_k, \tilde{w}_k, \hat{R}_k$) computes cut directions η_{kji} of new cells D_{ku_j} defined in (4.39). This procedure solves the following system of linear

4. A DECOMPOSITION-BASED INNER AND OUTER REFINEMENT ALGORITHM FOR NONCONVEX MINLP

Algorithm 4.7 CG for sub-paths

```

1: function ADDCOLSPATHS( $u_k, \tilde{w}_k, V_k, D, R, \mathcal{P}, \tilde{K}$ )
2:   for  $v \in V_k$  do
3:      $u_k \leftarrow v$ 
4:      $d_k \leftarrow \text{COMPUTESearchDirection}(\tilde{w}_k, D_{ku_k})$ 
5:      $y_k \leftarrow \text{SOLVELAGSUBPROBLEM}(d_k, D_{ku_k})$ 
6:      $R_k \leftarrow R_k \cup \{A_k y_k\}$ 
7:      $(\hat{z}, \mu) \leftarrow \text{SOLVERESTRICTIA}(u, D, R)$ 
8:     for  $\ell \in K$  do
9:        $y_\ell \leftarrow \text{SOLVELAGSUBPROBLEM}((1, \mu^T), D_{\ell u_\ell})$ 
10:       $R_\ell \leftarrow R_\ell \cup \{A_\ell y_\ell\}$ 
11:      if  $|\tilde{K}| < |K|$  then
12:         $\hat{z} \leftarrow \text{SOLVERESTRICTIA}(u, D, R)$ 
13:         $\mathcal{P} \leftarrow \mathcal{P} \cup \{(u, \hat{z}, \tilde{K})\}$  # add new path
14:   return  $(\mathcal{P}, R)$ 

```

equations

$$\begin{aligned}
\eta_{kji}^T(r - w_k(\hat{z}_k)) &= 0, \\
\eta_{kji} &\in \text{span}\{(r - \tilde{w}_k)\}_{r \in \hat{R}_k}, \\
r &\in \hat{R}_k \cup \{\tilde{w}_k\} \setminus \{r_{kj}, r_{ki}\}, \\
j &\in [\hat{R}_k], \quad i \in [\hat{R}_k] \setminus \{j\},
\end{aligned} \tag{4.40}$$

where $\eta_{kji} \in \text{span}\{(r - \tilde{w}_k)\}_{r \in \hat{R}_k}$ means that vector η_{kji} is a linear combination of vectors $(r - \tilde{w}_k), r \in \hat{R}_k$. If $\eta_{kji}^T(r_{kj} - w_k(\hat{z}_k)) \geq 0$, then η_{kji} is multiplied by -1 . Although procedure GETSUPPORTCOLUMNS removes redundant columns, there is no guarantee that system (4.40) is solvable. In order to be able always to compute the cut η_{kji} , the procedure constructs it by computing a basis of the null space of (4.40). Finally, procedure SUBDIVIDE(η_k, \hat{z}_k, D_k) splits the cell D_{ku_k} into new cells $D_{kv}, v \in V_k(u_k)$, defined in (4.39), and updates the index set of cells U_k , i.e. $U_k \leftarrow U_k \setminus \{u_k\} \cup V_k(u_k)$.

In the final, third step, Algorithm 4.8 generates new columns and paths regarding new cells $D_{kv}, v \in V(u_k)$. This step is described in Algorithm 4.7. First, procedure COMPUTESearchDirection(\tilde{w}_k, D_{ku_k}) computes a search direction d_k as follows

$$d_k = \tilde{w}_k - \hat{r}_k, \quad \hat{r}_k \notin D_{ku_k}. \tag{4.41}$$

Then, procedure SOLVELAGSUBPROBLEM(d_k, D_{ku_k}) uses a search direction d_k to perform a restricted CG. This procedure solves the following restricted sub-problem re-

4.4 A DIOR algorithm for computing a MIP inner approximation

Algorithm 4.8 Inner refinement

```

1: function INNERREFINE( $u, \hat{z}, D, R$ )
2:    $V \leftarrow \emptyset, \mathcal{P} \leftarrow \{(u, \hat{z}, \emptyset)\}$ 
3:   repeat
4:      $(\hat{z}, u, \tilde{K}) \leftarrow \operatorname{argmin}\{\underline{\nu}(\hat{z}) : (\hat{z}, u, \tilde{K}) \in \mathcal{P}\}$ 
5:      $\mathcal{P} \leftarrow \mathcal{P} \setminus \{(u, \hat{z}, \tilde{K})\}$ 
6:      $(k, \tilde{w}_k, \hat{R}_k) \leftarrow \operatorname{SELECTBLOCK}(u, \hat{z}, D, R, V, \tilde{K})$  # 1. select new block
7:     if  $k \neq \emptyset$  then
8:        $(V_k, D_k, \tilde{K}) \leftarrow \operatorname{REFINE}(u_k, \tilde{w}_k, \hat{z}_k, D_k, \hat{R}_k, \tilde{K})$  # 2. refinement
9:        $(\mathcal{P}, R) \leftarrow \operatorname{ADDCOLSPATHS}(u_k, \tilde{w}_k, V_k, D, R, \mathcal{P}, \tilde{K})$  # 3. add new paths
10:    until  $\mathcal{P} = \emptyset$  or stopping criterion
11:  return  $(V, D, R)$ 

```

garding cell D_{ku_k}

$$\begin{aligned}
 y_k &= \operatorname{argmin} d_k^T A_k x_k \\
 \text{s. t. } x_k &\in X_k, \\
 A_k x_k &\in D_{ku_k}.
 \end{aligned} \tag{4.42}$$

After obtaining the new column $A_k y_k$ by solving problem (4.42), the algorithm computes a new dual solution μ to perform a restricted CG for all blocks. Procedure $\operatorname{SOLVERELECTIA}(u, D, R)$ computes a dual solution μ of the following restricted LP-IA problem regarding fixed cells

$$\begin{aligned}
 \min \quad & \sum_{k \in K} w_{k0}(z_k) \\
 \text{s. t. } \quad & w(z) \in H, \\
 & z_k \in \Delta_{|R_k|}, \quad z_{kj} = 0, \quad j \notin [R_k \cap D_{ku_k}], \quad k \in K.
 \end{aligned} \tag{4.43}$$

LP-IA problem (4.43) is defined regarding the columns that belong only to the cells $D_{ku_k}, k \in K$. Since some columns might be missing in the cells $D_{ku_k}, k \in K$, the algorithm generates more columns for all blocks regarding these cells by solving restricted sub-problems (4.42) regarding dual solution μ of problem (4.43). In the end, the algorithm computes a solution point \hat{z} of restricted LP-IA problem (4.43) and adds a new path to the set \mathcal{P} .

As mentioned before, Algorithm 4.8 iteratively selects a path and refines it. The algorithm may not select a new block for refinement. In this case, it does not generate new paths. Also, the algorithm may not generate new paths even after performing the

4. A DECOMPOSITION-BASED INNER AND OUTER REFINEMENT ALGORITHM FOR NONCONVEX MINLP

Algorithm 4.9 The heuristic DIOR for computing a MIP inner approximation

```

1: function DIOR2
2:    $(\hat{z}, u, D, R) \leftarrow \text{INITDIOR}, \quad x^* \leftarrow \emptyset$                                 # LP-IA refine
3:   repeat
4:      $(V, D, R) \leftarrow \text{INNERREFINE}(u, \hat{z}, D, R)$                                 # MIP-IA refine
5:     if  $V \neq \emptyset$  then
6:        $(u, \hat{z}) \leftarrow \text{SOLVEINNERMIP}(R, D)$                                 # MIP-IA solution
7:   until  $V = \emptyset$  or stopping criterion
8:    $\underline{v} \leftarrow \sum_{k \in K} \hat{w}_{k0}(\hat{z}_k)$                                 # estimated lower bound
9:   for  $k \in K$  do
10:     $y_k \leftarrow \text{SOLVERESPROJECTSUBPROBLEM}(u_k, \hat{z}_k)$                                 # partial sol.
11:     $\tilde{x} \leftarrow \text{SOLVEFIXEDNLP}(y)$                                 # solution candidate
12:    if  $\tilde{x} \in X$  then
13:       $x^* \leftarrow \tilde{x}$ 
14:   return  $(\underline{v}, x^*)$ 

```

refinement step, because a set of refined blocks \tilde{K} may contain all blocks from set K , i.e. $\tilde{K} = K$. If there exists no path to refine, i.e. $\mathcal{P} = \emptyset$, then Algorithm 4.8 terminates.

4.4.3 DIOR using a MIP inner approximation

Algorithm 4.9 presents a DIOR algorithm based on MIP-IA refinement. The purpose of the algorithm is to compute a solution candidate of original problem (1.2). It iteratively divides a feasible set and generates new columns regarding the partition. In the beginning, procedure INITDIOR, depicted in Algorithm 4.3, initializes the IA. Then, the algorithm alternately calls procedures INNERREFINE(u, \hat{z}, D, R) and SOLVEINNERMIP(R, D). Procedure INNERREFINE, described in Algorithm 4.8, refines MIP-IA problem (4.33) by generating disjunctive partitions of the feasible set. As input, it takes index u and solution point \hat{z} . Procedure SOLVEINNERMIP(R, D) computes the input for INNERREFINE by solving MIP-IA problem (4.33). Index u denotes an index of active cells after solving MIP-IA problem (4.33), i.e. $t_u = 1$. Note that procedure INNERREFINE returns the index set V , which contains indices of new cells. If $V = \emptyset$, then the iterative procedure stops, since no cells were subdivided and no new columns were generated.

In the next stage, Algorithm 4.9 computes a solution candidate of original problem (1.2). Using the last solution point \hat{z} and index of active cells u , procedure

SOLVERPROJECTSUBPROBLEM solves resource-constrained projection sub-problem (4.38) and returns an integer-feasible solution y . After obtaining point y , procedure SOLVEFIXEDNLP(y) computes a solution candidate \tilde{x} of problem (1.2) by solving NLP problem with fixed integer variables (2.9). If \tilde{x} is feasible, i.e. $\tilde{x} \in X$, then it is assigned to the point x^* , a solution candidate of problem (1.2).

Procedure INNERREFINE(u, \hat{z}, D, R) does not guarantee to generate all possible columns, so Algorithm 4.9 provides only an estimated lower bound $\underline{\nu}$ of problem (1.2).

4.5 Numerical results

Algorithm 4.4 and 4.9 were implemented with Pyomo [48], as a part of the solver DECOGO. Note the sub-problems were not solved in parallel. For the experiments, we used SCIP 5.0 [44] for solving MINLP sub-problems, Gurobi 9.0.3 [47] for solving MIP/LP problems and IPOPT 3.12.13 [106] for solving NLP problems. We reformulated problems into block-separable form using natural block structure detection, described in Section 1.3. All computational experiments were performed using a computer with Intel Core i7-7820HQ 2.9 GHz CPU and 16 GB RAM.

4.5.1 Experiment with Algorithm 4.4 (DIOR1)

In this section, we illustrate the results of Algorithm 4.4 with Example 1.4 defined with one global constraint, outlined in Section 1.6.4. The optimal value of the problem is -8.5 with the optimal resources $(-4, 3.5)$ in space W_1 and $(-4.5, 6.5)$ in space W_2 . Note that we solved Pareto line-search sub-problem (4.19) to global optimality, in order to guarantee that the cone point is an NDP. As a stopping criterion for the algorithm, we used a tolerance on improvement of MIP-OA objective value and set it to $\epsilon_{MIP} = 10^{-5}$.

For line-search, the algorithm computed the ideal point $\underline{w}_1 = (-8, 0)$ and $\underline{w}_2 = (-6.2, 5)$. Algorithm 4.3 computed an initial OA point $\hat{w}_1 = (-3.6, 3)$ and $\hat{w}_2 = (-5, 7)$. Figure 4.1 shows that the optimal value of the MIP-OA (4.12) converged to the global optimum of (1.2) in 20 iterations after 12.5 seconds. It is interesting to notice that in space W_2 , for almost all OA solution points, the corresponding cone point v_2 was identical, i.e. OA solution \hat{w}_2 belonged to the feasible set.

For the instances with more than one global constraint, convergence was slower. In each iteration, the algorithm generated $m+1$ disjunctive cuts for the selected cell. These cuts were weak, since the algorithm first selected cells which did not improve the OA objective value. The cells, which improved the OA objective value, were selected only when other resources could not be improved anymore. Also, after several iterations,

4. A DECOMPOSITION-BASED INNER AND OUTER REFINEMENT ALGORITHM FOR NONCONVEX MINLP

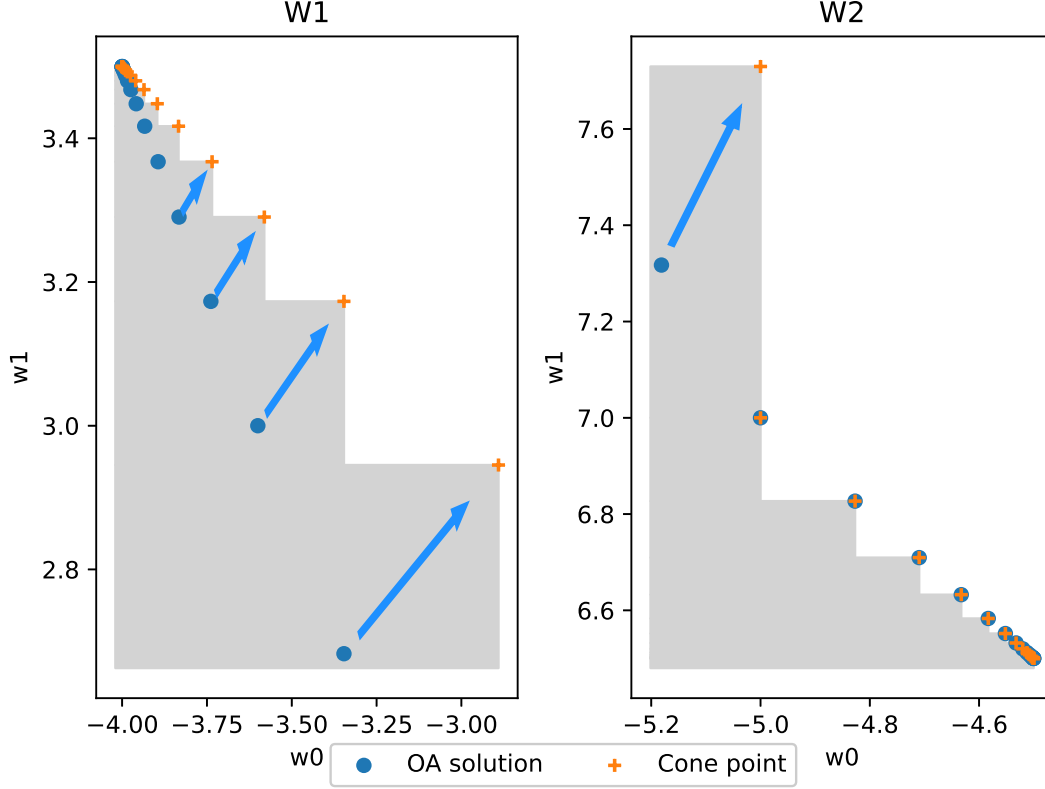


Figure 4.1: Steps 5-10 of Algorithm 4.4 for Example 1.4. The blue arrows represent a line-search towards the feasible set defined by OA solution \hat{w} and ideal point $\underline{w}_1 = (-8, 0)$ and $\underline{w}_2 = (-6.2, 5)$. The grey shaded area represents eliminated cones.

the MIP-OA master problem became more difficult to solve due to the huge amount of generated disjunctive cone cuts.

4.5.2 Experiments with Algorithm 4.9 (DIOR2)

In this section, we present the results for CG Algorithm 4.2 and DIOR2 Algorithm 4.9. For testing purpose, we selected several instances from MINLPLib [78]. More detailed statistics on the selected instances is given in Table 4.1.

In Column Generation (Algorithm 4.2), often smaller MINLP sub-problems can still be difficult to solve. Therefore, we set termination criteria for SCIP in order to generate feasible (not necessarily optimal) points of MINLP sub-problems. For this purpose, we utilized two parameters for the MINLP solver with the following values: (i) 500 for

maximum number of processed nodes after the last improvement of the primal bound and (ii) 0.01 for relative gap tolerance. In order to check convergence of Algorithm 4.2, we executed its one iteration without early termination of sub-problem solving, i.e. we did not apply any stopping criteria for the sub-solver. The disadvantage of this strategy is that Column Generation needs more iterations to converge than in situation, when sub-problems are always solved to optimality. Therefore, it solves more MINLP and LP sub-problems.

Table 4.1: Performance of CG Algorithm 4.2, CG.

Instances		n	$ K $	m	N_{LP}	N_{sub}	$ R $	$\nu^* - \nu_{LP}$, %
1	alkyl	14	4	6	13	68	27	23.2
2	ex2_1_1	5	5	1	3	30	15	11.2
3	example 1.4	4	2	1	4	12	8	1.2
4	pooling_rt2tp	34	3	18	19	75	41	25.9
5	sep1	29	2	10	15	48	29	41.8
6	st_e05	5	3	2	4	39	9	78.3
7	st_glmp_kky	7	3	4	5	42	12	20.0
8	st_jcbpaf2	10	5	13	2	65	19	17.9
9	tln2	8	3	6	4	36	9	21.5
10	util	145	3	15	39	120	55	3.6

Table 4.1 shows the characteristics for the selected test set: problem size n , number of blocks $|K|$ and number of global constraints m . In the table, the performance measures of Column Generation are given by the number of solved LP master problems N_{LP} , number of solved MINLP sub-problems N_{sub} and number of generated columns $|R|$. Note that N_{LP} also denotes the number of iterations of Algorithm 4.2. We also computed a relative duality gap $\nu^* - \nu_{LP}$, where ν_{LP} denotes the objective value of LP master problem (1.35) and ν^* denotes the best known objective function value.

Table 4.1 illustrates that the number of generated columns $|R|$ in Algorithm 4.2 is smaller than the number of solved MINLP sub-problems N_{sub} . This indicates that the MINLP sub-problem generated the same column several times. For some instances, it was necessary to solve more LP master problems, but these sub-problems were easy, as can be observed in Table 4.3.

For the experiments with DIOR2, the maximum number of supporting columns in (4.37) was set to $p = 3$, and the maximum number of MIP iterations (number of solved

4. A DECOMPOSITION-BASED INNER AND OUTER REFINEMENT ALGORITHM FOR NONCONVEX MINLP

MIP-IA master problem (4.33)) to 20, i.e. $N_{MIP} \leq 20$. Some MINLP sub-problems in steps 3-10 of Algorithm 4.9 were solved to optimality. e.g. sub-problem (4.38), in order to check whether the resources of MIP-IA are feasible. Other MINLP sub-problems were not solved to optimality, i.e. they were solved as in Algorithm 4.2.

Table 4.2: Performance of Algorithm 4.9, DIOR2.

Instances		N_{sub}	$ R $	N_{MIP}	$\underline{\nu}$	$\bar{\nu}$	ν^*
1	alkyl	150	79	4	-1.7	-1.8	-1.8
2	ex2.1.1	95	20	2	-17.0	-17.0	-17.0
3	example 1.4	21	11	2	-8.5	-8.5	-8.5
4	pooling_rt2tp	603	530	20	-4647.6	-3274.0	-4391.8
5	sep1	366	334	20	-499.3	-510.1	-510.1
6	st_e05	119	76	7	7049.3	7049.2	7049.2
7	st_glnp_kky	95	23	3	-2.5	-2.5	-2.5
8	st_jcbpaf2	252	93	5	-794.9	-794.9	-794.9
9	tlm2	88	17	3	5.3	5.3	5.3
10	util	653	450	20	1115.3	1034.7	999.6

In Table 4.2 about DIOR2, N_{sub} and $|R|$ include the number of solved MINLP sub-problems and number of generated columns from Algorithm 4.2, respectively. The indicators are the number of MIP iterations N_{MIP} (number of solved MIP-IA (4.33)), optimal value $\underline{\nu}$ of MIP-IA (4.33) and objective value $\bar{\nu}$ at the primal solution point computed by the local solver.

Table 4.2 illustrates that the decomposition-based successive approximation approach is able to solve nonconvex MINLP models to global optimality. Notice that, like for CG, the number of solved MINLP sub-problems N_{sub} is higher than the number of generated columns $|R|$. For Example 1.4, DIOR2 reduced the number of iterations from 20 to 2 compared to DIOR1.

Table 4.3 compares Algorithm 4.9 to SCIP 5.0 [44] in terms of computing time. All settings of SCIP were set to default. For each instance, we compared total solution time T of DIOR2 with time spent by SCIP T_{SCIP} . Note that T also includes time spent for decomposition T_{dec} . T_{LP} , T_{MIP} and T_{MINLP} denote the time spent on solving LP master problems, MIP master problems and MINLP sub-problems, respectively.

Table 4.3: Comparing Algorithm 4.9 with the SCIP solver. All values in seconds.

	Instances	T_{dec}	T_{LP}	T_{MIP}	T_{MINLP}	T	T_{SCIP}
1	alkyl	1.2	0.1	1.3	22.1	26.7	2.02
2	ex2.1.1	0.5	0.01	0.4	6.8	8.7	0.45
3	example 1.4	0.1	0.01	0.4	0.9	1.7	0.01
4	pooling_rt2tp	2.1	0.1	12.5	163.1	203.0	1.84
5	sep1	0.9	0.1	11.9	80.0	104.4	1.81
6	st_e05	0.2	0.1	2.6	13.3	17.8	1.59
7	st_glm_p_kky	0.7	0.1	0.9	8.5	11.4	1.8
8	st_jcbpaf2	0.7	0.1	1.7	22.2	28.3	0.89
9	tln2	0.5	0.01	0.8	4.7	7.1	0.02
10	util	18.7	0.1	12.0	88.3	143.0	2.28

Table 4.3 shows that SCIP required less total time than Algorithm 4.9. However, Algorithm 4.9 spent most of the runtime T on solving MINLP sub-problems. This shows the potential for solving these sub-problems in parallel. T_{MIP} depends on N_{MIP} , i.e. more iterations leads to more time spent on solving MIP master problems. The MIP master problem becomes more difficult to solve when a lot of cuts are generated. However, T_{LP} is relatively small and it does not depend on the number of solved LP master problems. Note that for Example 1.4, DIOR2 improved the solution compared to DIOR1 from 12.5 s to 1.7 s.

4.6 Conclusions

This chapter presents two algorithms for solving MINLP problems (1.2). The depicted algorithms are based on Column Generation. They solve smaller MINLP sub-problems, of which the solutions are combined into a master LP and MIP problem up to convergence. Both algorithms, DIOR1 and DIOR2, use low-dimensional MINLP sub-problem solutions for refining resource-constrained LP and MIP master problems.

We focused on the question whether DIOR1 is an efficient algorithm to solve large-scale problems with many coupling constraints. The experiments show that DIOR1 is slow. This is due to the fact that this approach generates weak cuts to improve an OA. Another research question was about DIOR2. We investigated whether the heuristic IA refinement procedure DIOR2 is capable of computing high-quality solution candidates of large-scale problems. The numerical results demonstrate that DIOR2 is

4. A DECOMPOSITION-BASED INNER AND OUTER REFINEMENT ALGORITHM FOR NONCONVEX MINLP

a faster heuristic algorithm than DIOR1. Moreover, DIOR2 is able to produce high-quality solutions. However, an established well-implemented BB algorithm is faster than DIOR2.

The advantage of these approaches is that they avoid building one global BB tree to solve a problem. Moreover, the sub-problems can be solved in parallel to generate the columns, which do not have to be optimal. Furthermore, an arbitrary solver can be used for solving the sub-problems.

The investigation of this chapter has been presented in [86].

A heuristic Column Generation Algorithm for solving energy system planning problems

The numerical results of the heuristic DIOR algorithm, presented in Chapter 4, show that most of the time is spent for solving low-dimensional MINLP sub-problems during the CG procedure. Our research questions are: how to generate quickly new columns with techniques other than solving MINLP sub-problems; whether projection from a convex relaxation can improve the quality of solution candidates. Although the elaborated CG heuristic algorithm can be applied to any MINLP problem (1.2), in this chapter, we focus on instances used for modeling decentralized energy supply systems (DESS) [5, 83].

5.1 Introduction

The experience with the heuristic DIOR algorithm have shown that it spends most of the computational time for Column Generation (CG). This is due to the fact that CG is based on solving small nonconvex MINLP problems using BB. Moreover, very often, the algorithm generates a column that was already generated before, which makes CG inefficient. To address these issues, we introduce a multi-tree decomposition-based heuristic algorithm, which is an extension of the Column Generation (CG) algorithm, presented in Chapter 4.

We develop several strategies for accelerating the Column Generation (CG): a simple rounding heuristic for solving MINLP sub-problems and the Frank-Wolfe algorithm [39]. Our research questions are: whether these techniques help to speed up the CG procedure and generate more columns that lead to a better inner approximation; how to guarantee that the algorithm can compute high-quality solutions of the original

5. A HEURISTIC COLUMN GENERATION ALGORITHM FOR SOLVING ENERGY SYSTEM PLANNING PROBLEMS

problem. We investigate the potential of projecting a solution of the convex relaxation onto the feasible set, similar to a Feasibility Pump [22, 32]. Moreover, we explore the possibility of generating several solution candidates while solving the projection problem. Such approach may increase the probability to find better solution candidates. Moreover, it may help to avoid the situation where the algorithm is not able to find any feasible solution.

We also look into the properties of the DESS model. Many blocks of those models are defined by linear constraints and continuous variables, i.e. $G_k = Y_k = \mathbb{R}^{n_k}$. In this chapter, these blocks are merged into one single linear block and denoted as follows

$$X_1 := \{y \in [\underline{x}_1, \bar{x}_1] \subset \mathbb{R}^{n_1} : a_{1j}^T y - b_{1j} \leq 0, j \in J_1\}. \quad (5.1)$$

In fact, polyhedral set X_1 is a polytope defined by its extreme points (vertices). If the size of block X_1 is large, then the convergence of Column Generation might be slow, since it would generate all vertices of polytope X_1 . Here, the main question is how to deal with this linear block efficiently.

This chapter is organized as follows. Section 5.2 presents how to deal with the linear block and recaps CG from Chapter 4. Section 5.3 explains acceleration ideas for CG. Section 5.4 describes a primal heuristic for finding solution candidates. Section 5.5 presents the main CG heuristic algorithm. Section 5.6 analyzes convergence of the presented algorithms. Section 5.7 shows numerical results of the algorithm with a DESS model. Finally, Section 5.8 discusses conclusions.

5.2 Traditional Column Generation

In this section, we briefly recall the CG procedure, described in Chapter 4. Moreover, we explain the idea how to deal with the linear block of the DESS model.

5.2.1 Handling the linear block (sub-problem)

In the DESS model the dimension n_1 of the linear block (5.1) is much higher than the dimension n_k of other blocks. Preliminary numerical experiments showed that, if we distinguish the linear block instead of running CG over all blocks, we obtain a reduced runtime in order of magnitude from 48 hours towards 2 hours. Mathematically this is explained by the fact that a large number of polytope vertices, corresponding to the linear block, needs be generated as columns if it is treated as a nonlinear block. Therefore, we use the following modified LP-IA master problem, for which the linear

constraints of X_1 in (5.1) are directly integrated in the LP-IA:

$$\begin{aligned}
& \min \sum_{k \in K} w_{k0}(z_k, x_1) \\
& \text{s. t. } \sum_{k \in K} w_{ki}(z_k, x_1) \leq b_i, \quad i \in M_1, \\
& \quad \sum_{k \in K} w_{ki}(z_k, x_1) = b_i, \quad i \in M_2, \\
& \quad z_k \in \Delta_{|R_k|}, \quad k \in K \setminus \{1\}, \quad x_1 \in X_1,
\end{aligned} \tag{5.2}$$

where $w_k(z_k, x_1)$ is defined using (1.37) as follows

$$w_k(z_k, x_1) := \begin{cases} w_k(z_k) & : k \in K \setminus \{1\}, \\ A_1 x_1 & : k = 1. \end{cases} \tag{5.3}$$

We distinguish $K \setminus \{1\}$ as a set of nonlinear blocks, whereas $x_1 \in \mathbb{R}^{n_1}$ are the variables of the linear block, and X_1 is defined by only local linear constraints as in (5.1). Note that from the definition of LP-IA (5.2) follows that R_1 does not contain the columns, i.e. $R_1 = \emptyset$. Therefore, it is not necessary always to solve the LP sub-problem corresponding to the linear block.

Lemma 5.1. Let $R_1 = \text{vert}(W_1)$ be the set of vertices of W_1 . Then problem (5.2) and (1.35) are equivalent.

Proof. By definition (5.1), X_1 defines a polytope. Hence, W_1 is a polytope defined by a linear transformation of X_1 , and $W_1 = \text{conv}(R_1)$. Replacing $A_1 x_1$ in (5.3) by $w_1(z_1)$ proves the statement. \square

Procedure `SOLVEINNERLP`(R) solves (5.2). It returns a primal solution $(x_k)_{k \in K}$, where x_1 is a solution of the linear block and $x_k = x_k(z_k)$ with

$$x_k(z_k) := \sum_{j \in [S_k]} z_{kj} y_{kj}, \quad z_k \in \Delta_{|S_k|}, \quad k \in K \setminus \{1\}, \tag{5.4}$$

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

5.2.2 Column Generation using MINLP sub-problems

The algorithm generates columns by solving MINLP sub-problems, defined by (4.3), regarding a search direction $d \in \mathbb{R}^{m+1}$, where d is typically defined by a dual solution

5. A HEURISTIC COLUMN GENERATION ALGORITHM FOR SOLVING ENERGY SYSTEM PLANNING PROBLEMS

Algorithm 5.1 Generation of columns

```

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

```

$\mu \in \mathbb{R}^m$ of LP-IA (1.35), i.e. $d = (1, \mu^T)$. Notice that the result y_k corresponds to an extreme point of X_k as well as W_k and is a supported Pareto point in the resource space, see Section 1.6.4.

Procedure $\text{SOLVEMINLP SUBPROBL}(d)$ solves sub-problem (4.3) and is used in procedure $\text{ADDCOL}(d, R_k)$, described in Algorithm 5.1, to add a column $A_k y_k$ to R_k . Moreover, the procedure computes the reduced cost, defined by (4.4), which is used later to measure the impact of the procedure. If $\delta_k < 0$, then column $A_k y_k$ improves the objective value of (5.2). In other cases, the objective value of (5.2) will be unchanged [91].

Algorithm 5.2 presents a procedure for generating columns by alternately solving LP-IA master problem (5.9) and generating columns using Algorithm 5.1. The set \hat{K} is a subset of the set of blocks K .

Algorithm 5.2 Column generation

```

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

```

5.2.3 Initialization of the column set

Algorithm 5.3 presents the procedure which initializes column set R_k . Like in Algorithm 4.1, it is based on the subgradient method which maximizes dual function (4.1). The computation of the step length α^p is done by the rule described in (4.2). In contrary to Algorithm 4.1, Algorithm 5.3 treats linear block differently, i.e. it solves the LP

Algorithm 5.3 IA initialization

```

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

```

sub-problem without adding the corresponding columns to the column set R .

5.3 Acceleration of Column Generation

In order to accelerate CG without calling an MINLP routine, we developed two methods. The first approach generates columns by performing NLP local search from starting points provided by an LP-IA. The second approach is a Frank-Wolfe (FW) algorithm for quickly solving the convex hull relaxation (1.31).

5.3.1 Fast Column Generation using NLP local search and rounding

Since in the beginning only few columns are available, often LP-IA master problem (5.2) is infeasible, i.e. $H \cap \prod_{k \in K} \operatorname{conv}(R_k) = \emptyset$. Therefore, we use the following modified LP-IA master problem with slacks, similar to problem (4.5)

$$\begin{aligned}
 \min \quad & \sum_{k \in K} w_{k0}(z_k, x_1) + \theta \sum_{i \in M_1 \cup M_2} s_i^+ + s_i^- \\
 \text{s. t.} \quad & \sum_{k \in K} w_{ki}(z_k, x_1) \leq b_i + s_i^+, \quad i \in M_1, \\
 & \sum_{k \in K} w_{ki}(z_k, x_1) = b_i + s_i^+ - s_i^-, \quad i \in M_2, \\
 & z_k \in \Delta_{|R_k|}, \quad k \in K \setminus \{1\}, \quad x_1 \in X_1, \\
 & s_i^+, s_i^- \geq 0, \quad i \in M_1 \cup M_2,
 \end{aligned} \tag{5.5}$$

where a penalty weight $\theta > 0$ is sufficiently large. For the sake of simplicity, we consider a vector s as all slacks together, i.e. $s = (s^+, s^-)$. Note that linear block

5. A HEURISTIC COLUMN GENERATION ALGORITHM FOR SOLVING ENERGY SYSTEM PLANNING PROBLEMS

Algorithm 5.4 Approximate sub-problem solving

```

1: function APPROXSOLVEMINLPSUBPROBLEM( $x_k, d$ )
2:    $\tilde{y}_k \leftarrow \text{SOLVENLP SUBPROBLEM}(x_k, d)$ 
3:    $\hat{x}_k \leftarrow \text{ROUND}(\tilde{y}_k)$ 
4:    $\tilde{x}_k \leftarrow \text{SOLVEFIXEDNLP SUBPROBLEM}(\hat{x}_k)$ 
5:   return  $\tilde{x}_k$ 

```

X_1 is directly integrated as in problem (5.2). Procedure $\text{SOLVESLACKINNERLP}(R)$ solves (5.5) and returns a solution point x , dual solution μ and slack values s . If the slack variables are nonzero, i.e. $s \neq 0$, in order to eliminate nonzero slack variables, procedure $\text{GETSLACKDIRECTIONS}$ computes a search direction $d \in \mathbb{R}^m$, as defined in (4.6).

Since, for the CG algorithm, it is sufficient to compute high-quality local feasible solutions, we present the local search procedure $\text{APPROXSOLVEMINLP SUBPROBL}$ in Algorithm 5.4 based on a rounding of locally feasible point. The goal of this procedure is to avoid usage of a MINLP solver for solving sub-problems and, therefore, reduce the time for sub-problem solving. The inputs of local search procedure $\text{APPROXSOLVEMINLP SUBPROBLEM}$, sketched in Algorithm 5.4, are the block solution x_k as a starting point and the direction d or $(1, \mu)$ as a search direction. Procedure $\text{SOLVENLP SUBPROBLEM}$ computes a local minimizer of the integer relaxed sub-problem

$$\begin{aligned} \tilde{y}_k &:= \operatorname{argmin} d^T A_k x \\ \text{s. t. } x &\in G_k \cap L_k \end{aligned} \quad (5.6)$$

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

$$\begin{aligned} \tilde{x}_k &:= \operatorname{argmin} c_k^T x_k \\ \text{s. t. } x_k &\in G_k \cap L_k, \quad x_{ki} = \hat{x}_{ki}, \quad i \in [n_{k2}], \end{aligned} \quad (5.7)$$

and using the continuous variable values of \tilde{x}_k as a starting point. The complete Column Generation procedure is depicted in Algorithm 5.5.

5.3.2 CG using a Frank-Wolfe algorithm

In this section, we present a Frank-Wolfe algorithm which is an alternative way to generate columns. It solves convex hull relaxation (1.31), where global constraints H

Algorithm 5.5 Approximate Column Generation

```

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

```

are replaced by the following quadratic penalty function

$$\sum_{i \in [m]} \sigma_i \left(\sum_{k \in K} w_{ki} - b_i \right)^2, \quad (5.8)$$

where $\sigma \in \mathbb{R}_+^m$ denotes a vector of penalty weights. Consider the following reformulation of problem (1.31)

$$\min Q(w, \sigma) \quad \text{s. t.} \quad w_k \in \text{conv}(W_k), \quad k \in K, \quad (5.9)$$

where $Q(w, \sigma)$ is defined as a sum of original objective function $\sum_{k \in K} w_{k0}$ and quadratic penalty function (5.8):

$$Q(w, \sigma) := \sum_{k \in K} w_{k0} + \sum_{i \in [m]} \sigma_i \left(\sum_{k \in K} w_{ki} - b_i \right)^2, \quad (5.10)$$

Let μ^* be an optimal dual solution of (1.31) regarding global constraints $w \in H$ and set the penalty weights $\sigma_i = 0$ if $\mu_i^* = 0$, and $\sigma_i \geq |\mu_i^*|$ else, for $i \in [m]$. Then it can be shown that (5.8) is an exact penalty function and (5.9) is a reformulation of convex relaxation (1.31), i.e. (1.31) is equivalent to (5.9).

5. A HEURISTIC COLUMN GENERATION ALGORITHM FOR SOLVING ENERGY SYSTEM PLANNING PROBLEMS

Algorithm 5.6 Fast Column Generation using a Frank-Wolfe approach

```

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

```

Algorithm 5.6 presents a Frank-Wolfe (FW) algorithm for approximately solving convex problem (5.9). For acceleration, we use the Nesterov direction update rule [90], line 19. We set the penalty weight $\sigma = |\mu|$, where μ is a dual solution of LP-IA (5.5). One step of the FW algorithm is performed by approximately solving the problem (5.9) with a linearized objective

$$\min \nabla_w Q(\tilde{w}, \sigma)^T w \quad \text{s. t. } w_k \in \text{conv}(W_k), \quad k \in K, \quad (5.11)$$

which is equivalent to solving the sub-problems for all $k \in K$

$$\min \nabla_{w_k} Q(\tilde{w}, \sigma)^T w_k \quad \text{s. t. } w_k \in \text{conv}(W_k). \quad (5.12)$$

5.4 A heuristic algorithm for finding solution candidates

In order to compute quickly new columns, sub-problem (5.12) is solved by procedure `APPROXSOLVEMINLP SUBPROBLEM` (Algorithm 5.4). Note that the gradient $\nabla_{w_k} Q(\tilde{w}, \sigma)$ is defined by

$$\begin{aligned} \frac{\partial}{\partial w_{k0}} Q(w, \sigma) &= 1, \\ \frac{\partial}{\partial w_{ki}} Q(w, \sigma) &= 2\sigma_i \left(\sum_{\ell \in K} w_{\ell i} - b_i \right) =: \eta_i(w, \sigma). \end{aligned} \quad (5.13)$$

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

$$\theta = \underset{t}{\operatorname{argmin}} Q(\tilde{w} + t(r - \tilde{w}), \sigma) \quad (5.14)$$

in step 16 of Algorithm 5.6 can be easily solved, since its objective is defined by a convex quadratic function.

5.4 A heuristic algorithm for finding solution candidates

In this section, we present two heuristic procedures for computing solution candidates. The first one computes a feasible solution from the solution of LP-IA with slacks problem (5.5). The second one computes high-quality solution candidates for original problem (1.2).

Algorithm 5.7 presents an initial procedure to compute a solution candidate. The aim of this procedure is to eliminate slacks in LP-IA master problem (5.5). Procedure `NLPRESOURCEPROJECT` performs an NLP local search of the following integer relaxed *resource-projection* NLP master problem

$$\begin{aligned} \min \quad & \sum_{k \in K} \|A_k x_k - A_k \tilde{x}_k\|^2 \\ \text{s. t. } \quad & x \in P, \\ & x_k \in G_k, \quad k \in K, \end{aligned} \quad (5.15)$$

where \tilde{x} is a solution of LP-IA master problem (5.5).

Using the potentially fractional solution \tilde{y} of (5.15), procedure `MIPPROJECT`(\tilde{y}) computes an integer globally feasible solution \hat{y} by solving the following *MIP-projection* master problem

$$\begin{aligned} \hat{y} = \underset{y}{\operatorname{argmin}} \quad & \sum_{k \in K} \|x_k - \tilde{y}_k\|_\infty \\ \text{s. t. } \quad & x \in P, \\ & x_k \in Y_k, \quad k \in K. \end{aligned} \quad (5.16)$$

5. A HEURISTIC COLUMN GENERATION ALGORITHM FOR SOLVING ENERGY SYSTEM PLANNING PROBLEMS

Algorithm 5.7 Initial heuristic algorithm to compute a solution candidate

```

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

```

The integer globally feasible solution \hat{y} is then used to perform an NLP local search, where integer variables are fixed starting from \hat{y} . Procedure $\text{SOLVEFIXEDNLP}(\hat{y})$ performs this operation by solving NLP problem with fixed integer variables (2.9).

Algorithm 5.8 presents a heuristic algorithm for computing a high-quality solution candidate of MINLP problem (1.2). The procedure is similar to Algorithm 5.7, but it does not use NLP resource-projection problem (5.15). Instead, the solution of LP-IA (5.2) is used directly in MIP projection problem (5.16). There is no guarantee that the optimal solution of (5.16) provides the best primal bound. Moreover, it may be infeasible for original problem (1.2). Therefore, we generate a pool \hat{Y} of feasible solutions of problem (5.16) provided by the MIP solver. \hat{Y} provides starting points for an NLP local search over the global space and increases the possibility of improving the quality of solution candidate.

Similar to Algorithm 5.7, Algorithm 5.8 starts with computing a solution \tilde{x} of problem (5.2) by calling procedure SOLVEINNERLP . Procedure $\text{SOLPOOLMIPPROJECT}(\tilde{x}, N)$ uses point \tilde{x} to generate solution set (pool) \hat{Y} of (5.16) of size N , which also includes an optimal solution. Like in Algorithm 5.7, those alternative solutions are used to perform an NLP local search over the global space, defined in (2.9) by fixing the integer valued variables. In order to find better solution candidates, these steps are repeated iteratively with updated point \tilde{x} . In each iteration, the point \tilde{x} is shifted towards the point x^* which corresponds to the best current primal bound of original problem (1.2). This is a typical heuristic local search procedure, which aims to generate a different solution pool \hat{Y} in each iteration of the algorithm. Algorithm 5.8 terminates when the maximum number of iterations is reached or the best primal bound in the current iteration does not improve the best primal bound from the previous iteration.

Algorithm 5.8 Heuristic algorithm to compute solution candidates

```

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

5.5 Main algorithm

Algorithm 5.9 describes a multi-tree procedure for computing a solution candidate of original problem (1.2). Procedure IAINIT (Algorithm 5.3) initializes IA. Since problem (5.5) may have nonzero slack values, the algorithm eliminates them by computing a first primal solution. This is done by alternately calling procedures APPROXCOLGEN (Algorithm 5.5) and FINDSOLUTIONINIT (Algorithm 5.7). For quickly improving convex relaxation (1.31), the algorithm calls FW-based Column Generation procedure FWCOLGEN (Algorithm 5.6).

In the main loop, the algorithm alternately performs COLGEN (Algorithm 5.2) and heuristic procedure FINDSOLUTION (Algorithm 5.8) for computing solution candidates. Procedure COLGEN is performed for a subset of blocks $\hat{K} \subseteq K \setminus \{1\}$, in order to keep the number of solved MINLP sub-problems low. Moreover, focusing on a subset of blocks helps to avoid computing already existing columns. The blocks can be excluded for a while by looking at the value of reduced cost $\delta_k, k \in K \setminus \{1\}$, which is computed in line 14, as defined in (4.4). Reduced block set \hat{K} contains the blocks where the reduced cost is negative, i.e. $\delta_k < 0, k \in K \setminus \{1\}$, and is updated at each main iteration by solving the sub-problems for the full set K . Note that if the reduced cost δ_k is nonnegative for all blocks, i.e. $\delta_k \geq 0, k \in K \setminus \{1\}$, the Column Generation procedure converges and the algorithm terminates.

5. A HEURISTIC COLUMN GENERATION ALGORITHM FOR SOLVING ENERGY SYSTEM PLANNING PROBLEMS

Algorithm 5.9 The heuristic CG Algorithm

```

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

```

5.6 Convergence analysis

In this section, we discuss the convergence of Column Generation in Algorithm 5.9 and the Frank-Wolfe procedure in Algorithm 5.6.

5.6.1 Convergence of Column Generation (Algorithm 5.9)

The convergence proof of the Column Generation algorithm is due to its equivalence to the dual cutting-plane algorithm, see Lemma 4.10 in [91]. Note that the proof is not based on the computation of reduced cost $\delta_k, k \in K$, defined in (4.4). However, it can be used for measuring the impact of new columns and as a criterion for algorithm termination, see [91]. For the convergence proof, we assume that all LP master problems (1.35) and MINLP sub-problems (4.3) are solved to optimality. Since Algorithm 5.2 is performed regarding the subset of the blocks \hat{K} , we ensure that main Algorithm 5.9 converges by performing a standard CG step in line 14 regarding all blocks. Note that direct integration of linear block (5.1) into LP-IA master problem (5.2) is equivalent to performing the CG algorithm regarding that block until convergence, as shown in

Lemma 5.1.

Proposition 5.2. Let x^p be a solution of LP-IA (1.35) at the p -th iteration of Algorithm 5.9 at line 12 and ν^* be an optimal value of convex hull relaxation (1.31). Then $\lim_{p \rightarrow \infty} c^T x^p = \nu^*$.

Proof. The proof is equivalent to the proof of Proposition 4.11 in [91]. \square

5.6.2 Convergence of the Frank-Wolfe algorithm (Algorithm 5.6)

Algorithm 5.6 combines the original Frank-Wolfe algorithm [39] with Nesterov update rule [90]. The approach proposed by Nesterov in [90] has a convergence rate $O(1/p^2)$, whereas the original Frank-Wolfe algorithm has a slower convergence rate of $O(1/p)$, see Theorem 1 in [55]. In order to prove the convergence of Algorithm 5.6, we assume that all sub-problems (5.12) are solved to global optimality. Next, we state that Algorithm 5.6 has a convergence rate $O(1/p^2)$.

Proposition 5.3. Let $\nu^p := Q(\tilde{w}^p, \sigma)$ be a value of the objective function defined by quadratic penalty function (5.10) at p -th iteration of Algorithm 5.6 and ν^* be an optimal value of convex hull relaxation (1.31). Assume that $\sigma_i \geq |\mu_i^*|, i \in [m]$, where μ^* defines an optimal dual solution of (1.31). Then, there exists a constant C such that $\forall p \geq 0$

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

Proof. The proof is equivalent to the proof of Theorem 1 in [82]. \square

5.7 Numerical results

In this section, we evaluate the performance of Algorithm 5.9 by solving several DESS model instances taken from DESSLib. More details on DESSLib are given in Section 5.7.1. Algorithm 5.9 was implemented with Pyomo [48], as a part of the solver DECOGO. Note that the sub-problems were not solved in parallel. For these experiments, we used SCIP 7.0.0 [40] for solving MINLP sub-problems, Gurobi 9.0.1 [47] for solving MIP/LP master-problems and IPOPT 3.12.13 [106] for performing an NLP local search in master and sub-problems. All computational experiments were performed using a computer with AMD Ryzen Threadripper 1950X 16-Core 3.4 GHz CPU and 128 GB RAM.

For the experiments with Algorithm 5.9, we used the following stopping criteria, unless another is mentioned:

5. A HEURISTIC COLUMN GENERATION ALGORITHM FOR SOLVING ENERGY SYSTEM PLANNING PROBLEMS

- Time and iteration limit in Algorithm 5.9 were set to 12 hours and 20 iterations, respectively.
- The iteration limit of Algorithm 5.2 was set to 5.
- The iteration limit of the outer and inner loop of Algorithm 5.6 were set to 5 and 10, respectively.
- In Algorithm 5.8, the iteration limit was set to 5, the pool size to $N = 100$ and $\tau = 0.5$. To generate a solution pool with Gurobi, we used a parameter value of `POOLSEARCHMODE=1`. In this approach, the solver computes a set of N high-quality solutions including an optimal one, see [47] for more details.
- For SCIP, we set the maximum number of processed nodes after the last improvement of the primal bound to 1000, since, for CG, it is sufficient to compute feasible (not necessarily optimal) solutions, for more details see Section 4.5.

Our main research question is how to use the decomposable structure of DESS models to generate a new approach which may be an alternative to branch-and-bound-based state-of-the-art software like BARON [100]. During the investigation, we developed several elements to speed up the computation. To evaluate their effect, first in Section 5.7.2 we measure the effect of distinguishing the linear block and adding the fast FW approach for generating columns that lead to a better inner approximation. The idea to use a pool of high-quality MIP solutions is numerically investigated in Section 5.7.3. The pool generates a group of starting points to look for feasible solutions. That does not only affect the quality of the reached best solution, but also provides alternatives for the design problem that may be analyzed by the decision maker. Finally in Section 5.7.4, we compare the outcomes and the solution process of Algorithm 5.9 to that of the state-of-the-art solver BARON and to the problem-tailored MIP approximation algorithm presented in [45].

5.7.1 DESSLib model instances

A decentralized energy supply system (DESS) is a complex, integrated system consisting of several energy conversion units, energy supply and demand forms. The MINLP models are used to design these systems. The DESS model is typically defined by a superstructure which includes all possible components (e.g. boiler, engine, turbo-chiller, absorption chiller) and their interconnections. The optimization goal is to identify a system design and its schedule such that it simultaneously minimizes or maximizes an objective function value. The DESS model is described in detail in [45].

The DESS model is published as library DESSLib [5] including multiple model instances of varying complexity. The instances differ in three properties:

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

For example, instance S8L4-0 consists of maximal two units for each of the four component types (boiler, engine, turbo-chiller, absorption chiller). Both units can have different nominal sizes and can be operated independently. Furthermore, instance S8L4-0 includes four load cases (L4), and uses the data variation with the index zero.

The performance of the problem-tailored MIP approximation method, presented in [45], has been compared to state-of-the-art solvers using DESSLib. The results of comparison are accessible for downloading at [5].

We developed the DESS model with the algebraic modeling language Pyomo. The blocks K were defined manually using Pyomo component `Block`. This component serves as a container for variables, constraints, etc [48]. For each component of the DESS model, individual Pyomo blocks were created. These main blocks contained all variables associated with these blocks. Additionally, one or multiple sub-blocks were created within the component block to store the variables and constraints with load case dependency. Each sub-block can contain one or more load cases and its size can be controlled by the user.

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

To illustrate the impact of direct integration of one linear block into the LP-IA master problem, as defined in (5.2), we compare two runs of Algorithm 5.9 on the same instance: the first run used the LP-IA formulation given by (5.2) and the second run used the formulation given by (1.36). The comparison was done for instance S16L16-1, see Table 5.2 for more details.

For this particular instance, approximately 37% of all variables belongs to the linear block. Whereas the CG algorithm for formulation (1.36) relies on generating linear block polytope vertices as columns, the second formulation (5.2) may save this effort. The difference can be observed in Figure 5.1. The runtime of Algorithm 5.9 with direct linear block integration was drastically reduced from 48 hours to approximately 2 hours. It also showed a significant improvement of the convergence speed of the IA objective value and a significant improvement of its final value.

The next question focuses on the convergence impact of including procedure FW-COLGEN (Algorithm 5.6) into Algorithm 5.9. In order to measure the effect, we per-

5. A HEURISTIC COLUMN GENERATION ALGORITHM FOR SOLVING ENERGY SYSTEM PLANNING PROBLEMS

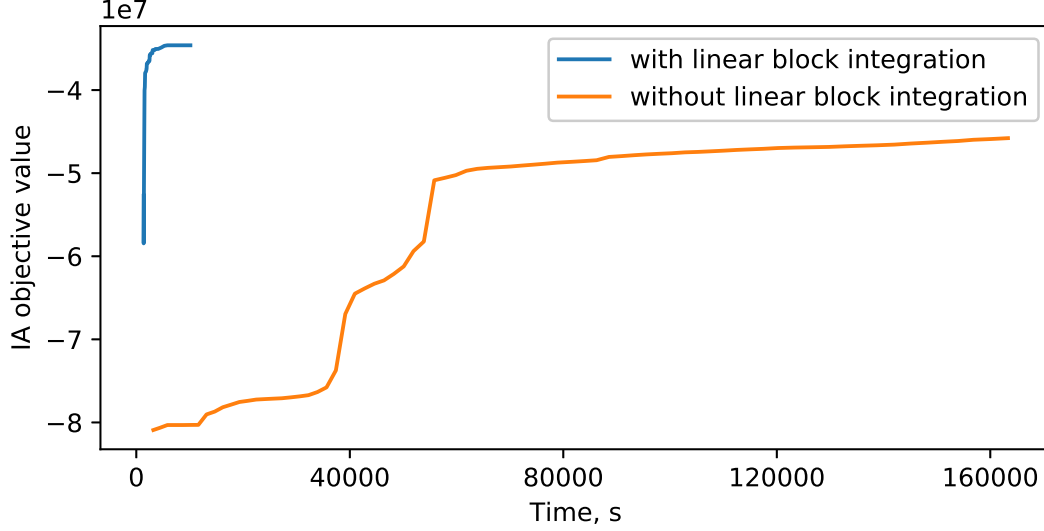


Figure 5.1: Convergence of the IA objective of Algorithm 5.9 with and without linear block integration for instance S16L16-1.

formed two equivalent runs of Algorithm 5.9: one of those did not use procedure FWCOLGEN. Since we only focused on the convergence of the IA objective value, we did not execute procedure FINDSOLUTION (Algorithm 5.8). The test instance is S4L4-1, see Table 5.2 for more details.

Figure 5.2 shows that the IA objective value of problem (5.2) converged faster with procedure FWCOLGEN at the initial stage of Algorithm 5.9. Moreover, at the beginning of the algorithm, FWCOLGEN IA objective value was very close to the final IA objective value, i.e. after 50 seconds, the IA objective value with procedure FWCOLGEN is approximately 0.5 % worse than the final IA objective value. Figure 5.2 indicates that Algorithm 5.9 with procedure FWCOLGEN converges slower to the final IA objective value than without procedure FWCOLGEN. The reason for that is that FWCOLGEN fails to generate high-quality columns in the later stages of the algorithm. This is due to the fact that we have no guarantee that penalty weights σ , defined in (5.8), are sufficiently large. Another reason is that the procedure generates columns by solving NLP sub-problems heuristically. However, it has an advantage in speed when comparing column generation by solving MINLP sub-problems.

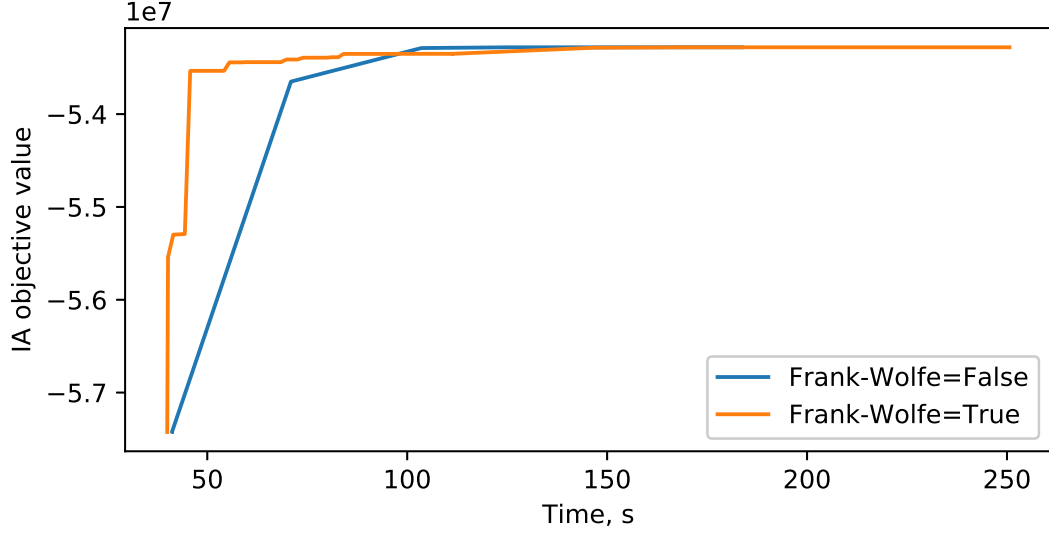


Figure 5.2: Convergence of the IA objective value of Algorithm 5.9 for S4L4-1 with and without the fast FW Column Generation.

5.7.3 Impact of using the solution pool in Algorithm 5.8

This section focuses on analyzing the usage of the solution pool in the heuristic procedure `FINDSOLUTION` (Algorithm 5.8). Note that using a single integer feasible point, we perform a local search with fixed NLP problem (2.9). This may imply that the result is not a feasible solution of the problem. Using a pool of starting points, may alleviate this effect.

Figure 5.3 shows the objective function values of all pool solutions, generated for instance S16L16-1. The solution with the best primal bound of approximately -5.15×10^7 was identified in the seventh iteration of Algorithm 5.9. From Figure 5.3, one can notice that other high-quality solutions could be computed in the earlier iterations of the algorithm. Note that the MIP solver was not always able to provide a solution pool of prescribed size of $N = 100$. Instead, it computed a solution pool of smaller size. Therefore, in Figure 5.3 one can observe that the number of generated feasible solutions varies over the iterations. Moreover, Figure 5.3 indicates that a solution pool may contain many feasible points with a very similar objective function value. These points are distinct, since GUROBI adds only a feasible point to the solution pool if the value of its integer variables is different for at least one integer variable [47].

The advantage of the approach is that a variety of feasible solutions are generated

5. A HEURISTIC COLUMN GENERATION ALGORITHM FOR SOLVING ENERGY SYSTEM PLANNING PROBLEMS

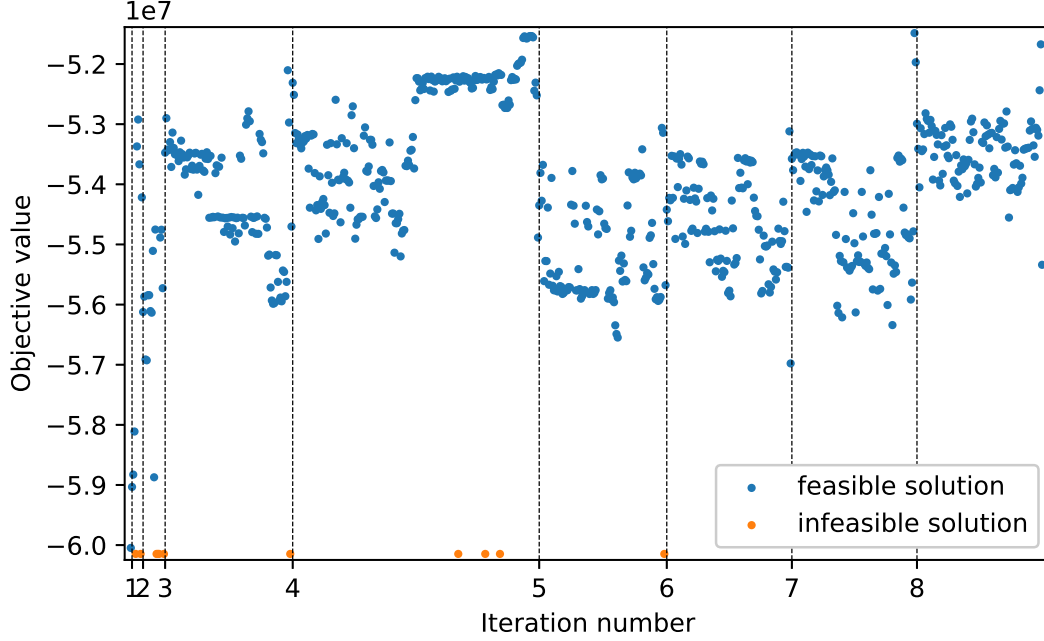


Figure 5.3: Solution pool for S16L16-1.

during the process and stored in the solution pool. The user may gain a more profound knowledge of the optimization problem by utilizing these near-optimal solutions and prefer them over the optimal solution if they better satisfy the requirements not considered in the mathematical model, e.g. safety, maintainability, operability, etc.

5.7.4 Comparison to other approaches

In this section, we compare the performance and solution quality of Algorithm 5.9 to other solvers on 12 selected instances from DESSLib [5]. The test instances were selected varying the number of unit components and number of load cases, as described in Section 5.7.1. Instance characteristics are reported in Table 5.2. The results of Algorithm 5.9 are compared to results obtained by the state-of-the-art MINLP solver BARON 20.4.14 [97]. We also compare results provided by adaptive discretization MINLP algorithm (AdaptDiscAlgo) reported in [45]. We do not compare the performance of Algorithm 5.9 to AdaptDiscAlgo, since the implementation of AdaptDiscAlgo is not publicly available. We analyze the results by looking at the primal bounds of AdaptDiscAlgo, reported in [5]. Note that AdaptDiscAlgo does not provide a valid lower bound and cannot be applied to general MINLP problems.

We used a 12 hour time limit for BARON and Algorithm 5.9. To evaluate the quality of a feasible solution, we define the gap to a reference (base) objective function value b as

$$\text{gap}(a, b) = -100 \frac{a - b}{\max\{|a|, |b|\} + 10^{-7}}, \quad (5.17)$$

where a is an objective value of the feasible solution point. Note that a “−” is used before the term in (5.17), since we consider a maximization problem. In case of minimization, it is omitted.

Table 5.1 compares the primal solution quality of decomposition Algorithm 5.9 versus that of BARON and AdaptDiscAlgo. We use the following notation: ν_{CG} and ν_{B} are the primal bound of CG Algorithm 5.9 and BARON, respectively; ν^* denotes the best known primal bound among BARON and AdaptDiscAlgo. Table 5.1 also presents the duality gap of Algorithm 5.9 and BARON. For this comparison, we denote $\underline{\nu}_{\text{CG}}$ as a dual bound of Algorithm 5.9 defined by the objective value of the last solution of (5.2) and $\underline{\nu}_{\text{B}}$ as a the dual bound of BARON given by the lower bound provided by BARON.

Table 5.1: Solution quality comparison of Algorithm 5.9 solution ν_{CG} to the primal BARON solution ν_{B} and best known solution ν^* . Note that negative value means that the primal bound has been improved. All values are given as percentage.

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

Table 5.1 shows that, for five instances, Algorithm 5.9 improves the primal bound

5. A HEURISTIC COLUMN GENERATION ALGORITHM FOR SOLVING ENERGY SYSTEM PLANNING PROBLEMS

of BARON. The Algorithm 5.9 result differs at most 3 % from the best known bound and for one instance improves the solution. We also evaluated the solution after performing two main iterations with Algorithm 5.9. The solution quality is similar. This means that with a limited solution time, high-quality solutions can be generated after two iterations. For the instances with more than 500 variables, the duality gap of Algorithm 5.9 was smaller than of BARON. We also observed that the duality gap value of Algorithm 5.9 depends mostly on the number of unit components (parameter S) and is less sensitive to the number of load cases (parameter L).

Table 5.2 provides the following statistics: problem size n ; number of blocks $|K|$; number of iterations N_{iter} performed by Algorithm 5.9; runtime of both algorithms, Algorithm 5.9 and BARON.

Table 5.2: Characteristics of selected test instances and performance comparison of Column Generation Algorithm 5.9 and BARON.

Instances	n	$ K $	N_{iter}	Time, s	
				Alg. 5.9	BARON
S4L4-1	204	9	5	247.05	2.71
S4L8-1	332	13	6	571.57	34.41
S4L16-1	588	17	11	1768.54	21.83
S4L24-1	844	25	11	2084.46	43202.13
S12L4-1	570	25	10	5013.30	43203.63
S12L8-1	926	37	18	31062.51	43201.73
S12L16-1	1638	49	11	43202.12	43203.14
S12L24-1	2350	73	5	43200.31	43203.20
S16L4-1	744	33	6	5241.29	43202.53
S16L8-1	1208	49	13	26365.72	43201.97
S16L16-1	2136	65	6	43200.37	43201.73
S16L24-1	3064	97	3	43200.42	43203.20

Despite the fact that Algorithm 5.9 does not solve the sub-problems in parallel and it is implemented with Python, Table 5.2 shows that for several larger instances it required less time than BARON. Moreover, we can see that for some instances Algorithm 5.9 did not reach the maximum number of iterations and the time limit. This indicates that the Column Generation converged.

We are interested in the computing time with increasing size of the problem in-

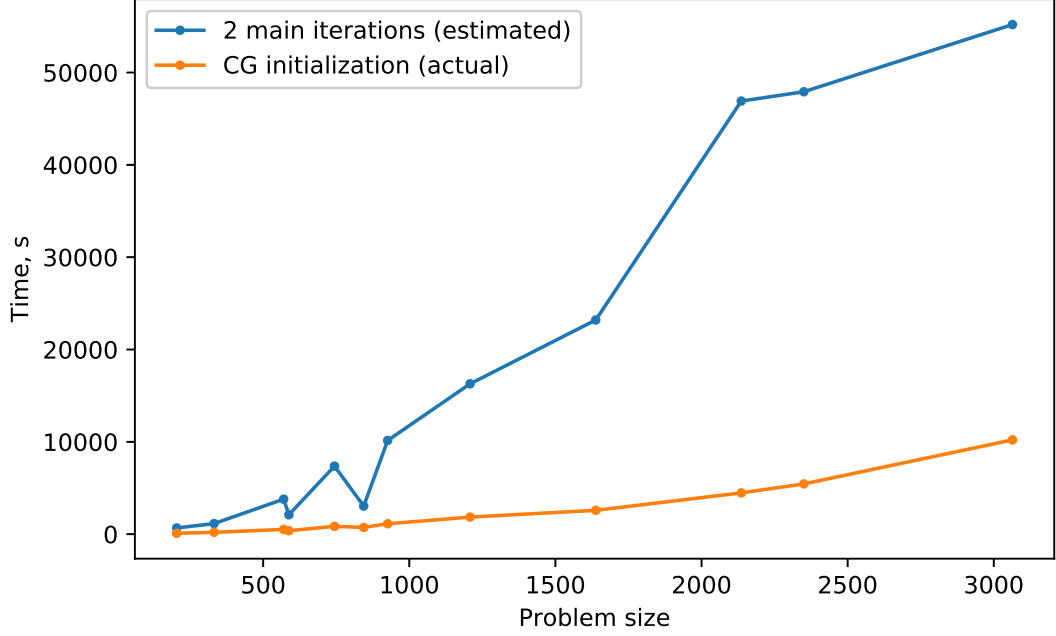


Figure 5.4: Algorithm 5.9 computing time versus problem size of all instances in Table 5.2.

stance with constant number of iterations. Figure 5.4 sketches the estimated runtime of Algorithm 5.9 after two main iterations and the actual time to initialize the Inner Approximation (all procedures before entering main iterations of Algorithm 5.9). The total time after two iterations includes the actual time for the CG initialization. We estimated the time for two main iterations based on the assumption that, for each instance, in each call of `FINDSOLUTION` (Algorithm 5.8), we obtain a solution pool by a MIP solver of size 500 ($500 = 5 \cdot 100$, where 5 is the prescribed number of iterations of Algorithm 5.8 and 100 is a prescribed size of the solution pool). In fact, the solution pool provided by the MIP solver can be smaller than the prescribed number, see Figure 5.3. Therefore, the estimated time indicates how long it would take if we would call the NLP solver 500 times to solve fixed NLP problem (2.9) in the first two main iterations of Algorithm 5.9. This time could be also useful for estimating the time complexity of similar instances with a much larger size. Figure 5.4 presents that the time to initialize Column Generation is relatively small in comparison to the estimated time after two main iterations of Algorithm 5.8. Figure 5.2 illustrates that the CG bound in the initial stage of the algorithm is almost converged. Figure 5.4 indicates that `FINDSOLUTION` (Algorithm 5.8) has a big influence on the runtime of the entire algorithm.

5. A HEURISTIC COLUMN GENERATION ALGORITHM FOR SOLVING ENERGY SYSTEM PLANNING PROBLEMS

5.8 Conclusions

This chapter presents a decomposition-based multi-tree heuristic algorithm for solving MINLP problem (1.2). The algorithm was applied to several DESS models, which are typically modeled by a MINLP. Such energy system models collect potentially difficult sub-models into a global model. Due to their high dimension, these models may be difficult to solve by generic state-of-the-art solvers. Decomposition looks to be an appropriate concept to apply to such models due to the structure of sub-models and global constraints. Our research question is how to do this in an efficient way.

Like in Chapter 4, the presented algorithm is based on Column Generation. One of the findings of this chapter is, that it is more efficient to deal with linear blocks in a separate way and include them directly into the master problem, instead of generating columns for them. In order to speed up the CG, we utilized a fast Frank-Wolfe algorithm for generating columns. In order to compute solution candidates, we developed a heuristic procedure which is based on solving NLP problems with fixed integer variables regarding a solution pool provided by a MIP projection master problem.

Typical features of the presented approach are: (i) no global branch-and-bound tree is used, (ii) sub-problems can be solved in parallel to generate columns, which do not have to be optimal, (iii) an arbitrary solver can be used to generate solutions of sub-models, (iv) a set of high-quality solutions, which may be inspected by the user, is generated.

Experiments with DESS model instances of several hundreds and thousands of variables showed that a state-of-the-art solver is faster for smaller problems and reaches high-quality solutions. An interesting observation is that, for larger models, the presented decomposition-based heuristic approach was able to compute better solutions for problems with thousands of variables than the generic well-established solver.

This study has been submitted to the international journal [83].

The implementation of the DECOGO solver

DECOGO (**D**ecomposition-based **G**lobal **O**ptimizer) is an object-oriented framework to solve convex as well as nonconvex MINLP problems [93]. The source code of the solver is written in Python. The algorithms presented in Chapters 2, 3, 4 and 5 were implemented within the DECOGO solver. Note that the presented numerical results in the previous chapters do not include a parallel sub-problem solving. The development of the solver has started more than three years ago and it is currently being further developed. In this chapter, we give a brief overview of the software.

6.1 Motivation

Python is a high-level open-source programming language which focuses on code readability. It has a comprehensive and large standard library that contains implementation of many standard functions. The Python syntax simplifies implementation of some task comparing to other major languages like C++. However, Python is slower than other major languages, because it is an interpreter-based language and not a compiler-based one. We chose Python to implement DECOGO, in order to quickly develop a working prototype without focusing on the performance aspect.

The solver is linked to the modelling language Pyomo [48], which is a Python-based open-source software package used to formulate, to solve and to analyze optimization models. Pyomo supports a various number of problem types, e.g. linear programming, nonlinear programming, mixed-integer nonlinear programming, stochastic programming, etc. One of the key features of Pyomo is a support of an object-oriented design for model definition. This means that modelling components are defined in Pyomo via Python classes. For example, class `Var` is used for declaring variables in a

6. THE IMPLEMENTATION OF THE DECOGO SOLVER

model, class `Constraint` is used for creating constraints of a model, etc. The functionality of Pyomo is flexible and can easily be extended.

There exist several MINLP solvers today. Like DECOGO, many of these are linked to one or more modelling systems, e.g. AMPL [37], GAMS [11, 17], JuMP [27], Pyomo [48], etc. Examples of nonconvex global MINLP solvers are α BB [3], ANTIGONE [79], Alpine [88, 89], BARON [97, 100], Couenne [7], GALINI [19], LINDO [66], SCIP [1, 2]. Examples of convex MINLP solvers are AlphaECP [107], AOA [53], BONMIN [13], DICOPT [46, 56], Juniper [57], Minotaur [72], Muriqui [75], Pajarito [20]. [18, 59, 102] provide a comprehensive overview of the MINLP software available today.

6.2 Structure and classes

The goal of the solver design is to split all modules into logically connected sub-packages. These sub-packages are:

1. *Solver* contains the implementation of algorithms and modules to manage solver settings and solver results.
2. *Model* contains all modules to store an original block-separable model and includes a module, which reformulates general problem (1.1) into block-separable problem (1.2).
3. *Problem* has all modules to create and manage all master and sub-problems; includes modules to store and manage approximation data, e.g. list of columns, linearization cuts, etc.
4. *Utility* contains the modules that define block data structures.

We describe shortly the classes, which are used for the implementation of the DECOA algorithm for convex MINLP problems (Algorithm 2.3) and the CG-based heuristic algorithm (Algorithm 5.9).

6.2.1 Model

The purpose of sub-package *model* is to implement the containers that store the data of the original model independently of Pyomo. However, the implemented containers store the nonlinear expressions of nonlinear constraints using Pyomo expressions. This exception requires less effort needed later to create Pyomo sub-problems for the algorithms. Sub-package *model* consists of the following classes:

- `PyomoModelDecomposer` detects the block-structure of the original model, as described in Section 1.3, or reads the block data based on the definitions in the

original model using Pyomo class **Block**; reformulates the original model into block-separable form (1.2) based on obtained block information.

- **VarDomain** stores the variable data, i.e. type, upper and lower bound.
- **LinearConstraint** stores information about a linear constraint; provides a possibility to evaluate the constraint.
- **NonlinearConstraint** stores information about a nonlinear constraint; contains Pyomo nonlinear expression from the original model; provides a possibility to evaluate the constraint using a Pyomo expression.
- **ObjectiveFunction** stores the data of an objective function; provides a method to evaluate the objective function.
- **CutPool** is a container class that stores and manages all linear constraints and an objective function given in the original model; creates and manages classes **LinearConstraint** and **ObjectiveFunction**.
- **SubModel** is a container class that stores data related to a specific block, i.e. variables and nonlinear local constraints; creates and manages classes **VarDomain** and **NonlinearConstraint**.
- **BlockModel** is a main container class that reads and stores the reformulated model obtained from **PyomoModelDecomposer**; creates and manages classes **CutPool** and **SubModel**.

6.2.2 Problem

Sub-package *problem* collects all types of sub-problems and master problems used in the algorithms. Only this sub-package contains Pyomo models. Each sub-problem and master problem has its corresponding class in this package. Each problem class contains its own Pyomo model as an attribute. Sub-package *problem* consists of the following classes:

- **InnerPoints** stores inner points $x_k \in X_k$ and corresponding columns $w_k = A_k x_k$.
- **LinearizationCuts** stores the linearization cuts that define OA problem (2.1).
- **ApproxData** is a container class that manages all data obtained during the solution process, i.e. list of linearization cuts and list of columns; manages classes **InnerPoints** and **LinearizationCuts**.
- **InnerMasterProblem** implements IA problem (1.36); utilizes columns (points in the transformed space) for construction of the problem; contains a Pyomo model.
- **MasterProblemBase** defines a base class for definition of master problems in the original space; constructs the variables of all blocks and the global constraints;

6. THE IMPLEMENTATION OF THE DECOGO SOLVER

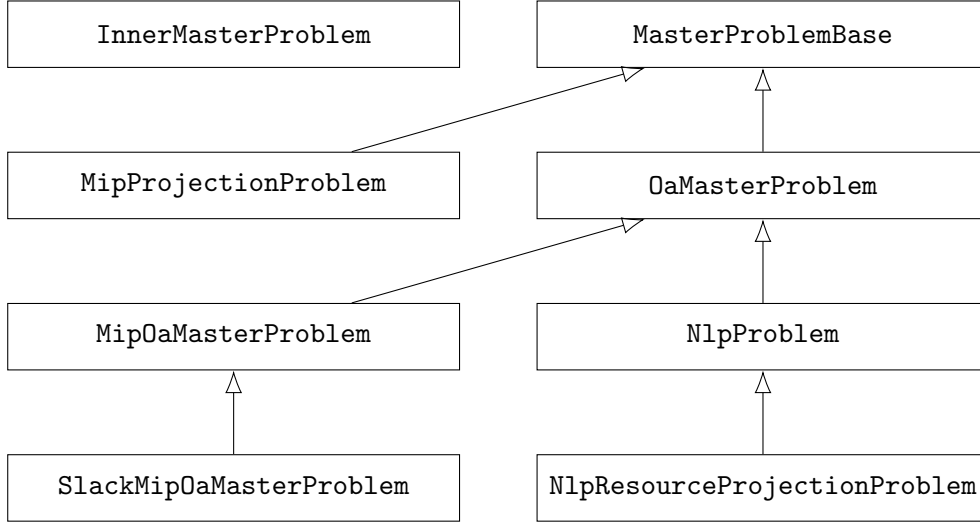


Figure 6.1: Class inheritance diagram of the master problems.

has the methods to include or relax the integrality constraints; does not contain the objective function; contains a Pyomo model.

- **OaMasterProblem** contains objective function $c^T x$ from original model (1.2); is derived from **MasterProblemBase**.
- **NlpProblem** implements the NLP master problem, i.e. nonlinear constraints that define set G are added; is derived from **OaMasterProblem**.
- **MipOaMasterProblem** contains linearization cuts; implements OA problem (2.1); is derived from **OaMasterProblem**.
- **SlackMipOaMasterProblem** implements partly-fixed OA problem (2.10); is derived from **MipOaMasterProblem**.
- **NlpResourceProjectionProblem** implements NLP resource-projection master problem (5.15); is derived from **NlpProblem**.
- **MipProjectionProblem** implements MIP projection problem (5.16); is derived from **MasterProblemBase**.
- **MasterProblems** is a container class that provides an access to all master problems mentioned above.
- **SubProblemBase** is a base class that defines a sub-problem; constructs variables of single block k and local constraints X_k ; has the methods to include or relax the integrality constraints; contains a Pyomo model.
- **MinlpSubProblem** contains objective function $c^T x$; implements sub-problem (4.3); is derived from **SubProblemBase**.

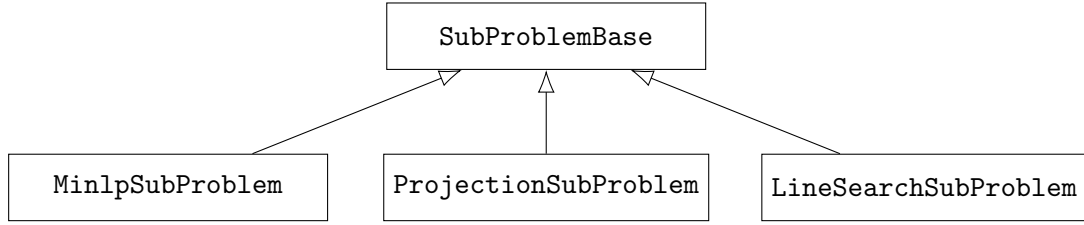


Figure 6.2: Class inheritance diagram of the sub-problems.

- `ProjectionSubProblem` implements projection sub-problem (2.5); is derived from `SubProblemBase`.
- `LineSearchSubProblem` implements line-search sub-problem (2.7); is derived from `SubProblemBase`.
- `SubProblems` is a container class that provides an access to all sub-problems mentioned above.
- `DecomposedProblem` is a main container class that manages all sub-problems and master problems as well as approximation data; provides an access to attributes and methods of `BlockModel`, `ApproxData`, `MasterProblems` and `SubProblems` from solver classes.

Figure 6.1 and Figure 6.2 illustrate the class inheritance diagrams for master problems and sub-problems, respectively. Note that class `InnerMasterProblem` is not connected to other master problem classes, since the corresponding model is defined in the transformed space using columns as described in (1.36).

6.2.3 Solver

Sub-package *solver* contains the implementation of the algorithms as well as the classes used for running the solver in general. It consists of the following classes:

- `ColGen` contains the CG heuristic algorithm, presented in Algorithm 5.9.
- `OaSolver` contains the OA algorithm for convex MINLP problems, presented in Algorithm 2.3.
- `Settings` stores and manages all settings for algorithms.
- `Results` stores all results obtained during and after of a solver execution.
- `DecogoSolver` initiates a solution process.
- `DecogoProcess` is an extension of original Python `Process` class which starts and, if necessary, interrupts a solution process in a controllable manner.

6. THE IMPLEMENTATION OF THE DECOGO SOLVER

- `DecogoSolverManager` is a class which starts an appropriate algorithm based on given settings; initiates the reformulating process by creating an instance of `PyomoModelDecomposer`; creates instances of container classes `BlockModel` and `DecomposedProblem`.
- `DecogoLogger` sets up the logger that prints results during and after the solution process.

6.2.4 Utility

Sub-package *utility* collects the classes that provide a general functionality that is not implemented in the standard libraries. For instance, a block-separable reformulation (1.2) requires a vector with two indices, i.e. block index k and index within the block i . Such data structures are implemented in this package. However, they are mainly an extension of standard Python package NumPy. Sub-package *utility* consists of the following classes:

- `SparseBlockVector` represents a vector which stores only nonzero values with respect to a block structure; provides an access to elements using two indices.
- `BlockVector` represents a vector which stores all values with respect to a block structure; provides an access to elements using two indices; supports basic operations like '+', '-', '*', '/'.

This thesis reports the results and findings from investigating decomposition-based successive approximation methods for MINLP problems. This chapter presents conclusions and a brief summary of the main issues. Moreover, it discusses some suggestions for research directions for future work.

We can distinguish two main research areas in this thesis. The first one deals with the Outer Approximation approach to solve convex as well as nonconvex MINLP problems. The second research area deals with the Column Generation approach for nonconvex MINLP problems. Both methods provided findings for the research questions investigated in this thesis.

7.1 Outer Approximation

The investigation of Outer Approximation covers several chapters of this thesis. Chapter 2 describes an algorithm to solve convex MINLP problems using a decomposition-based OA. The main question of this chapter is whether it is possible to solve convex optimization problems with a moderate number of MIP-OA problems. We focused on a quick generation of OA using linear programming instead of mixed-integer programming. The numerical results demonstrate that in average the algorithm requires only 2–3 MIP problems to be solved to reach convergence. However, it might be necessary to solve more MIP problems, if the problem is defined with nonlinear convex constraints that have many nonlinear terms. The number of MIP problems could be reduced further by performing additional cut generation methods like line-search and fix-and-refine. However, they increase the total runtime of the algorithm, since additional sub-problems need to be solved. Looking at the comparison of the algorithm

7. CONCLUSIONS

to other solvers, it is evident that the algorithm efficiency should be improved. There exist several possibilities to do this: (i) extract more feasible solution points from a MIP solver to generate more supporting hyperplanes per iteration; (ii) reduce the time for solving LP master problems and small sub-problems; (iii) add the implementation of parallel sub-problem solving.

Chapter 3 covers an OA approach for solving nonconvex MINLP problems. This chapter describes how to construct a piecewise nonconvex outer approximation of the feasible set defined by nonconvex constraint functions which are twice differentiable. We introduced the OA definition based on a so-called DC approach. However, the piecewise nonconvex outer approximation constructed using DC is unsatisfactory. If the algorithm adds too many partition points, then the MIP-OA problem is not solvable in a reasonable time. If the number of partition points is small, then the quality of the lower bound, provided by the MIP-OA problem, is unacceptable. The investigation, in fact, leads to new research questions on new ideas to define a tighter outer approximation for a nonconvex set. Another future research direction is to explore the strategies of handling the partitions. A possible partition strategy is to generate a predefined number of partitions and adapt them with respect to a reference point.

Chapter 4 investigates another MIP OA approach based on disjunctive cuts for nonconvex problems. In contrast to Chapter 2 and Chapter 3, the idea of the OA definition for DIOR1 is based on the multi-objective view on the original problem. For this algorithm, we define an OA by valid cuts in the resource space. These cuts are computed while generating an IA using CG. One of the research questions of this chapter is whether removing nondominated cones from OA is an efficient tool to solve optimization problems with many coupling constraints. This chapter concludes that disjunctive cuts used to eliminate nondominated cones are too weak to solve problems with many coupling constraints. The fact that OA is constructed while generating IA, gave us the question whether IA can be a more efficient technique than the OA-based approach to remove parts of a infeasible region. This question motivated the investigation reported in Chapter 4 which was focused on solving large-scale optimization problems.

7.2 Column Generation

The investigation on Column Generation covers two chapters in this thesis. Chapter 4 demonstrates an algorithm to compute a convex relaxation of the original problem using CG. The algorithm is based on the resource-constrained reformulation of the original problem. A research question of Chapter 4 is whether generation of feasible

(not necessarily optimal) points speeds up the convergence of the whole CG procedure. Even though the described procedure did not always solve the MINLP sub-problems to optimality, the numerical results conclude that solving the sub-problems took most of the algorithm time. This implies that branch-and-bound is not a well-suited method to solve these small MINLP sub-problems. The results illustrate the importance of a further investigation on acceleration techniques for CG. One of the options is to use a simple rounding heuristic to solve the sub-problems. This and other options were explored in Chapter 5.

Another question of Chapter 4 is whether a MIP inner approximation is an efficient approach to compute high-quality solution candidates for large-scale problems. This question was raised after concluding that the OA-based algorithm, DIOR1, is not a suitable method to solve problems with many coupling constraints. To explore the potential of the IA-based technique, we developed heuristic procedure DIOR2. The comparison to DIOR1 shows that DIOR2 is a faster heuristic procedure. Moreover, the heuristic approach, which divides supporting columns to remove the infeasible parts from IA, seems to provide a tighter bound than elimination of nondominated cones from the OA. Despite the fact that DIOR2 was able to compute high-quality solutions, the comparison with the BB solver suggested that its efficiency needs to be improved. As mentioned before, a drawback of the algorithm is that the CG procedure is slow. Moreover, the MIP-IA problem gets harder to solve when the number of iterations is growing. As a future research direction, one can elaborate column division idea to refine the IA, e.g. use several reference points to divide supporting columns.

Chapter 5 explores the options to accelerate the CG procedure. One of the questions of this chapter is to study the influence of the rounding heuristic and the Frank-Wolfe algorithm on the CG convergence. The experiments show that these methods indeed improve the convergence speed. Moreover, thanks to these acceleration methods, the bound computed by CG before starting the iterations of the main algorithm almost converges to the optimal solution of the convex relaxation. We have shown that it is more efficient to deal with the linear structure blocks in another way than the nonlinear blocks. The chapter concludes that integrating a linear block into the master problem is much more efficient than generating columns for them. We aimed to investigate the potential of projection to compute solution candidates and to look into capabilities of generating multiple solution candidates. We presented the heuristic CG-based algorithm that computes solution candidates by projecting a solution of the convex relaxation on to the feasible set. The results with DESS instances demonstrate that the heuristic algorithm is capable to compute various high-quality solutions. This pro-

7. CONCLUSIONS

vides an advantage for engineers to inspect several possibilities to plan and operate an energy system. An interesting finding is that the quality of the solution computed by the CG algorithm is similar to the solution quality computed by the tailored MIP approximation method [45]. Moreover, for several instances, the new method computes better solutions than the BB solver. However, the time measurements illustrated that the algorithm is slower than the existing approaches. An advantage of the CG heuristic algorithm is that it can be applied to any MINLP problem.

The algorithm presented in Chapter 5 generates most of the columns heuristically. This excludes the possibility to define an OA with valid cuts, as investigated in Chapter 4. Even though the algorithm computes some of the columns by a BB solver to guarantee the convergence of CG, the related cuts would not be sufficient to define a tight OA. We experimented with computing an OA using the IA, i.e. an OA defined using the existing columns. In this way, we wanted to speed up the CG convergence. However, this approach did not provide sufficient improvement of the convergence speed. Moreover, the Frank-Wolfe algorithm and rounding heuristic were more efficient acceleration techniques than using the Inner Approximation to find an Outer Approximation.

The numerical results of Chapter 4 and Chapter 5 demonstrate that the duality gap of most instances is large. A large duality gap might affect the solution quality. In particular, the numerical results in Chapter 5 show that the solution quality tends to decrease, when the duality gap increases. It sets a new research direction to reduce the duality gap. One of the options is block aggregation. This and several other ideas are discussed in [91]. The idea of aggregation consists of generating new columns regarding aggregated blocks and to include them into the IA master problem, in order to obtain a tighter convex relaxation (IA). By solving aggregated sub-problems, one can also generate new global constraints, which can also improve the objective function value of the OA master problem. In this thesis, we presented two options for the block definition: (i) automatic block identification (Section 1.3) and (ii) manual block definition (Chapter 5). Block aggregation can be more important when the blocks are identified in an automatic way, since we do not have the control over the block size. For instance, this type of identification may generate many blocks with a small size. Usually, such block sizes provide a large duality gap. When dealing with manually defined blocks, we can control block sizes in advance and may have a smaller duality gap.

Publications arising from this thesis

The investigation for the present thesis resulted in a number of publications. This appendix lists them sorted by the year of publication (oldest first) within each category.

A.1 Journal publications

- P. Muts, I. Nowak, and E. M. T. Hendrix. The decomposition-based outer approximation algorithm for convex mixed-integer nonlinear programming. *Journal of Global Optimization*, 77:75–96, 2020. doi:10.1007/s10898-020-00888-x
Impact factor JCR 2019: 1.805. Subject categories: Applied Mathematics - Q1 (56/261); Operations Research & Management Science - Q3 (42/83);
- P. Muts, I. Nowak, and E. M. T. Hendrix. On decomposition and multiobjective-based column and disjunctive cut generation for MINLP. *Optimization and Engineering*, 2020. doi:10.1007/s11081-020-09576-x
Impact factor JCR 2019: 1.829. Subject categories: Mathematics, Interdisciplinary Applications - Q2 (42/106); Operations Research & Management Science - Q2 (41/83); Multidisciplinary Engineering - Q2 (44/91).

A.2 Submitted journal publication

- P. Muts, S. Bruche, I. Nowak, O. Wu, E. M. T. Hendrix, and G. Tsatsaronis. A Column Generation Algorithm for Solving Energy System Planning Problems. *Submitted to Optimization and Engineering*, 2020
Impact factor JCR 2019: 1.829. Subject categories: Mathematics, Interdisciplinary Applications - Q2 (42/106); Operations Research & Management Science

A. PUBLICATIONS ARISING FROM THIS THESIS

- Q2 (41/83); Multidisciplinary Engineering - Q2 (44/91).

A.3 Publications in international conference proceedings

- P. Muts and I. Nowak. Towards Multi-tree Methods for Large-Scale Global Optimization. In H. L. Thi, H. Le, and T. P. Dinh, editors, *Optimization of Complex Systems: Theory, Models, Algorithms and Applications. WCGO 2019. Advances in Intelligent Systems and Computing*. Springer, Cham, 2019. doi:10.1007/978-3-030-21803-4_50;
- P. Muts, I. Nowak, and E. M. T. Hendrix. A Resource Constraint Approach for One Global Constraint MINLP. In O. Gervasi and et al., editors, *Computational Science and Its Applications – ICCSA 2020*. Springer, Cham, 2020. doi:10.1007/978-3-030-58808-3_43.



Other publications produced during the elaboration of this thesis

The research effort invested during the time span in which this thesis was elaborated produced some additional publications as the result of other research lines not included in the present dissertation. This appendix lists them sorted by the year of publication (oldest first).

B.1 Book chapter

- I. Nowak, P. Muts, and E. M. T. Hendrix. Multi-Tree Decomposition Methods for Large-Scale Mixed Integer Nonlinear Optimization. In J. Velásquez-Bermúdez, M. Khakifirooz, and M. Fathi, editors, *Springer Optimization and Its Applications*, pages 27–58. Springer International Publishing, 2019. doi:10.1007/978-3-030-22788-3_2.

B.2 Publication in international conference proceedings

- I. Nowak and P. Muts. Decomposition-based successive approximation methods for global optimization. In M. T. M. Emmerich, A. H. Deutz, S. Hille, and Y. Sergeyev, editors, *Proceedings LeGO – 14th International Global Optimization Workshop*. AIP Publishing, 2019. doi:10.1063/1.5089985.

Introducción

La programación no lineal de enteros mixtos es un campo de optimización importante y desafiante. Este tipo de problemas pueden contener variables continuas e enteras, así como restricciones lineales y no lineales. Esta clase de problemas tiene un papel fundamental en la ciencia y la industria, ya que proporcionan una forma precisa de describir fenómenos en diferentes áreas como ingeniería química y mecánica, cadena de suministro, gestión, etc. La mayoría de los algoritmos de última generación para resolver los problemas de programación no lineal de enteros mixtos no convexos están basados en los métodos de ramificación y acotación. El principal inconveniente de este enfoque es que el árbol de búsqueda puede crecer muy rápido impidiendo que el algoritmo encuentre una solución de alta calidad en un tiempo razonable. Una posible alternativa que evite la generación de grandes árboles consiste en hacer uso del concepto de descomposición para hacer que el procedimiento sea más manejable. La descomposición proporciona un marco general en el que el problema original se divide en pequeños subproblemas y sus resultados se combinan en un problema maestro más sencillo.

Esta tesis analiza los métodos de descomposición para la programación no lineal de enteros mixtos. El principal objetivo de esta tesis es desarrollar métodos alternativos al de ramificación y acotación, basados en el concepto de descomposición. Para la industria y la ciencia, es importante calcular una solución óptima, o al menos, mejorar la mejor solución disponible hasta ahora. Además, esto debe hacerse en un plazo de tiempo razonable. Por lo tanto, el objetivo de esta tesis es diseñar algoritmos

RESUMEN EN ESPAÑOL

eficientes que permitan resolver problemas de gran escala que tienen una aplicación práctica directa. En particular, nos centraremos en modelos que pueden ser aplicados en la planificación y operación de sistemas energéticos. En esta tesis se pueden distinguir dos líneas principales de investigación. La primera se ocupa de los métodos de Aproximación Externa (Outer Approximation), mientras que la segunda estudia un solución basada en el método de Generación de Columnas (Column Generation). En esta tesis investigamos y analizamos aspectos teóricos y prácticos de ambas ideas dentro del marco de la descomposición. El objetivo principal de este estudio es desarrollar métodos sistemáticos basados en la descomposición para resolver problemas de gran escala utilizando los métodos de Aproximación Externa y Generación de Columnas. En el capítulo 1 se introduce un concepto importante necesario para la descomposición. Este concepto consiste en una reformulación separable en bloques del problema de programación no lineal de enteros mixtos. En el capítulo 1 también se hace una descripción de los métodos mencionados anteriormente, incluyendo los de Ramificación y Acotación, además de otros conceptos clave que son necesarios para esta tesis, como por ejemplo los de Aproximación Interior, etc.

Los capítulos 2, 3 y 4 investigan el uso del concepto de Aproximación Externa. Específicamente, en el capítulo 2 se presenta un algoritmo de Aproximación Externa basado en descomposición para resolver problemas de programación no-lineales convexos enteros-mixtos, basados en la construcción de hiperplanos soporte para un conjunto factible. El capítulo 3 amplía el marco de aplicación de un algoritmo de Aproximación Externa basado en descomposición, a problemas de programación no lineales no convexos enteros mixtos, introduciendo una Aproximación Externa convexa por partes de un conjunto factible no convexo. Otra perspectiva de la definición de Aproximación Externa para problemas no convexos se considera en el capítulo 4, que presenta un algoritmo de Refinamiento Interno y Externo basado en descomposición, que construye una Aproximación Externa al mismo tiempo que calcula la Aproximación Interna usando Generación de Columnas. La Aproximación Externa usada en el algoritmo de Refinamiento Interno y Externo se basa en la visión multiobjetivo de la denominada versión recursos restringidos del problema original.

Dos capítulos están dedicados a la Generación de Columnas. En el capítulo 4 se presenta un algoritmo de Generación de Columnas para calcular una Aproximación Interna del problema original. Además se describe un algoritmo heurístico basado en particiones que usa un refinamiento de la Aproximación Interna. El capítulo 5 analiza varias técnicas de aceleración para la Generación de Columnas, donde se describe un algoritmo heurístico general basado en la Generación de Columnas, que puede generar

varias soluciones candidatas de alta calidad.

El capítulo 6 contiene una breve descripción de la implementación en Python de DECOGO (software de programación no lineal de enteros mixtos).

Resumen del capítulo 1

Este capítulo presenta varios conceptos clave que se utilizan en esta tesis. Describe una formulación general y separable por bloques de un problema de programación no lineal entero mixto, que es una idea fundamental para la descomposición. Además, demostramos cómo reformular un problema general de programación no lineal entero mixto como uno separable por bloques. El método se basa en una identificación natural de estructura de bloques. Para determinar los bloques, se construye un grafo de adyacencia Hessiana del problema original, entonces calculamos los componentes conectados de este gráfico, que definen los bloques y en base a esta información de bloque se reformula el modelo original en forma de bloques separables. Este método se usa para los experimentos numéricos de los capítulos 2, 3 y 4. Este procedimiento no se usa en los experimentos del capítulo 5, donde la estructura de bloques se determina de forma manual. En el capítulo 1 también se describe la gran colección de problemas de optimización MINLPLib, que se utilizan en los resultados numéricos de los capítulos 2, 3 y 4.

En esta tesis, los resultados numéricos de los métodos desarrollados se comparan con los obtenidos por algoritmos de Ramificación y Acotación. En este capítulo, describimos brevemente este método y presentamos un pequeño ejemplo ilustrativo. El algoritmo de Ramificación y Acotación divide recursivamente el problema original en subproblemas disjuntos más pequeños hasta que se encuentra y verifica la solución óptima. Estos subproblemas se almacenan en una estructura de árbol. La idea de acotar consiste en podar los nodos del árbol (subproblemas) que no contienen una solución óptima. Sin embargo, el método de Ramificación y Acotación requiere muchos recursos de computación y largos periodos de computo. Dependiendo del tamaño del problema, el número de nodos del árbol puede ser demasiado grande para la capacidad de la memoria de la computadora y dificulta enormemente que el algoritmo llegue a calcular un límite inferior suficientemente preciso.

En este capítulo, presentamos los conceptos necesarios para los métodos de Aproximación Externa y de Generación de columnas. Para la Aproximación Externa, explicamos cómo se puede construir para los diferentes tipos de problemas. Si un problema de programación no lineal de enteros mixtos se define mediante funciones diferenciables

RESUMEN EN ESPAÑOL

convexas, entonces, para aproximar el conjunto factible no lineal, es suficiente calcular linealizaciones de funciones de restricción en cualquier punto. Tales linealizaciones forman planos de corte válidos y proporcionan una aproximación externa del conjunto factible no lineal. Si un problema está definido por restricciones no convexas, normalmente se emplea una aproximación externa convexa del conjunto factible no convexo. La construcción de una aproximación externa convexa a menudo se denomina proceso de convexificación. Se basa en explotar las propiedades de estructuras matemáticas particulares, por ejemplo, funciones cóncavas, cuadráticas, bilineales, fraccionarias.

Para la Generación de Columnas, presentamos los conceptos de programa con recursos restringidos y de Aproximación Interna. Una reformulación de recursos restringidos para el problema de bloques separables consiste en introducir nuevas variables de recursos en el espacio transformado usando restricciones globales lineales. Demostramos que un programa con recursos restringidos es equivalente al original. Dicha reformulación reduce el número de variables al número de restricciones globales. Además, muchas de las restricciones globales son dispersas. Demostramos cómo se utiliza un programa con recursos restringidos desde un punto de vista multiobjetivo. Construimos un frente de Pareto minimizando simultáneamente todos los recursos distintos de cero del subproblema con recursos restringidos. El frente de Pareto proporciona una base para un refinamiento de la Aproximación Externa utilizando el algoritmo de Refinamiento Interno y Externo basado en la descomposición que se presenta en el capítulo 4. Otro concepto esencial para la Generación de Columnas es la Aproximación Interna. Para ello, definimos una relajación convexa del problema original. Se aproxima por la envolvente convexa de los puntos factibles (internos) resolviendo pequeños subproblemas de programación no lineales enteros mixtos. La envolvente convexa se formula mediante una Aproximación Interna basada en programación lineal en el espacio original, o por su variante con recursos restringidos.

En este capítulo, también discutimos métodos multi-árbol en contrario del árbol único. Un método multi-árbol resuelve una secuencia de problemas de programación lineales enteros mixtos. En otras palabras, no reutiliza los árboles de ramificación y acotación de iteraciones anteriores. Los métodos de árbol único utilizan el mismo árbol de ramificación y acotación en todo el proceso de resolución.

Resumen del capítulo 2

La Aproximación Externa es un método de aproximación sucesiva para resolver un problema de optimización. A diferencia de los métodos de ramificación y acotación,

este método no utiliza un único árbol de búsqueda global. En cambio, construye una secuencia de árboles resolviendo problemas de programación enteros mixtos. Sin embargo, resolver una gran cantidad de problemas de programación enteros mixtos ralentiza la convergencia del método. El objetivo principal de este capítulo es diseñar un algoritmo que sea capaz de resolver problemas de programación no lineal enteros mixtos convexos con un número pequeño de problemas de programación enteros mixtos utilizando Aproximación Externa.

Este capítulo describe un algoritmo de Aproximación Externa basado en descomposición multi-árbol para problemas de programación no lineales enteros mixtos convexos. El algoritmo construye una Aproximación Externa de problemas de programación enteros mixtos generando hiperplanos soporte. Estos hiperplanos se obtienen mediante la linealización de funciones de restricción no lineales. La diferencia clave con respecto a todos los demás métodos de Aproximación Externa es que el algoritmo presentado utiliza una generación de planos de corte basada en descomposición, es decir, los planos de corte resultantes de la linealización se construyen simplemente resolviendo pequeños subproblemas en paralelo.

Una de las preguntas científicas que se plantean en este capítulo es: ¿puede reducirse el número de problemas de programación enteros mixtos que tienen que ser resueltos para alcanzar la convergencia del algoritmo de descomposición. Por lo tanto, este capítulo presenta dos versiones del algoritmo: básica y mejorada. La versión básica utiliza una proyección para generar las linealizaciones, es decir, los puntos no factibles se proyectan sobre el conjunto factible resolviendo pequeños subproblemas. Además, utiliza solo un problema maestro de programación entero mixto para la definición de la Aproximación Externa. El inconveniente de este algoritmo es que el número de problemas de programación de enteros mixtos a resolver puede ser grande. Esbozamos una prueba de convergencia para la versión básica del algoritmo de Aproximación Externa basado en descomposición. Para ello, demostramos varias propiedades importantes del algoritmo. Estas propiedades son: (i) exclusión de la solución en la siguiente iteración después de generar nuevos hiperplanos soporte; (ii) el límite de una subsecuencia convergente generada por el algoritmo pertenece al conjunto factible del problema original.

Para reducir el número de problemas de programación enteros mixtos a resolver y generar rápidamente una Aproximación Externa de alta calidad, presentamos un Algoritmo mejorado de Aproximación Externa basado en Descomposición de dos fases multi-árbol. En una primera (fase de programación lineal), el algoritmo inicializa una Aproximación Externa que resuelve un problema maestro de programación lineal. En la segunda fase (fase de programación entero mixto), el algoritmo refina una Aprox-

RESUMEN EN ESPAÑOL

mación Externa que resuelve un problema maestro de programación entero mixto. El problema maestro final de Aproximación Externa de programación entero mixto es una reformulación del problema original. A diferencia de la versión básica del algoritmo de Aproximación Externa basado en descomposición, la versión mejorada utiliza procedimientos de búsqueda lineal y de corrección y refinamiento para generar hiperplanos adicionales. La pregunta científica que se plantea es: ¿cómo estos métodos adicionales reducen el número de problemas de programación enteros mixtos que hay que resolver y cual es su influencia en el rendimiento general del algoritmo?. También se presenta una demostración de la convergencia del algoritmo mejorado.

Los resultados numéricos demostraron que, en promedio, el algoritmo presentado, aproximadamente, solo requiere resolver entre 2 y 3 problemas de programación enteros mixtos para alcanzar la convergencia. Este comportamiento se debe a los planos de corte generados en la fase de programación lineal. Además, los resultados muestran que el número promedio de problemas de programación enteros mixtos es independiente del tamaño del problema. Además de esto, el tiempo dedicado a la resolución de subproblemas es mayor que el tiempo necesario para resolver problemas de programación lineal y de programación enteros mixtos.

Se han probado cuatro variantes del algoritmo de Aproximación Externa basado en la descomposición sobre un conjunto de problemas de programación no lineal enteros mixtos convexos, con el fin de determinar si los métodos de generación de planos de corte adicionales como la búsqueda lineal y la corrección y refinamiento reducen aún más el número de problemas de programación enteros mixtos a resolver. Los experimentos han demostrado que en todos los casos, el número promedio de problemas de programación enteros mixtos podría reducirse aún más. Sin embargo, los nuevos métodos aumentan el tiempo de ejecución total del algoritmo, ya que es necesario resolver subproblemas adicionales.

El rendimiento del algoritmo de Aproximación Externa basado en descomposición se ha comparado con SCIP (software de programación no lineal entero mixto basado en el método de ramificación acotación) y con el método clásico de Aproximación Externa. Aunque el algoritmo de Aproximación Externa basada en descomposición se ha implementado en Python, demuestra ser incluso más rápido para algunos (9%) de los problemas que una implementación avanzada como SCIP. La comparación con la Aproximación Externa clásica demuestra que la Aproximación Externa basada en descomposición reduce el número de problemas de programación enteros mixtos y es más eficiente en los casos en los que el problema debe resolverse con un gran número de problemas de programación enteros mixtos.

De la comparación con otros métodos que presentamos, es evidente que el algoritmo puede mejorarse aún más. Existen varias posibilidades para hacer esto: (i) extraer más puntos de solución factibles del software de programación enteros mixtos para generar más hiperplanos soporte en cada iteración; (ii) reducir el tiempo necesario para resolver los problemas maestros de programación lineal y los pequeños subproblemas; (iii) incorporar una implementación paralela de la resolución de los pequeños subproblemas.

Resumen del capítulo 3

El objetivo de este capítulo es extender el concepto de Aproximación Externa basado en descomposición, descrito en el capítulo 2, para resolver problemas de programación no lineal enteros mixtos no convexos. El objetivo es diseñar un método de Aproximación Externa basado en descomposición que resuelva problemas de optimización definidos por restricciones que pueden ser no convexas.

En el capítulo 2, se construye una Aproximación Externa poliédrica calculando planos de corte de linealización válidos en puntos de prueba. Sin embargo, cuando se trata de funciones de restricción no convexas, los planos de corte de linealización pueden no ser válidos. Para construir una Aproximación Externa poliédrica de un conjunto factible no convexo, normalmente se emplean subestimadores y sobreestimadores convexos de funciones no convexas. La mayoría de los métodos que permiten definir subestimadores y sobreestimadores convexos de funciones no convexas aprovechan estructuras matemáticas, por ejemplo, términos bilineales, etc. La desventaja de estos métodos es que sólo se pueden utilizar para clases especiales de funciones. Una de las preguntas científicas de este capítulo es: ¿cómo construir subestimadores convexos ajustados para funciones arbitrarias?

Existen varios métodos que resuelven problemas de programación no lineal enteros mixtos no convexos mediante la construcción de una Aproximación Externa convexa a trozos, de un conjunto factible no convexo. Estos métodos se basan en estructuras matemáticas, como se describió anteriormente. El reto de estos métodos consiste en obtener una partición eficaz del espacio de variables, de modo que la aproximación resultante de la programación de enteros mixtos aún se pueda resolver en un tiempo razonable. Algunos de estos métodos refinan la aproximación de la programación de enteros mixtos a trozos agregando de forma adaptativa nuevos puntos de partición. Nuestra pregunta científica es: ¿cómo agregar de manera efectiva los puntos de partición, para definir una Aproximación Externa convexa a tramos ajustada, que finalmente, en un tiempo razonable, obtenga una solución a un problema de programación

de enteros mixtos?.

En este capítulo, extendemos el algoritmo de Aproximación Externa basado en descomposición para problemas de programación no lineal de enteros mixtos no convexos. Para las funciones de restricción no convexas, el método utiliza sub-estimadores poliédricos por partes. La construcción de sub-estimadores poliédricos por partes se basa en el llamado método de optimización DC (Diferencias de Funciones Convexas). Al igual que la Aproximación Externa basada en descomposición para problemas convexos, es un método de dos fases de multi-árbol que genera planos de corte y puntos de partición resolviendo subproblemas de proyección de baja dimensión. El algoritmo refina la relajación de la programación de enteros mixtos por trozos calculando planos de corte para las restricciones convexas y agregando de forma adaptativa nuevos puntos de partición para las variables que aparecen en las restricciones no convexas. La formulación de un problema maestro de programación entero mixto se basa en un conjunto ordenado especial de tipo 2 (restricciones SOS2 (Special Ordered Set 2)). Esta formulación involucra variables binarias adicionales, lo que hace que el problema maestro de programación entero mixto sea más difícil de resolver. Para evitar este problema, el algoritmo utiliza varios métodos para reducir el número de puntos de partición. Uno de ellos consiste en la eliminación de algunos puntos del conjunto de puntos de partición. Otro método consiste en una optimización basada en una acotación rigurosa. Esta técnica resuelve la relajación de un problema de programación entero mixto minimizando y maximizando los valores de las variables individualmente. De esta forma, el algoritmo puede reducir los límites de las variables y eliminar puntos de partición.

Un experimento con un ejemplo a pequeña escala ha demostrado que la relajación convexa a trozos construida utilizando una optimización DC no es satisfactoria. Además, la velocidad de convergencia del algoritmo es pobre. La razón principal de una velocidad de convergencia lenta es que el problema maestro de programación entero mixto es desde un punto de vista computacional muy exigente. Si el algoritmo agrega demasiados puntos de partición, entonces el problema maestro de programación entero mixto no se puede resolver en un tiempo razonable. Si el número de puntos de partición es pequeño, entonces la calidad del límite inferior, proporcionada por el problema maestro de programación entero mixto, es inaceptable. La investigación, de hecho, conduce a preguntas científicas sobre nuevas ideas para definir una Aproximación Externa convexa más rigurosa para un conjunto no convexo. Una línea de investigación futura consiste en explorar las estrategias de manejo de particiones. Una posible estrategia de partición consiste en generar un número predefinido de particiones y adaptarlas con respecto a un punto de referencia.

Resumen del capítulo 4

En el capítulo 4, presentamos un algoritmo de Refinamiento Interno y Externo basado en descomposición multi-árbol para resolver problemas de programación no lineal enteros mixtos no convexos. El método se basa en la denominada reformulación del problema original con recursos restringidos. Este método define las variables de recursos en función de las limitaciones globales del problema original. Esta visión de recursos restringidos puede ser prometedora, ya que, para algunos problemas, el número de restricciones globales que conectan los subproblemas puede ser significativamente menores que el tamaño del subproblema. Basándonos en este método, calculamos Aproximaciones Internas y Externas de relajación convexa del problema original. Nuestra pregunta científica es: ¿Es adecuada la Aproximación Externa para resolver problemas no convexos con una gran cantidad de restricciones emparejadas? Otra pregunta es: ¿Qué calidad de la solución se puede lograr cuando se usa la aproximación interna?

Al igual que en el capítulo 2 y el capítulo 3, investigamos el potencial del concepto de descomposición en contraste con la aplicación del algoritmo de ramificación y acotación. Para hacerlo, desarrollamos dos algoritmos de Refinamiento Interno y Externo basados en descomposición multi-árbol. Ambos métodos se basan en la idea de reducción de la dimensión utilizando el concepto de un programa con recursos restringidos. Al igual que un algoritmo de Aproximación Externa basado en descomposición, un algoritmo de Refinamiento Interno y Externo basado en descomposición es un método de dos fases. En la primera etapa, ambos algoritmos calculan una aproximación de programación lineal del programa con recursos restringidos con respecto a columnas no dominadas utilizando dos métodos: (i) método de sub-gradiente y (ii) generación de columnas. Nuestra pregunta científica aquí es: ¿Se puede mejorar la convergencia del procedimiento Generación de Columnas?. En particular, nos centramos en la generación de puntos factibles en lugar de puntos de solución óptimos de subproblemas de programación no lineal enteros mixtos.

En la segunda etapa, ambos algoritmos calculan una aproximación de programación enteros mixtos del programa con recursos restringidos agregando planos de corte disyuntivos. La diferencia de los métodos consiste en cómo se define una aproximación de programación de enteros mixtos. El primer algoritmo utiliza una Aproximación Externa de programación entero mixto. En contraste con el capítulo 2 y el capítulo 3, definimos una Aproximación Externa mediante planos de corte válidos en el espacio de recursos. Estos planos de corte se calculan mientras se genera una Aproximación Interna mediante la Generación de Columnas. El algoritmo mejora de forma iterativa una

Aproximación Externa al eliminar las regiones no dominadas usando una búsqueda lineal multi-objetivo. Esbozamos una demostración de la convergencia de este algoritmo. La demostración es similar a la del algoritmo de Aproximación Externa basado en descomposición presentado en el capítulo 2. Además, el algoritmo utiliza un concepto de complemento del frente de Pareto (el frente de Pareto extendido). En este método, nos centramos en la cuestión de si dicho método puede aplicarse a problemas de gran escala, en particular con un gran número de restricciones emparejadas. El segundo algoritmo utiliza una Aproximación Interna de programación entero mixto. Para eliminar las partes de la región posiblemente infactible, agrega heurísticamente planos de corte disyuntivos utilizando un subproblema de recursos restringidos. La pregunta científica es: ¿Puede el algoritmo calcular soluciones de alta calidad del problema original usando la Aproximación Interna de programación enteros mixtos. Además, nos concentramos en la cuestión de: ¿Qué ganancia de rendimiento computacional puede proporcionar este algoritmo en comparación con otros métodos existentes?

Los resultados numéricos obtenidos con el algoritmo de Generación de Columnas para calcular una relajación convexa del problema original, demuestran que la resolución de los subproblemas acapara la mayor parte del tiempo del algoritmo. En estos experimentos, investigamos la posibilidad de usar una terminación temprana del software de ramificación y acotación para generar nuevas columnas. Este método no pudo proporcionar una mejora significativa de la velocidad de convergencia. Esto implica que el método de Ramificación y Acotación no es adecuado para resolver estos pequeños subproblemas de programación no lineales enteros mixtos. Los resultados ilustraron la importancia de una mayor investigación sobre las técnicas de aceleración para la Generación de Columnas. Una de las opciones consiste en utilizar una simple heurística de redondeo para resolver los subproblemas. Esta y otras opciones se exploran en el capítulo 5.

Los experimentos con el algoritmo de Refinamiento Interno y Externo de descomposición basado en una Aproximación Externa de programación entero mixto han demostrado que no es adecuado para problemas de optimización con muchas restricciones emparejadas. Los planos de corte disyuntivos utilizados para eliminar conos no dominados de Aproximación Externa son demasiado débiles, ya que el algoritmo primero selecciona celdas que no mejoran el valor de la función objetivo de un problema maestro de Aproximación Externa. El hecho de que la Aproximación Externa se construya mientras se genera la Aproximación Interna, nos lleva a la pregunta de si la Aproximación Interna puede ser una técnica más eficiente para eliminar partes de una región no factible. Esta cuestión motivó la investigación del capítulo 5, que se centra en la

resolución de problemas de optimización de gran escala.

Para investigar el potencial de la técnica basada en una Aproximación Interna, definimos una Aproximación Interna de programación enteros mixtos basada en la separación de columnas. Para la separación, utilizamos columnas activas, es decir, columnas con un peso distinto de cero. Las columnas se dividen utilizando un punto de referencia obtenido después de resolver el subproblema de recursos restringidos. La comparación del algoritmo de refinamiento basado en la Aproximación Interna con el algoritmo de refinamiento basado en la Aproximación Externa demostró que el basado en Aproximación Interna es un procedimiento heurístico más rápido. Además, este método heurístico parece proporcionar un límite más riguroso que la eliminación de los conos no dominados de la Aproximación Externa. A pesar de que este algoritmo fue capaz de calcular soluciones de alta calidad, la comparación con el software de ramificación y acotación sugirió que su eficiencia se puede mejorar. Como se mencionó anteriormente, un inconveniente del algoritmo es que el procedimiento Generación de Columnas es lento. Además, el problema maestro de programación entero mixto tiende a ser más difícil de resolver cuando el número de iteraciones crece. Como dirección de investigación futura, se puede elaborar la idea de separación de columnas para refinar la Aproximación Interna utilizando, por ejemplo, varios puntos de referencia para seleccionar columnas activas.

Resumen del capítulo 5

Los resultados numéricos del algoritmo de Refinamiento Interno y Externo basado en descomposición heurística, presentado en el capítulo 4, muestran que la mayor parte del tiempo de computación se dedica a resolver subproblemas de programación no lineales enteros mixtos de baja dimensión que se generan en el procedimiento de Generación de Columnas. Esto se debe al hecho de que la Generación de Columnas se basa en la resolución de pequeños problemas de programación no lineales enteros mixtos utilizando métodos de Ramificación y Acotación. Además, muy a menudo, el algoritmo genera una columna que ya se generó antes, lo que hace que la Generación de Columnas sea ineficiente. Para abordar estos problemas, presentamos un algoritmo heurístico basado en descomposición multi-árbol. Este algoritmo es una extensión del algoritmo de Generación de Columnas presentado en el capítulo 4. Aunque la heurística elaborada de Generación de Columnas se puede aplicar a cualquier problema de programación no lineal entero mixto, en este capítulo nos centramos en los problemas que surgen del modelado de sistemas descentralizados de suministro de energía .

RESUMEN EN ESPAÑOL

Desarrollamos varias estrategias para acelerar la Generación de Columnas. Nos centramos en una heurística de redondeo simple para resolver subproblemas de programación no lineales enteros mixtos y en el algoritmo de Frank-Wolfe. El algoritmo de Frank-Wolfe calcula una relajación convexa penalizando las restricciones globales. El problema maestro con la función objetivo penalizada es un problema cuadrático. El método Frank-Wolfe no resuelve este problema maestro sino que solo resuelve pequeños subproblemas, ya que todas las restricciones globales están incluidas en la función de penalización. Estos subproblemas se resuelven mediante una heurística de redondeo. Nuestra pregunta científica es: ¿Ayudan estas técnicas a acelerar el procedimiento de Generación de Columnas y a generar más columnas de alta calidad? Otra cuestión que se plantea es: ¿Cómo garantizar que el algoritmo pueda calcular soluciones de alta calidad del problema original? Investigamos el potencial de proyectar la solución de relajación convexa en el conjunto factible, similar a la idea de "Bomba de Factibilidad (Feasibility Pump)". Además, exploramos la posibilidad de generar varias soluciones candidatas mientras resolvemos el problema de proyección. Este método puede aumentar la probabilidad de encontrar mejores candidatos para la solución. Además, puede ayudar a evitar que el algoritmo no pueda encontrar ninguna solución factible.

Para acelerar la Generación de Columnas, también analizamos las propiedades de los modelos de sistemas descentralizados de suministro de energía. Muchos de los submodelos (blocks) de esos modelos están definidos por restricciones lineales y variables continuas. En este capítulo, estos submodelos se fusionan en un solo submodelo lineal. De hecho, el submodelo lineal es un politopo definido por sus puntos extremos (vértices). Si el tamaño del submodelo es grande, entonces la convergencia de la Generación de Columnas podría ser lenta, ya que generaría todos los vértices del politopo. Aquí nuestra pregunta científica es: ¿Cómo trabajar con este submodelo lineal de manera eficiente. Elaboramos la idea de integrar este submodelo lineal en el problema maestro de Aproximación Interna. De esta forma, evitamos el cálculo de columnas en el submodelo lineal.

El algoritmo de Generación de Columnas del capítulo 5 inicializa las columnas usando un algoritmo de subgradiente, similar al del capítulo 4. Además, el algoritmo de Generación de Columnas utiliza un problema maestro de proyección de recursos para obtener la primera solución candidata. De esta manera, el problema de Aproximación Interna de programación lineal es factible. El algoritmo continúa la inicialización de la Aproximación Interna mediante la ejecución del procedimiento de Frank-Wolfe. Luego, el algoritmo en turno ejecuta la Generación de Columnas tradicional y ejecuta el algoritmo heurístico para resolver generar soluciones. El procedimiento de Generación de

Columnas se ejecuta usando un subconjunto de submodelos (bloques) que está basado en el denominado costo reducido. Sin embargo, para garantizar la convergencia de este procedimiento seguimos ejecutando la Generación de Columnas para todos los submodelos. El algoritmo heurístico se basa en proyectar la solución del problema maestro de Aproximación Interna de programación lineal sobre el conjunto factible definido por restricciones globales y enteras. Este problema se modela como un problema de programación lineal entero mixto, el cual genera varias soluciones candidatas.

Los experimentos realizados demostraron que la heurística de redondeo y el algoritmo de Frank-Wolfe mejoran la velocidad de convergencia de la Generación de Columnas. Además, gracias a estos métodos de aceleración, el límite calculado por la Generación de Columnas antes de ejecutar las iteraciones del algoritmo principal casi converge a la solución óptima de la relajación convexa. Hemos demostrado que es más eficiente trabajar con los submodulos lineales de forma diferente a la utilizada para los submodulos no lineales. El capítulo concluye que integrar un bloque lineal en el problema maestro es mucho más eficiente que generar columnas para ellos. Los resultados numéricos demuestran que el algoritmo heurístico de Generación de Columnas con la proyección es una herramienta capaz de calcular varias soluciones de alta calidad. Esto proporciona una ventaja para que los ingenieros inspeccionen varias posibilidades para planificar y operar un sistema de energía. El resultado interesante es que la calidad de la solución calculada por el algoritmo de Generación de Columnas fue similar a la calidad de la solución calculada por un método personalizado de aproximación de programación entero mixto. Además, para varios casos, El método nuevo obtuvo mejores soluciones que el software de Ramificación y Acotación. Sin embargo, las medidas de tiempo de ejecución mostraron que el algoritmo es más lento que los métodos existentes. Una ventaja del algoritmo heurístico de Generación de Columnas es que se puede aplicar a cualquier problema de programación no lineal entero mixto.

El algoritmo presentado en el capítulo 5 genera la mayoría de las columnas de forma heurística. Esto excluye la posibilidad de definir una Aproximación Externa usando planos de corte válidos, tal como se describe en el capítulo 4. Aunque el algoritmo calcula algunas de las columnas mediante un software de Ramificación y Acotación para garantizar la convergencia de la generación de columnas, los planos de corte relacionados no serían suficientes para definir una Aproximación Externa rigurosa. Experimentamos con el cálculo de una Aproximación Externa utilizando la Aproximación Interna, es decir, una Aproximación Externa definida utilizando las columnas existentes. De esta forma, queríamos acelerar la convergencia de la Generación de Columnas. Sin embargo, este método no proporcionó una mejora suficiente de la velocidad de conver-

gencia. Además, el algoritmo de Frank-Wolfe y la heurística de redondeo son técnicas de aceleración más eficientes que la técnica de usar la Aproximación Interna para encontrar una Aproximación Externa.

Los resultados numéricos del capítulo 4 y el capítulo 5 demostraron que la brecha de dualidad es grande para la mayoría de los problemas ejemplo. Una gran brecha de dualidad puede afectar a la calidad de la solución. En particular, los resultados numéricos del capítulo 5 mostraron que la calidad de la solución tiende a ser ligeramente peor cuando aumenta la brecha de dualidad. Estos resultados establecen una nueva línea de investigación que permita reducir la brecha de dualidad. Una de las opciones es la agregación de submodelos. La idea es generar nuevas columnas de submodelos agregados e incluirlas en el problema maestro de Aproximación Interior con el fin de obtener una relajación convexa más rigurosa (Aproximación Interior). Al resolver subproblemas agregados, también se pueden generar nuevas restricciones globales, que también pueden mejorar el valor de la función objetivo del problema maestro de Aproximación Externa. En esta tesis, presentamos dos opciones de definición de submodelos: (i) identificación automática de submodelos y (ii) definición manual de submodelos. La agregación de submodelos puede ser más importante cuando los submodelos se identifican de forma automática, ya que no tenemos el control sobre el tamaño del submodelo. Por ejemplo, este tipo de identificación puede generar muchos submodelos de pequeño tamaño, que generalmente proporcionan una gran brecha de dualidad. Cuando se trata de submodelos definidos manualmente, podemos controlar el tamaño de los submodelos de antemano y tener una brecha de dualidad más pequeña.

Resumen del capítulo 6

En este capítulo, hacemos una breve descripción del software DECOGO (**D**ecomposition-based **G**lobal **O**ptimizer). DECOGO es una estructura de programación orientada a objetos para resolver problemas de programación no lineales enteros mixtos convexos y no convexos. DECOGO está escrito en Python y hace uso de Pyomo, que es una colección de paquetes de software de Python para formular modelos de optimización. Pyomo admite un diseño orientado a objetos para la definición del modelo. La funcionalidad de Pyomo es flexible y se puede ampliar fácilmente. Por ejemplo, se implementó la reformulación automática de un problema general de programación no lineal entero mixto utilizando expresiones Pyomo. El diseño del software se basa en dividir todos los módulos en subpaquetes conectados lógicamente. Estos subpaquetes tienen los siguientes objetivos: (i) recopilar la implementación de algoritmos y módulos

para administrar la configuración y los resultados del software; (ii) recopilar todos los módulos para almacenar un modelo original separable en bloques; (iii) recopilar todos los módulos que crean y gestionan todos los problemas maestro y subproblemas y datos de aproximación (lista de columnas, planos de corte de linealización); (iv) recopilar módulos que definan estructuras de datos de los submodelos. Este capítulo presenta una descripción general de las clases que se implementaron en el software. Los algoritmos presentados en los capítulos 2 al 5 se implementaron dentro del software DECOGO.

Bibliography

- [1] T. Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1:1–41, 2009. doi:10.1007/s12532-008-0001-1.
- [2] T. Achterberg, T. Berthold, T. Koch, and K. Wolteri. Constraint Integer Programming: A New Approach to Integrate CP and MIP. In L. Perron and M. A. Trick, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. doi:10.1007/978-3-540-68155-7_4.
- [3] C. S. Adjiman, I. P. Androulakis, C. D. Maranas, and C. A. Floudas. A global optimization method, α BB, for process design. *Computers & Chemical Engineering*, 20:S419–S424, 1996. doi:10.1016/0098-1354(96)00080-4.
- [4] F. A. Al-Khayyal and J. E. Falk. Jointly Constrained Biconvex Programming. *Mathematics of Operations Research*, 8:273–286, 1983. doi:10.1287/moor.8.2.273.
- [5] B. Bahl, S. Goderbauer, F. Arnold, P. Voll, M. Lübbecke, A. Bardow, and A. M. C. A. Koster. DESSLib – Benchmark Instances for Optimization of Decentralized Energy Supply Systems. Technical report, RWTH Aachen University, 2016. URL: <http://www.math2.rwth-aachen.de/DESSLib/>.
- [6] E. M. L. Beale and J. J. H. Forrest. Global optimization using special ordered sets. *Mathematical Programming*, 10:52–69, 1976. doi:10.1007/bf01580653.

BIBLIOGRAPHY

- [7] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24:597–634, 2009. doi:10.1080/10556780903087124.
- [8] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962. doi:10.1007/bf01386316.
- [9] M. Bergner, M. E. Lübbecke, and J. T. Witt. A Branch-Price-and-Cut Algorithm for Packing Cuts in Undirected Graphs. *ACM Journal of Experimental Algorithmics*, 21:1–16, 2016. doi:10.1145/2851492.
- [10] D. E. Bernal, Q. Chen, F. Gong, and I. E. Grossmann. Mixed-Integer Nonlinear Decomposition Toolbox for Pyomo (MindtPy). In M. R. Eden, M. G. Ierapetritou, and G. P. Towler, editors, *13th International Symposium on Process Systems Engineering (PSE 2018)*. Elsevier, 2018. doi:10.1016/B978-0-444-64241-7.50144-0.
- [11] J. Bisschop and A. Meeraus. On the development of a general algebraic modeling system in a strategic planning environment. In *Mathematical Programming Studies*, pages 1–29. Springer Berlin Heidelberg, 1982. doi:10.1007/bfb0121223.
- [12] M. Bodur, S. Ahmed, N. Boland, and G. L. Nemhauser. Decomposition of loosely coupled integer programs: A multiobjective perspective. *Optimization Online*, 2016. URL: http://www.optimization-online.org/DB_FILE/2016/08/5599.pdf.
- [13] P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5:186–204, 2008. doi:10.1016/j.disopt.2006.10.011.
- [14] R. Borndörfer, A. Löbel, M. Reuther, T. Schlechte, and S. Weider. Rapid Branching. *Public Transport*, 5:3–23, 2013. doi:10.1007/s12469-013-0066-8.
- [15] R. Burlacu, B. Geißler, and L. Schewe. Solving mixed-integer nonlinear programmes using adaptively refined mixed-integer linear programmes. *Optimization Methods and Software*, 35:37–64, 2020. doi:10.1080/10556788.2018.1556661.
- [16] M. R. Bussieck, A. S. Drud, and A. Meeraus. MINLPLib – A Collection of Test Models for Mixed-Integer Nonlinear Programming. *INFORMS Journal on Computing*, 15:114–119, 2003. doi:10.1287/ijoc.15.1.114.15159.

-
- [17] M. R. Bussieck and A. Meeraus. General Algebraic Modeling System (GAMS). In *Applied Optimization*, pages 137–157. Springer US, 2004. doi:10.1007/978-1-4613-0215-5_8.
 - [18] M. R. Bussieck and S. Vigerske. MINLP Solver Software, 2014. URL: <https://www.math.hu-berlin.de/~stefan/minlpsoft.pdf>.
 - [19] F. Ceccona, R. Baltean-Lugojana, M. L. Bynumb, C. Lia, and R. Misener. GALINI: An extensible mixed-integer quadratically-constrained optimization solver. *Optimization Online*, 2021. URL: http://www.optimization-online.org/DB_FILE/2021/01/8207.pdf.
 - [20] C. Coey, M. Lubin, and J. P. Vielma. Outer approximation with conic certificates for mixed-integer convex problems. *Mathematical Programming Computation*, 12:249–293, 2020. doi:10.1007/s12532-020-00178-3.
 - [21] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *Computational Journal*, 8:250–255, 1965. doi:10.1093/comjnl/8.3.250.
 - [22] C. D’Ambrosio, A. Frangioni, L. Liberti, and A. Lodi. A storm of feasibility pumps for nonconvex MINLP. *Mathematical Programming*, 136:375–402, 2012. doi:10.1007/s10107-012-0608-x.
 - [23] G. B. Dantzig and P. Wolfe. Decomposition Principle for Linear Programs. *Operations Research*, 8:101–111, 1960. doi:10.1287/opre.8.1.101.
 - [24] G. B. Dantzig and P. Wolfe. The decomposition algorithm for linear programs. *Econometrica*, 29:767–778, 1961. doi:10.2307/1911818.
 - [25] J. Desrosiers and M. E. Lübbecke. A Primer in Column Generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 1–32. Springer-Verlag, 2005. doi:10.1007/0-387-25486-2_1.
 - [26] J. Desrosiers and M. Lübbecke. Branch-price-and-cut algorithms. In J. Cochran, L. Cox, P. Keskinocak, J. Kharoufeh, and J. Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., 2010. doi:10.1002/9780470400531.eorms0118.
 - [27] I. Dunning, J. Huchette, and M. Lubin. JuMP: A Modeling Language for Mathematical Optimization. *SIAM Review*, 59:295–320, 2017. doi:10.1137/15m1020575.

BIBLIOGRAPHY

- [28] M. Duran and I. Grossmann. An Outer-Approximation Algorithm for a Class of Mixed-integer Nonlinear Programs. *Mathematical Programming*, 36:307–339, 1986. doi:10.1007/BF02592064.
- [29] F. Engineer, G. Nemhauser, and M. Savelsbergh. Shortest Path Based Column Generation on Large Networks with Many Resource Constraints. Technical report, Georgia Tech, 2008.
- [30] J. E. Falk and R. M. Soland. An Algorithm for Separable Nonconvex Programming Problems. *Management Science*, 15:550–569, 1969.
- [31] S. Feltenmark and K. C. Kiwiel. Dual Applications of Proximal Bundle Methods Including Lagrangian Relaxation of Nonconvex Problems. *SIAM Journal of Optimization*, 10:697–721, 2000. doi:10.1137/S1052623498332336.
- [32] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104:91–104, 2005. doi:10.1007/s10107-004-0570-3.
- [33] R. Fletcher and S. Leyffer. Solving Mixed Integer Nonlinear Programs by Outer Approximation. *Mathematical Programming*, 66:327–349, 1994. doi:10.1007/BF01581153.
- [34] C. A. Floudas. *Deterministic Global Optimization: Theory, Methods, Applications*. Springer US, 2000. doi:10.1007/978-1-4757-4949-6.
- [35] L. R. Ford and D. R. Fulkerson. A Suggested Computation for Maximal Multi-Commodity Network Flows. *Management Science*, 5:97–101, 1958. doi:10.1287/mnsc.5.1.97.
- [36] J. Forrest. *Cbc repository*. COIN-OR, 2020. doi:10.5281/zenodo.3700700.
- [37] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. DUXBURY, 2002.
- [38] B. Franck, K. Neumann, and C. Schwindt. Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR-Spektrum*, 23:297–324, 2001. doi:10.1007/pl00013356.
- [39] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956. doi:10.1002/nav.3800030109.

- [40] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. L. Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig. The SCIP Optimization Suite 7.0. Technical report, Optimization Online, March 2020. URL: http://www.optimization-online.org/DB_HTML/2020/03/7705.html.
- [41] G. Gamrath and M. E. Lübbecke. Experiments with a Generic Dantzig-Wolfe Decomposition for Integer Programs. In P. Festa, editor, *Experimental Algorithms*, pages 239–252. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-13193-6_21.
- [42] A. M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972. doi:10.1007/BF00934810.
- [43] A. M. Geoffrion. Lagrangean relaxation for integer programming. In M. L. Balinski, editor, *Approaches to Integer Programming*, pages 82–114. Springer Berlin Heidelberg, 1974. doi:10.1007/bfb0120690.
- [44] A. Gleixner, L. Eifler, T. Gally, G. Gamrath, P. Gemander, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, F. Serrano, Y. Shinano, J. M. Viernickel, S. Vigerske, D. Weninger, J. T. Witt, and J. Witzig. The SCIP Optimization Suite 5.0. Technical report, www.optimization-online.org/DB_HTML/2017/12/6385.html, 2017.
- [45] S. Goderbauer, B. Bahl, P. Voll, M. Lübbecke, A. Bardow, and A. Koster. An adaptive discretization MINLP algorithm for optimal synthesis of decentralized energy supply systems. *Computers & Chemical Engineering*, 95:38–48, 2016. doi:10.1016/j.compchemeng.2016.09.008.
- [46] I. E. Grossmann, J. Viswanathan, A. Vecchiotti, R. Raman, and E. Kalvelagen. GAMS/DICOPT: A Discrete Continuous Optimization Package. Technical report, GAMS Corporation Inc 37, 2002.
- [47] LLC Gurobi Optimization. Gurobi Optimizer Reference Manual, 2020. URL: <http://www.gurobi.com>.
- [48] W. E. Hart, C. D. Laird, J.-P. Watson, D. L. Woodruff, G. A. Hackebeil, B. L. Nicholson, and J. D. Siirola. *Pyomo—optimization modeling in Python*,

BIBLIOGRAPHY

- volume 67. Springer Science & Business Media, second edition, 2017. doi:10.1007/978-3-319-58821-6.
- [49] E. M. T. Hendrix and B. G.-Tóth. *Introduction to Nonlinear and Global Optimization*. Springer New York, 2010. doi:10.1007/978-0-387-88670-1.
- [50] J. F. R. Herrera, J. M. G. Salmerón, E. M. T. Hendrix, R. Asenjo, and L. G. Casado. On parallel Branch and Bound frameworks for Global Optimization. *Journal of Global Optimization*, 69:547–560, 2017. doi:10.1007/s10898-017-0508-y.
- [51] R. Horst and P. M. Pardalos, editors. *Handbook of Global Optimization*. Springer US, 1995. doi:10.1007/978-1-4615-2025-2.
- [52] R. Horst and H. Tuy. *Global Optimization (Deterministic Approaches)*. Springer, Berlin, 1990. doi:10.1007/978-3-662-02598-7.
- [53] M. Hunting. The AIMMS outer approximation algorithm for MINLP. Technical report, AIMMS B.V, 2011.
- [54] T. Ibaraki. Theoretical comparisons of search strategies in branch-and-bound algorithms. *International Journal of Computer & Information Sciences*, 5:315–344, 1976. doi:10.1007/bf00998631.
- [55] M. Jaggi. Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization. In S. Dasgupta and D. McAllester, editors, *Proceedings of Machine Learning Research*, volume 28, Atlanta, Georgia, USA, 2013. PMLR. URL: <http://proceedings.mlr.press/v28/jaggi13.html>.
- [56] G. R. Kocis and I. E. Grossmann. Computational experience with DICOPT solving MINLP problems in process systems engineering. *Computers & Chemical Engineering*, 13:307–315, 1989. doi:10.1016/0098-1354(89)85008-2.
- [57] O. Kröger, C. Coffrin, H. Hijazi, and H. Nagarajan. Juniper: An Open-Source Nonlinear Branch-and-Bound Solver in Julia. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 377–386. Springer International Publishing, 2018. doi:10.1007/978-3-319-93031-2_27.
- [58] J. Kronqvist, D. E. Bernal, and I. E. Grossmann. Using regularization and second order information in outer approximation for convex MINLP. *Mathematical Programming*, 180:285–310, 2018. doi:10.1007/s10107-018-1356-3.

- [59] J. Kronqvist, D. E. Bernal, A. Lundell, and I. E. Grossmann. A review and comparison of solvers for convex MINLP. *Optimization and Engineering*, 20:397–455, 2018. doi:10.1007/s11081-018-9411-8.
- [60] J. Kronqvist, A. Lundell, and T. Westerlund. The extended supporting hyperplane algorithm for convex mixed-integer nonlinear programming. *Journal of Global Optimization*, 64:249–272, 2016. doi:10.1007/s10898-015-0322-3.
- [61] J. Kronqvist, A. Lundell, and T. Westerlund. Reformulations for utilizing separability when solving convex MINLP problems. *Journal of Global Optimization*, 71(3):571–592, 2018. doi:10.1007/s10898-018-0616-3.
- [62] A. H. Land and A. G. Doig. An Automatic Method of Solving Discrete Programming Problems. *Econometrica*, 28:497–520, 1960. doi:10.2307/1910129.
- [63] C. Lemaréchal and A. Renaud. A geometric study of duality gaps, with applications. *Mathematical Programming*, 90:399–427, 2001. doi:10.1007/PL00011429.
- [64] S. Leyffer, A. Sartenaer, and E. Wanufelle. 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.
- [65] L. Liberti, S. Cafieri, and F. Tarissan. Reformulations in Mathematical Programming: A Computational Approach. In A. Abraham, A. Hassanien, P. Siarry, and A. Engelbrecht, editors, *Foundations of Computational Intelligence Volume 3*, pages 153–234. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-01085-9_7.
- [66] Y. Lin and L. Schrage. The global solver in the LINDO API. *Optimization Methods & Software*, 24:657–668, 2009.
- [67] J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An Algorithm for the Traveling Salesman Problem. *Operations Research*, 11(6):972–989, 1963.
- [68] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53:1007–1023, 2005. doi:10.1287/opre.1050.0234.
- [69] M. Lubin, E. Yamangil, R. Bent, and J. P. Vielma. Polyhedral approximation in mixed-integer convex optimization. *Mathematical Programming*, 172:139–168, 2018. doi:10.1007/s10107-017-1191-y.

BIBLIOGRAPHY

- [70] A. Lundell, J. Kronqvist, and T. Westerlund. The Supporting Hyperplane Optimization Toolkit. *Optimization Online*, 2020. URL: www.optimization-online.org/DB_HTML/2018/06/6680.html.
- [71] A. Lundell, A. Skjäl, and T. Westerlund. A reformulation framework for global optimization. *Journal of Global Optimization*, 57:115–141, 2012. doi:10.1007/s10898-012-9877-4.
- [72] A. Mahajan, S. Leyffer, J. Linderoth, J. Luedtke, and T. Munson. Minotaur: a mixed-integer nonlinear optimization toolkit. *Mathematical Programming Computation*, 2020. doi:10.1007/s12532-020-00196-1.
- [73] C. D. Maranas and C. A. Floudas. Finding all solutions of nonlinearly constrained systems of equations. *Journal of Global Optimization*, 7:143–182, 1995. doi:10.1007/bf01097059.
- [74] G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976. doi:10.1007/BF01580665.
- [75] W. Melo, M. Fampa, and F. Raupp. An overview of MINLP algorithms and their implementation in Muriqui Optimizer. *Annals of Operations Research*, 286:217–241, 2018. doi:10.1007/s10479-018-2872-5.
- [76] C. A. Meyer and C. A. Floudas. Convex Underestimation of Twice Continuously Differentiable Functions by Piecewise Quadratic Perturbation: Spline α BB Underestimators. *Journal of Global Optimization*, 32:221–258, 2005. doi:10.1007/s10898-004-2704-9.
- [77] K. Miettinen. *Nonlinear Multiobjective Optimization*. Springer US, 1998. doi:10.1007/978-1-4615-5563-6.
- [78] MINLPLib. Mixed-integer nonlinear programming library, 2021. Accessed 28 January 2021. URL: <http://www.minlplib.org/>.
- [79] R. Misener and C. Floudas. ANTIGONE: Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations. *Journal of Global Optimization*, 59:503–526, 2014. doi:10.1007/s10898-014-0166-2.
- [80] L. G. Mitten. Branch-and-Bound Methods: General Formulation and Properties. *Operations Research*, 18:24–34, 1970. doi:10.1287/opre.18.1.24.

- [81] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016. doi:10.1016/j.disopt.2016.01.005.
- [82] A. El Mouatasim and Y. Farhaoui. Nesterov Step Reduced Gradient Algorithm for Convex Programming Problems. In Y. Farhaoui, editor, *Big Data and Networks Technologies*. Springer, Cham, 2019. doi:10.1007/978-3-030-23672-4_11.
- [83] P. Muts, S. Bruche, I. Nowak, O. Wu, E. M. T. Hendrix, and G. Tsatsaronis. A Column Generation Algorithm for Solving Energy System Planning Problems. *Submitted to Optimization and Engineering*, 2020.
- [84] P. Muts and I. Nowak. Towards Multi-tree Methods for Large-Scale Global Optimization. In H. L. Thi, H. Le, and T. P. Dinh, editors, *Optimization of Complex Systems: Theory, Models, Algorithms and Applications. WCGO 2019. Advances in Intelligent Systems and Computing*. Springer, Cham, 2019. doi:10.1007/978-3-030-21803-4_50.
- [85] P. Muts, I. Nowak, and E. M. T. Hendrix. A Resource Constraint Approach for One Global Constraint MINLP. In O. Gervasi and et al., editors, *Computational Science and Its Applications – ICCSA 2020*. Springer, Cham, 2020. doi:10.1007/978-3-030-58808-3_43.
- [86] P. Muts, I. Nowak, and E. M. T. Hendrix. On decomposition and multiobjective-based column and disjunctive cut generation for MINLP. *Optimization and Engineering*, 2020. doi:10.1007/s11081-020-09576-x.
- [87] P. Muts, I. Nowak, and E. M. T. Hendrix. The decomposition-based outer approximation algorithm for convex mixed-integer nonlinear programming. *Journal of Global Optimization*, 77:75–96, 2020. doi:10.1007/s10898-020-00888-x.
- [88] H. Nagarajan, M. Lu, S. Wang, R. Bent, and K. Sundar. An adaptive, multivariate partitioning algorithm for global optimization of nonconvex programs. *Journal of Global Optimization*, 74:639–675, 2019. doi:10.1007/s10898-018-00734-1.
- [89] H. Nagarajan, M. Lu, E. Yamangil, and R. Bent. Tightening McCormick Relaxations for Nonlinear Programs via Dynamic Multivariate Partitioning. In M. Rueher, editor, *Principles and Practice of Constraint Programming. CP 2016. Lecture Notes in Computer Science, vol 9892*, pages 369–387. Springer, Cham, 2016. doi:10.1007/978-3-319-44953-1_24.

BIBLIOGRAPHY

- [90] Y. E. Nesterov. A method for solving the convex programming problem with convergence rate $O(1/k^2)$. *Dokl. akad. nauk SSSR*, 269:543–547, 1983.
- [91] I. Nowak. *Relaxation and Decomposition Methods for Mixed Integer Nonlinear Programming*. Birkhäuser, 2005. doi:10.1007/3-7643-7374-1.
- [92] I. Nowak. Parallel Decomposition Methods for Nonconvex Optimization - Recent Advances and New Directions. In L. G. Casado, I. García, and E. M. T. Hendrix, editors, *Proceedings of the XII global optimization workshop MAGO 2014*, pages 73–76, 2014.
- [93] I. Nowak, N. Breitfeld, E. M. T. Hendrix, and G. Njacheun-Njanzoua. Decomposition-based Inner- and Outer-Refinement Algorithms for Global Optimization. *Journal of Global Optimization*, 72:305–321, 2018. doi:10.1007/s10898-018-0633-2.
- [94] I. Nowak and P. Muts. Decomposition-based successive approximation methods for global optimization. In M. T. M. Emmerich, A. H. Deutz, S. Hille, and Y. Sergeyev, editors, *Proceedings LeGO – 14th International Global Optimization Workshop*. AIP Publishing, 2019. doi:10.1063/1.5089985.
- [95] I. Nowak, P. Muts, and E. M. T. Hendrix. Multi-Tree Decomposition Methods for Large-Scale Mixed Integer Nonlinear Optimization. In J. Velásquez-Bermúdez, M. Khakifirooz, and M. Fathi, editors, *Springer Optimization and Its Applications*, pages 27–58. Springer International Publishing, 2019. doi:10.1007/978-3-030-22788-3_2.
- [96] T. Ralphs and M. Galati. Decomposition and Dynamic Cut Generation in Integer Linear Programming. *Mathematical Programming*, 106:261–285, 2006. doi:10.1007/s10107-005-0606-3.
- [97] N. V. Sahinidis. BARON 20.4.14: Global Optimization of Mixed-Integer Nonlinear Programs, *User’s Manual*, 2020. URL: <http://www.minlp.com/>.
- [98] N. Shor. *Minimization methods for non-differentiable functions*. Springer-Verlag, Berlin New York, 1985. doi:10.1007/978-3-642-82118-9.
- [99] L. Su, L. Tang, D. E. Bernal, and I. E. Grossmann. Improved quadratic cuts for convex mixed-integer nonlinear programs. *Computers & Chemical Engineering*, 109:77–95, 2018. doi:10.1016/j.compchemeng.2017.10.011.

- [100] M. Tawarmalani and N. Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103:225–249, 2005. doi:10.1007/s10107-005-0581-8.
- [101] M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*. Kluwer Academic Publishers, 2002. doi:10.1007/978-1-4757-3532-1.
- [102] F. Trespalacios and I. E. Grossmann. Review of Mixed-Integer Nonlinear and Generalized Disjunctive Programming Methods. *Chemie Ingenieur Technik*, 86:991–1012, 2014. doi:10.1002/cite.201400037.
- [103] H. Tuy. D.C. Optimization: Theory, Methods and Algorithms. In R. Horst and P. M. Pardalos, editors, *Handbook of Global Optimization. Nonconvex Optimization and Its Applications*, pages 149–216. Springer US, 1995. doi:10.1007/978-1-4615-2025-2_4.
- [104] S. Vigerske. *Decomposition in Multistage Stochastic Programming and a Constraint Integer Programming Approach to Mixed-Integer Nonlinear Programming*. PhD thesis, Humboldt-Universität zu Berlin, 2012.
- [105] A. Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. PhD thesis, Carnegie Mellon University, Pittsburgh, USA, <http://researcher.watson.ibm.com/researcher/files/us-andreasw/thesis.pdf>, 2002.
- [106] A. Wächter and B. T. Lorenz. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006. doi:10.1007/s10107-004-0559-y.
- [107] T. Westerlund and K. Lundqvist. Alpha-ECP, An Interactive MINLP-Solver Based on the Extended Cutting Plane Method. Technical report, Abo Akademi University, 2005. URL: <http://users.abo.fi/twester1/A-ECPManual.pdf>.
- [108] T. Westerlund and F. Pettersson. An Extended Cutting Plane Method for Solving Convex MINLP Problems. *Computers and Chemical Engineering*, 21:131–136, 1995. doi:10.1016/0098-1354(95)87027-X.