University of London
Imperial College of Science, Technology and Medicine
Department of Computing

Branching Strategies for Mixed-Integer Programs Containing Logical Constraints and Decomposable Structure

Miten Narendra Mistry

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of the University of London and
the Diploma of Imperial College,

Declaration of Originality

I declare that all work in this thesis that has not been carried out by myself has been properly acknowledged.

Miten Narendra Mistry

Copyright

Abstract

Decision-making optimisation problems can include discrete selections, e.g. selecting a route, arranging non-overlapping items or designing a network of items. Branch-and-bound (B&B), a widely applied divide-and-conquer framework, often solves such problems by considering a continuous approximation, e.g. replacing discrete variable domains by a continuous superset. Such approximations weaken the logical relations, e.g. for discrete variables corresponding to Boolean variables. Branching in B&B reintroduces logical relations by dividing the search space. This thesis studies designing B&B branching strategies, i.e. how to divide the search space, for optimisation problems that contain both a logical and a continuous structure.

We begin our study with a large-scale, industrially-relevant optimisation problem where the objective consists of machine-learnt gradient-boosted trees (GBTs) and convex penalty functions. GBT functions contain if-then queries which introduces a logical structure to this problem. We propose decomposition-based rigorous bounding strategies and an iterative heuristic that can be embedded into a B&B algorithm. We approach branching with two strategies: a pseudocost initialisation and strong branching that target the structure of GBT and convex penalty aspects of the optimisation objective, respectively. Computational tests show that our B&B approach outperforms state-of-the-art solvers in deriving rigorous bounds on optimality.

Our second project investigates how satisfiability modulo theories (SMT) derived unsatisfiable cores may be utilised in a B&B context. Unsatisfiable cores are subsets of constraints that explain an infeasible result. We study two-dimensional bin packing (2BP) and develop a B&B algorithm that branches on SMT unsatisfiable cores. We use the unsatisfiable cores to derive cuts that break 2BP symmetries. Computational results show that our B&B algorithm solves 20% more instances when compared with commercial solvers on the tested instances.

Finally, we study convex generalized disjunctive programming (GDP), a framework that supports logical variables and operators. Convex GDP includes disjunctions of mathematical constraints, which motivate branching by partitioning the disjunctions. We investigate separation by branching, i.e. eliminating solutions that prevent rigorous bound improvement, and propose a greedy algorithm for building the branches. We propose three scoring methods for selecting the next branching disjunction. We also analyse how to leverage infeasibility to expedite the B&B search. Computational results show that our scoring methods can reduce the number of explored B&B nodes by an order of magnitude when compared with scoring methods proposed in literature. Our infeasibility analysis further reduces the number of explored nodes.

Acknowledgements

My PhD has been challenging, but I could not have completed it without the support of the people around me. Firstly, I would like to thank my PhD advisor, Dr Ruth Misener, for her time, support, encouragement and guidance during my PhD. In particular, Dr Misener's pragmatic approach to conducting research ensured that I remained grounded, her constant willingness to make time in her schedule ensured that I was always supported, and her sense of humour ensured that I was always smiling. She has been an excellent mentor over the past five years and continues to be one of my role models. I am truly fortunate to have had the opportunity to work with her. I would also like to thank my second supervisor, Professor Michael Huth for his guidance and support, especially in my earlier years when I was learning new concepts that proved essential thoughout my PhD. I thank my BASF collaborators Dr Robert Matthew Lee and Dr Gerhard Krennrich for their guidance in machine-learning concepts.

I have also been fortunate enough to be surrounded by my postdoctoral mentors and PhD colleagues who have made my PhD years all the more enjoyable while supporting my development as a researcher. I thank my postdoctoral mentors: Dr Dimitrios Letsios for his support and guidance in academic writing and insight into approximation algorithms, Dr Jan Kronqvist for his willingness to discuss my ideas at length and for showing me what it means to be inquisitive, Dr Juan Campos Salazar for ensuring that my intuition was backed by rigour, and Dr Andrea Callia D'Iddio for his insight in logic-based methods. I also thank my PhD colleagues Dr Georgia Kouyialis, Dr Radu Baltean-Lugojan, Dr Simon Olofsson, Dr Chin Pang Ho, Dr Vahan Hovhannisyan, Dr Sei Howe, Francesco Ceccon, Johannes Wiebe, Alexander Thebelt and Haoyang Wang for their advice, the many coffee breaks, lunches, dinners, pub trips, and hours of laughter. My gratitude is also expressed to my hipeds cohort: Dr Jo Schlemper, Domagoj Margan, Ilia Kisil, Giannis Evagorou, Davide Cavezza, Daniel Coelho de Castro, Sanket Kamthe, Tianjiao Sun, Matthew Douthwaite, Andrea Paudice, Anastasios Andronidis and Andrei Lascu for their support, especially in my first year.

I thank my examiners Professor Ignacio Grossmann and Dr Panos Parpas for taking the time to read and assess my thesis. I would also like to thank Professor Francesca Toni for supporting me as I transistioned to moving on from my PhD. Additional expressions of gratitude are extended to Dr Amani El-Kholy, Dr Krysia Broda, Bridget Gundry and Joanne Day for their

assistance with administrative tasks. A special thanks is also extended to Dr Steffen van Bakel, my undergraduate tutor, who encouraged me to consider a PhD as an undergraduate student.

I am incredibly lucky to have my friends outside of my PhD colleagues whose invaluable support, guidance and companionship have helped to complete my PhD. I thank Sam Robinson and James Eade, two of my oldest friends, for always being there. Furthermore, I am grateful for the support of my friends from Leicester, namely: Alex Taylor, Amy Taylor, David Monger, Hayden Nickson, Thomas Worley, Alex Romankiw, Jack Robinson and Oliver Monger. Also, I thank my undergraduate friends: Leo Mak, George Nishimura, Peter Sullivan-Watkins, Niket Shah and Thomas Przepiorka; for the years of fun. I also appreciate the time and conversations had with my flatmates: Charlotte Abberger, Juliette Hannay, Max Kretschmer, Marine Billieres, Ariane Plantive, Melissa Pesce, Alix Berlizot, Louis Briatta, Anna Salles and Rose Connan.

Last but by no means least, my final expressions of gratitude, I leave for my family whose unconditional love and support thoughout my life have ensured that I never feel alone. Specifically, I thank my grandparents: Ba and Bapa; my uncles and aunts: Hasulafoi, Natvarlalfuaji, Rameshkaka, Ranjanakaki, Bhartifoi, Dipaknana, Kalpananani, Babumama, Gitamami, Kantamasi, Chottumasaji, Jasumasi and Ishvarmasaji; and my cousins: Vinay, Geeta, Fabio, Kiran, Riken, Manoj, Babita, Teena, Manoj, Punita, Rikesh, Ashish and Sajel. A special thanks is reserved for my sister and brother-in-law, Keerti and Vimal, who always make me smile. Finally, I thank my parents, Narendra and Prafulla, for everything.

Contents

List of Tables

List of Figures

List of Publications

- Mistry, M. and Misener, R. (2016). Optimising heat exchanger network synthesis using convexity properties of the logarithmic mean temperature difference difference. *Computers & Chemical Engineering*, 94:1–17.

- Mistry, M., Callia D'Iddio, A., Huth, M., and Misener, R. (2018a). Satisfiability modulo theories for process systems engineering. *Computers & Chemical Engineering*, 113:98–114.

- Mistry, M., Letsios, D., Krennrich, G., Lee, R. M., and Misener, R. (2018b). Mixed-integer convex nonlinear optimization with gradient-boosted trees embedded. *arXiv e-prints*. arXiv:1803.00952.

- Letsios, D., Baltean-Lugojan, R., Ceccon, F., Mistry, M., Wiebe, J., and Misener, R. (2020). Approximation algorithms for process systems engineering. *Computers & Chemical Engineering*, 132:106599.

Abbreviations and Notation

Abbreviations.

Table 1 lists the abbreviations used in this thesis.

Table 1: Abbreviations

Abbreviation	Description
2BP	Two-dimensional bin packing
B&B	Branch-and-bound
CNF	Conjunctive normal form
CLay	Constrained layout
Convex MINLP	Mixed-integer convex programming
CP	Constraint programming
D2BP	Two-dimensional bin packing decision problem
DNF	Disjunctive normal form
DPLL	Davis-Putnam-Logemann-Loveland
GBT	Gradient-boosted tree
GDP	Generalized disjunctive programming
IIS	Irreducible infeasible subsystem
LBBD	Logic-based Benders decomposition
LP	Linear programming
MILP	Mixed-integer linear programming
MINLP	Mixed-integer nonlinear programming
MISOCP	Mixed-integer second-order cone programming
ML	Machine-learning
MLLP	Mixed logical-linear programming
NLP	Nonlinear programming
OPP	Orthogonal packing problem
PCA	Principal component analysis
PSO	Particle swarm optimisation
SA	Simulated annealing
SAT	Propositional satisfiability

Abbreviation	Description
SMT	Satisfiability modulo theories
SOCP	Second-order cone programming

Notation

Table 2: General notation used for mathematical and propositional symbols, and in formulation frameworks.

Symbol	Description
Sets	
$\mathbb{B}$	Boolean domain: $\{\text{True}, \text{False}\}$.
$\mathbb{R}$	Set of real numbers.
$\mathbb{Z}$	Set of integers.
$S_{\bowtie k}$	$\{x \in S \mid x \bowtie k\}$ where $S \subseteq \mathbb{R}$, $\bowtie \in \{\geq, >, <, \leq\}$, and $k \in \mathbb{R}$.
$\emptyset$	Empty set.
$[n]$, $n \in \mathbb{Z}$	$\{1, \ldots, n\}$ if $n > 0$, otherwise $\emptyset$.
K	Index set of disjunctions in generalized disjunctive programming.
D_k	Index set of disjuncts for disjunction $k \in K$ in generalized disjunctive programming.
Parameters	
$n_r, n_b \in \mathbb{Z}_{\geq 0}$	Number of real and Boolean variables, respectively.
$\varepsilon > 0$	Small constant, e.g. 10^{-6}.
$f(\cdot)$	Optimisation objective function.
$g(\cdot)$	Optimisation constraint function, often stated as $g(\cdot) \leq 0$.
$r(\cdot)$	Constraint present in generalized disjunctive programming disjunctive constraint.
$\boldsymbol{c}$	Linear objective cost coefficients.
$\boldsymbol{A}$, $\boldsymbol{B}$	Linear constraint coefficients.
$\boldsymbol{b}$	Linear constraint right-hand side.
$\boldsymbol{x}^L$, $\boldsymbol{x}^U$	Lower and upper bounds on variables $\boldsymbol{x}$, respectively.
$\Omega : \mathbb{B}^{n_b} \to \mathbb{B}$	Propositional constraint.
M_{ikj}	Big-M parameter for constraint r_{ikj} in generalized disjunctive programming.
Variables	
$x \in \mathbb{R}$, $\boldsymbol{x} \in \mathbb{R}^{n_r}$	Continuous variables: scalar and vector form, respectively.
$y \in \{0,1\}$, $\boldsymbol{y} \in \{0,1\}^{n_b}$	Binary variables: scalar and vector form, respectively.
$z \in \mathbb{Z}$, $\boldsymbol{z} \in \mathbb{Z}^{n_I}$	Integer variables: scalar and vector form, respectively.
$Y \in \mathbb{B}$, $\boldsymbol{Y} \in \mathbb{B}^{n_b}$	Boolean variables: singular and vector form, respectively.

Symbol	Description
$\boldsymbol{\nu_{ik}} \in \mathbb{R}^{n_r}$	Disaggregation of $\boldsymbol{x}$ corresponding to generalized disjunctive programming disjunction $k \in K$ and disjunct $i \in D_k$.
Functions and Operators	
$\neg : \mathbb{B} \to \mathbb{B}$	Logical NOT.
$\wedge, \vee : \mathbb{B}^n \to \mathbb{B}$	Logical AND and OR, respectively. May be stated with an infix notation when $n = 2$.
$\to, \leftrightarrow: \mathbb{B}^2 \to \mathbb{B}$	Logical IF-THEN and IFF, respectively. Usually stated with an infix notation.
$\veebar : \mathbb{B}^n \to \mathbb{B}$	True if and only if exactly one of the arguments is True.
$\boldsymbol{c}^\top$	Transpose of vector $\boldsymbol{c}$.
$\|\cdot\|, \|\cdot\|_p : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$	Arbitrary norm and p-norm, respectively.
proj	Projection operator
Branch-and-bound	
Q	List/set of unexplored nodes
S	Node of the B&B tree
UB	Global upper bound
$\text{lb}(S)$	Local lower bound on B&B node S

Table 3: Symbols used for Chapters 3 and 4 model and B&B algorithm.

Symbol	Description
GBT Ensemble Definition	
n_C	Number of the GBT-trained function (continuous) variables
i	Continuous variable index
x_i	Continuous variable
$\boldsymbol{x}$	Vector $(x_1, \ldots, x_{n_C})^T$
$\mathcal{T}$	Set of gradient boosted trees
t	Gradient boosted tree
$\mathcal{V}_t$	Set of split nodes (vertices) in tree t
$\mathcal{L}_t$	Set of leaf nodes in tree t
s	Split node associated with a tree t and mainly referred to as (t, s)
$i(t, s)$	Continuous variable index associated with split node s in tree t
$v(t, s)$	Splitting value of variable $x_{i(t,s)}$ at split node s in tree t
$\text{GBT}_t(\boldsymbol{x})$	Tree t evaluation at point $\boldsymbol{x}$
$\text{GBT}(\boldsymbol{x})$	GBT ensemble evaluation at point $\boldsymbol{x}$
Convex MINLP with GBTs Problem Definition	
$\text{cvx}(\boldsymbol{x})$	Convex function evaluation at point $\boldsymbol{x}$
m_i	Number of variable x_i splitting values
$v_{i,j}$	j-th greatest variable x_i splitting value
v_i^L or $v_{i,0}$	Variable x_i lower bound
v_i^U or v_{i,m_i+1}	Variable x_i upper bound

Symbol	Description
$\boldsymbol{v}^L$	Vector $(v_1^L, \ldots, v_n^L)$
$\boldsymbol{v}^U$	Vector $(v_1^U, \ldots, v_n^U)$
$\text{Left}_{t,s}$	Set of leaves in the subtree rooted in the left child of s in tree t
$\text{Right}_{t,s}$	Set of leaves in the subtree rooted in the right child of s in tree t
$F_{t,l}$	Contribution of leaf node l in tree t
$y_{i,j}$	Binary variable indicating whether $x_i \leq v_{i,j}$, or not
$z_{t,l}$	Binary variable specifying whether tree t evaluates at leaf l
d	Maximum tree depth
Branch-and-Bound Algorithm Overview	
$[\boldsymbol{v}^L, \boldsymbol{v}^U]$	Optimisation problem global domain
$S = [\boldsymbol{L}, \boldsymbol{U}]$	Optimisation problem subdomain / B&B node
(x_i, v)	GBT splitting point / B&B branch
$S_{\text{left}}, S_{\text{right}}, S_c, S'$	B&B nodes
Q	Set of unexplored B&B nodes
P_{root}	Initial GBT ensemble partition at B&B root node
P, P', P''	GBT ensemble partitions
$b^{\text{cvx},S}$	Convex lower bound over domain S
$b^{\text{GBT},S,P}$	GBT lower bound over domain S with respect to partition P
Lower Bounding	
W^S	Optimal objective value, i.e., tightest relaxation
$\hat{W}^S$	Relaxation dropping linking constraints
$b^{\text{GBT},S,*}$	Optimal GBT lower bound over domain S
$\boldsymbol{x}^*$	Optimal solution
i, j, l	Subset indices of a GBT ensemble partition
k	GBT ensemble partition size
$\mathcal{T}_i, \mathcal{T}_j, \mathcal{T}', \mathcal{T}''$	Subsets of GBTs
N	GBT ensemble subset size
$n^{\mathcal{T},S}$	Number of leaves in GBT subset $\mathcal{T}$ over domain S
f^*	Best found feasible objective
q	Time limit on lower bound improvement algorithm
Branching	
B	Branch ordering
$r((x_i, v), t)$	Set of nodes in tree t that split on (x_i, v)
$d(s)$	Depth of split node s (root node has zero depth)
$w(s)$	Weight of split node s
$i(s)$	Number of inactive leaves below split s when branching with respect to (x_i, s)
$\text{weight}((x_i, v), t)$	Weight assigned to (x_i, v) in tree t
$\text{weight}((x_i, v), \mathcal{T})$	Weight assigned to (x_i, v) in GBT ensemble $\mathcal{T}$
$\text{inactive}((x_i, v), \mathcal{T})$	Number of inactive leaves when branching on pair (x_i, v) in $\mathcal{T}$
$\text{cover}(t, s)$	Set of leaves covered by split node s at tree t
$S, S_{\text{left}}, S_{\text{right}}, S_0$	B&B nodes denoted by their corresponding domain
l	Strong branching lookahead parameter

Table 4: Symbols used for the Chapter 6 convex generalized disjunctive programming analysis.

Symbol	Description
k, k'	Disjunction indexing variable
i, i'	Disjunct indexing variable
j	General indexing variable
p_{ik}	Number of mathematical constraints in disjunct (i, k)
K	Index set of disjunctions
$K^F(\hat{\boldsymbol{y}})$	Index set of disjunctions that a fractional under $\hat{\boldsymbol{y}}$
D_k	Index set of disjuncts in disjunction $k \in K$
$D'_k, D^i_k \subseteq D_k$	Index set of smaller disjunction derived from disjunction $k \in K$
$\boldsymbol{x}$, $\hat{\boldsymbol{x}}$, $\boldsymbol{x'}$	Continuous variables
$\boldsymbol{Y}$, $\hat{\boldsymbol{Y}}$, $\boldsymbol{Y'}$	Boolean variables
$\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, $\boldsymbol{y'}$, $\bar{\boldsymbol{y}}$	Binary variables (in hull relaxation)
$\boldsymbol{x}^*$, $\boldsymbol{y}^*$	Best known feasible solution
$\boldsymbol{\nu}$, $\hat{\boldsymbol{\nu}}$, $\boldsymbol{\nu'}$	Disaggregated variables associated with $\boldsymbol{x}$ in hull relaxation
S, $\hat{S}$, S', S_i	Convex GDP subdomain/B&B node
S^{root}	Initial subdomain/B&B root node
$R^H(S)$	Hull relaxation feasible region of convex GDP corresponding to B&B node S
Q	B&B unexplored nodes
$\text{child}(S, k', D'_k)$	B&B child node formed at S by restricting disjunction k' to disjuncts $i \in D'_k$ (Definition 6.2)
P	Partition of $D_{k'}$ for some disjunction $k \in K$ used for branching
$\text{proj}_{\boldsymbol{x}}$	Projection operator, e.g. $\text{proj}_{\boldsymbol{x}} S = \{\boldsymbol{x} \mid (\boldsymbol{x}, \boldsymbol{Y}) \in S\}$
$\mathcal{D}$	Subcombination for some B&B node GDP instance S (Definition 6.14)
$S_{\mathcal{D}}$	B&B node S restricted to subcombination $\mathcal{D}$ (Definition 6.14)
S^*	B&B node S instance without its global constraints
$\mathcal{I}$	Irreducible infeasible set of some hull relaxation $R^H(S)$

Chapter 1

Introduction

1.1 Motivation

Decision-making optimisation problems encountered in science and engineering can involve both logical and continuous elements. In chemical engineering, for example, heat exchanger network synthesis, an application that re-uses excess heat by designing a network of heat exchangers, involves the logical decisions of which heat exchangers exist and continuous constraints derived from the physics of heat exchange (Yee and Grossmann, 1990; Furman and Sahinidis, 2002). Similarly, in computer science, resource constrained scheduling, an application that schedules tasks subject to resource limitations, involves the logical relations relating to precedence between tasks and continuous constraints modelling resource capacities (Brucker et al., 1999; Hartmann and Briskorn, 2010). Globally optimal solutions to such problems are preferable since they can relate to lower operating costs, improved energy efficiency, and faster delivery times.

Modelling frameworks capable of formulating optimisation problems that contain both a continuous and logical structure include mixed-integer nonlinear programming (MINLP) (Williams, 1990) and generalized disjunctive programming (GDP) (Raman and Grossmann, 1994). MINLP formulations involve continuous variables, integral variables and mathematical inequality constraints, and GDP extends MINLP with Boolean variables, propositional constraints and disjunctions, i.e. logical OR, of mathematical inequality constraints (Williams, 1990; Nemhauser and Wolsey,

1988; Belotti et al., 2013; Raman and Grossmann, 1994; Grossmann and Trespalacios, 2013). Solution strategies for solving MINLP and GDP instances include the branch-and-bound (B&B) divide-and-conquer framework (Land and Doig, 1960). B&B proves optimality by constructing a search tree of subproblems. The search tree recursively partitions the feasible domain, i.e. the subproblems associated with a complete set of sibling nodes cover the feasible region of the problem associated with the parent node. In the worst case, B&B will have exponential runtime since the search tree may enumerate all logical/discrete assignments. However, this worst case is generally avoided by discarding search tree nodes that are proven to be devoid of any global optima, such a proof often compares local bounds on optimality to known feasible solutions. State-of-the-art solvers capable of solving subclasses of MINLP and GDP implement variants of B&B, e.g. Sahinidis (1996); Achterberg (2009); Misener and Floudas (2014).

This thesis aims at designing branching strategies, i.e. the 'divide' part of the B&B divide-and-conquer algorithm, for problems involving both logical and continuous structure. Different branching strategies can effect the total solve time, tree depth and number of B&B nodes visited (Achterberg et al., 2005; Vanderbeck, 2011; Ostrowski et al., 2011). The most common approach for branching involves selecting a variable and dividing its domain into two child problems. Alternative strategies include leveraging an underlying ordering (Beale and Tomlin, 1970; Beale and Forrest, 1976), forming subproblems with complementary constraints (Karamanov and Cornuéjols, 2011; Cornuéjols et al., 2011), or constructing multiple child problems (Beaumont, 1990; Lee and Grossmann, 2000; Morrison et al., 2014).

1.2 Objectives and Contributions

This thesis investigates B&B branching, on optimisation problems with a logical and mathematical structure, from three perspectives. Chapter 4 assesses how structural information from a large-scale application with an inherent logical structure may be leveraged for branching. Chapter 5 assesses how a tool that provides explanations of local infeasibilities can guide branching decisions. Chapter 6 assesses how branches may constructed for convex GDP to

reduce redundancy in the search.

Chapter 3 formulates a large-scale, industrially-relevant optimisation problem whose objective contains gradient-boosted trees (GBTs) and penalty functions mitigating risk. Chapter 4, our first line of work, develops a solution strategy for this optimisation problem. GBTs are a supervised statistical learning method for estimating an unknown function (Friedman, 2001; Hastie et al., 2009). Evaluating a trained GBT function involves traversing decision trees (Breiman et al., 1984). This evaluation via tree traversals introduces a logical structure to the Chapter 3 optimisation problem. Chapter 4 proposes solving the Chapter 3 optimisation problem with a B&B approach. With respect to branching, Chapter 4 questions:
How can GBT problem structure be exploited for branching?
Developing the B&B algorithm requires strategies for (i) branching, (ii) deriving bounds on optimality, and (iii) finding heuristic solutions. Chapter 4 addresses the Chapter 3 GBT optimisation problem with a decomposition that separates the GBT elements from the penalty functions. The algorithm addresses branching in two ways: (a) a pseudocost initialisation strategy and (b) applying strong branching. The pseudocost initialisation strategy assigns scores to potential branches and aims to quantify how effective a branch, if selected, is at reducing the size of the resulting GBT structure. Strong branching leverages relative efficiency with which the penalty aspect of the decomposition may be solved to derive suboptimality proofs quickly. We develop a further decomposition for the GBTs (assumed to be large-scale) to derive bounds on optimality. We propose an iterative deterministic heuristic that considers the optimisation problem for a subset of the GBT decision trees and iterative introduces previously excluded decision trees to generate candidate solutions. Numerical results show that our B&B algorithm achieves tighter bounds on optimality than a state-of-the-art commericial solver under a 1 hour time limit. On tested instances where the penalty is more heavily weighted, our optimality bounds are significantly tighter. Our pseduocost initialisation strategy is also shown to be effective as, when comparing with random branch selection, the pseudocost initialisation strategy consistently outperforms random selection. The decomposition-based GBT lower bound that we propose is also shown to be capable of achieving a 4-times speed up in the time-to-bound when comparing with commercial solvers. Chapters 3 and 4 contribute to:

Mistry, M., Letsios, D., Krennrich, G., Lee, R. M., and Misener, R. (2018b). Mixed-integer convex nonlinear optimization with gradient-boosted trees embedded. *arXiv e-prints*. arXiv:1803.00952.

Chapter 5, our second line of work, studies how satisfiability modulo theories (SMT), a constraint feasibility checker guided by a propositional satisfiability solver, may be leveraged in an optimisation setting (Bjørner and De Moura, 2011). SMT modelling offers more freedom than GDP, since mathematical constraints can be used in arbitrary propositional formulas (opposed to just disjunctions). Chapter 5 studies two-dimensional bin packing (Chung et al., 1982), an application that assigns a set of rectangles to a minimal number of identical larger rectangular bins such that each rectangle is contained in its assigned bin and no two rectangles overlap. Two-dimensional bin packing can contain many equivalent symmetric solutions, e.g. a feasible assignment of rectangles to more than one bin may be converted to an equivalent solution by permuting bin indices. Such symmetries can hinder optimality proofs (Margot, 2010; Liberti, 2012). Employing an SMT solver is interesting because it utilises the relative efficiency of modern propositional satisfiability solvers (Malik and Zhang, 2009), which opposes the common MINLP and GDP solving approach where efficient mathematical solvers take a similar role of a propositional satisfiability solver in SMT. Furthermore, SMT solvers are capable of deriving proofs of infeasibility, i.e. a subset of constraints responsible for the infeasibility. With respect to branching, Chapter 5 questions:

How can we leverage SMT derived infeasibility proofs for two-dimensional bin packing subproblems to branch in an instance specific manner?

Chapter 5 proposes three algorithms for leveraging SMT: (i) an iterative algorithm that removes the number of available bins that terminates at a result of infeasibility, (ii) an iterative algorithm, inspired by logic-based Benders decomposition (Hooker and Ottoson, 2003), that increases the number of available bins that terminates at a result of feasibility, and (iii) an extension to the second iterative algorithm where branches are formed on non-trivial proofs of infeasibility. Furthermore, the third algorithm proposes further symmetry breaking enhancements that are introduced by (a) branching on a proof of infeasibility and (b) the presence of identical rectangles. Numerical results show that our SMT-based B&B algorithm is capable of proving optimality of

an additional 20% of the tested instances when compared to commercial solvers. We also show that our branching and SMT-based symmetry breaking enhancements are effective, since the B&B algorithm is able to solve at least 10% more of the tested instances when compared with the two iterative algorithms. Chapter 5 contributes to:

> Mistry, M., Callia D'Iddio, A., Huth, M., and Misener, R. (2018a). Satisfiability modulo theories for process systems engineering. *Computers & Chemical Engineering*, 113:98–114.

Chapter 6, our final line of work, designs a B&B algorithm for convex GDP, a subclass of GDP that assumes all continuous constraints are convex. Convex GDP is an interesting subclass of GDP, since, the (more general) GDP framework is likely to utilise a convex GDP approximation when deriving bounds on optimality in a B&B setting (Lee and Grossmann, 2001), i.e. efficient solutions of convex GDPs may be required to solve GDP. With respect to branching, Chapter 6 questions:

How can we construct branches in a convex GDP B&B algorithm to avoid redundancy in the search?

Chapter 6 designs a B&B algorithm for convex GDP that (i) uses (mathematical programming) hull relaxations (Stubbs and Mehrotra, 1999; Grossmann and Lee, 2003), (ii) branches over the convex GDP disjunct selection constraint, and (iii) propagates local infeasibilities across the B&B search tree. Solving a hull relaxation to optimality derives a lower bound and provides a relaxation solution. If this relaxation solution does not correspond to a feasible convex GDP solution, allowing any descendant node to reach this solution again is a redundancy in the search. Chapter 6 proposes a greedy algorithm that constructs branches by splitting a disjunction and aims to avoid this redundancy, this algorithm adapts a cutting plane separation problem (Stubbs and Mehrotra, 1999; Vecchietti et al., 2003). The B&B algorithm has to select a disjunction for branching. Chapter 6 suggests three scoring methods for selecting which disjunction to branch on. Finally, the B&B algorithm propagates local infeasibilities in the form of conflict clauses (Marques-Silva and Sakallah, 1996) over the GDP Boolean variables by analysing irreducible infeasible subsystems (Chakravarti, 1994), derived by an underlying mathematical solver. We

test our methods on the constrained layout problem (CLay) (Sawaya, 2006) which, given a set of circles that are fixed in the 2-D plane, finds the optimal arrangement of a set of rectangles such that they do not overlap and each rectangle is contained in a circle. CLay, as defined by Sawaya (2006), is such that no disjunction shares variables with the objective. We consider an additional equivalent form of CLay that has disjunctions that do share variables with the objective to assess whether having disjunctions that are more directly connected to the objective, i.e. via shared variables, affects the B&B algorithm performance. Numerical tests show that our disjunction selecting scoring methods can result in a large reduction in the number of explored nodes. This result is more pronounced for the formulation where disjunctions share variables with the objective and our *centre-shifted most fractional* disjunction selection strategy consistently outperforms the other tested disjunction selection strategies. Our infeasibility propagation is also shown to be effective as it can result a significant reduction in the number of explored on the tested instances.

1.3 Thesis Outline

The remainder of this thesis is structured as follows. Chapter 2 covers background material used throughout the thesis. Chapter 3 introduces gradient-boosted trees and formulates an optimisation that contains gradient-boosted tree functions as part of its objective. Chapter 4 develops a B&B solution methodology and proposes heuristics for the Chapter 3 optimisation problem. Chapter 5 studies how satisfiability modulo theories-derived infeasibility proofs may be utilised in an optimisation context. In particular, Chapter 5 develops a B&B strategy that branches on the infeasibility proof that satisfiability modulo theories derives. Chapter 6 presents a B&B algorithm for convex GDP that constructs branches by separating integer infeasible relaxation solutions and propagates local infeasibilities by converting an infeasible subsystem of constraints to GDP no-good cuts. Chapter 7 concludes by summarising achievements and proposes interesting directions for future work.

Chapter 2

Background Theory

This chapter provides background theory for constraint satisfaction and optimisation modelling frameworks and solution algorithms.

2.1 Constraint Satisfaction Frameworks

Constraint satisfaction addresses whether a given set of constraints admits a satisfying solution. If such solution exists, the constraint set is called *feasible* or *satisfiable*. Otherwise, the constraint set is called *infeasible* or *unsatisfiable*. This section discusses propositional satisfiability (SAT), constraint programming (CP) and satisfiability modulo theories (SMT).

2.1.1 Propositional Satisfiability

Traditional propositional satisfiability only incorporates Boolean variables.

Definition 2.1. *Let* $\Omega : \mathbb{B}^n \to \mathbb{B}$ *be a propositional formula. The feasibility problem SAT assesses whether exists an assignment* $\hat{\boldsymbol{Y}} \in \mathbb{B}^n$ *such that* $\Omega(\hat{\boldsymbol{Y}}) = True$.

There exist efficient, satisfiability-preserving transformations from any propositional formula $\Omega(\boldsymbol{Y})$ to conjunctive normal form (CNF) (Tseitin, 1983; Plaisted and Greenbaum, 1986; Eén

and Sörensson, 2006), so SAT solvers typically assume that $\Omega(\boldsymbol{Y})$ is written in CNF. In CNF: literals, i.e. propositional variables Y_i or their negation $\neg Y_i$, form clauses $\Omega_j(\boldsymbol{Y})$, i.e. disjunctions ($\vee$) of literals. The final propositional formula $\Omega(\boldsymbol{Y}) = \bigwedge_{j=1}^{m} \Omega_j(\boldsymbol{Y})$ is a conjunction of clauses.

SAT applications include: planning (Kautz and Selman, 1992), model checking (Biere et al., 1999) and scheduling (Zhang, 2002). Although SAT is $\mathcal{NP}$-complete (Cook, 1971) and the worst-case complexity is exponential, modern SAT solvers can handle problems with hundreds of thousands of variables (Malik and Zhang, 2009).

Most SAT solvers implement the Davis-Putnam-Logemann-Loveland (DPLL) search algorithm (Davis and Putnam, 1960; Davis et al., 1962). DPLL fixes variable Y_i assignments, i.e. truth assignments, using a tree-based branching approach. DPLL propagates truth assignments to all clauses P_j. Propagating truth values may allow DPLL to assign further variables a truth value. If DPLL finds that a partial assignment is *unsatisfiable*, i.e. cannot satisfy Ω, then the algorithm backtracks and assigns a different value to one of the variables. DPLL continues until it either: (i) finds a combination of truth values for Y_i satisfying Ω or (ii) proves the formula Ω is unsatisfiable. DPLL also has functionality supporting warm starts.

SAT solving techniques include (Biere et al., 2009): (i) Boolean constraint propagation, where the current fixed variable set implies variable assignments, (ii) resolution, where sets of clauses derive additional clauses, (iii) and conflict driven clause learning, where an unsatisfiable result derives extra clauses pruning the search tree (Davis and Putnam, 1960; Davis et al., 1962; Silva and Sakallah, 1996). SAT solving methods are highly applicable to optimisation (Hooker and Osorio, 1999; Achterberg, 2007a).

2.1.2 Satisfiability Modulo Theories

Satisfiability modulo theories (SMT) incorporates continuous, integer, and Boolean variables to assess constraint set satisfiability by separating truth value assignment from the correctness reasoning with respect to a *theory*. SMT consists of: (i) a SAT solver and (ii) a theory solver for a theory of our choice (De Moura and Bjørner, 2008). The SMT approach to constraint

satisfaction uses powerful SAT solving to derive a, potentially smaller, set of constraints to assess theory satisfiability. A background theory is a set of axioms and symbols, e.g. the theory of arithmetic. An SMT solver consists of a SAT solver and a theory solver. The idea is to leverage the strength and robustness of modern SAT solvers to search for a feasible solution. The modelling framework exposed by SMT allows for Boolean variables to be used with background theory variables, e.g $Y \rightarrow (x \geq 0)$ where x is continuous and Y is Boolean, so SMT is a natural choice when logical decisions form a part of the modelled system.

SMT research dates back to the 1970s with early work on decision procedures (Nelson and Oppen, 1979, 1980; Shostak, 1979, 1982). Available SMT theories include: equality with uninterpreted functions, linear arithmetic, and arrays (Biere et al., 1999). Most SMT solvers, e.g. `Z3`, can also handle nonlinear arithmetic, i.e. polynomial functions. DPLL(T) generalises DPLL (Ganzinger et al., 2004). SMT is primarily applied in program verification and formal methods, but it also has scheduling and planning applications (Bjørner and De Moura, 2011). SMT provides a (provable) guarantee of feasibility/infeasibility. This is the different from the idea of a *feasibility pump* that uses heuristics to (hopefully) generate a feasible solution, e.g. in D'Ambrosio et al. (2012).

SMT assesses the satisfiability of a model and, if the model is satisfiable, the SMT solver returns a witness. If the model is unsatisfiable the SMT solver can return an unsatisfiable core, a mutually unsatisfiable subset of model constraints. An unsatisfiable core is a useful tool when addressing why a model does not behave how we expect or to understand why a model fails.

Example 2.1. *Suppose that we wish to satisfy Equation* (2.1)*. Equation* (2.1) *combines SAT and the theory of real arithmetic.*

$$(x_1 \leq 1) \wedge (x_2 \leq 2) \wedge ((x_1 \geq 5) \vee (x_3 \leq 3)) \wedge (x_1 + x_2 + x_3 \geq 10) \tag{2.1}$$

For Equation (2.1)*, SMT leverages a SAT solver by replacing each inequality with auxiliary propositional variables, e.g.* $Y_1 = (x_1 \leq 1)$*, and assessing propositional satisfiability of the*

resulting formula:

$$Y_1 \wedge Y_2 \wedge (Y_3 \vee Y_4) \wedge Y_5. \tag{2.2}$$

The SAT solver returns an assignment satisfying Equation (2.2)*, e.g.* $Y_i = True, \forall\, i$*. Then, the real arithmetic theory solver checks the propositional variable meaning. Here, the theory solver deduces that the assignment is incorrect because we cannot have both* $Y_1 = (x_1 \leq 1) = True$ *and* $Y_3 = (x_1 \geq 5) = True$*. The theory solver encodes additional propositional clauses, e.g.* $(\neg Y_1 \vee \neg Y_3)$*, augments Equation* (2.2)*, and passes Equation* (2.3) *to the SAT solver:*

$$Y_1 \wedge Y_2 \wedge (Y_3 \vee Y_4) \wedge Y_5 \wedge (\neg Y_1 \vee \neg Y_3). \tag{2.3}$$

SMT iterates between the SAT and theory solvers until the algorithm terminates, in this case with a proof of unsatisfiability.

Example 1 suggests that the SAT and theory solvers are disjoint, but the most efficient and stable SMT tools integrate the two components (Sebastiani, 2007). Interaction between the theory solver and partial SAT solutions allow the theory solver to identify unsatisfiability in a partial assignment.

The efficacy of an SMT solver depends on the quality of the theory solver since a propositional encoding has to be created for the SAT solver. If the encoding is weak and the theory solver cannot strengthen it effectively, the SMT solver will, in worst case, enumerate all propositional solutions. Being built on top of a SAT solver, SMT natively support propositional variables and connectives, regardless of the theory. In terms of mathematical constraints, i.e. constraints relevant to this thesis, SMT theories include linear and nonlinear rational and real arithmetic. SMT solvers often implement exact arithmetic as they are often used in verification contexts. SMT solvers may output (i) *sat*, (ii) *unsat*, and (iii) *unknown* when assessing constraint feasibility. An output of *sat* corresponds to the constraint set being feasible and the solver will generally return a witness. An output of *unsat* corresponds to the constraint set being infeasible and, if supported, the solver can return an unsatisfiable core. An output of *unknown* corresponds

to the solver being unable to assess whether the constraint set is feasible or infeasible. The output of unknown may occur when using nonlinear theories, since SMT solvers often use exact arithmetic and nonlinear equation systems can be undecidable (Zhu, 2006; Liberti, Leo, 2019). An alternative approach to handling nonlinear equations is using δ-completeness (Gao et al., 2012). δ-completeness takes positive rational parameter δ and introduces the notion of δ-sat to the SMT solver which corresponds to proving feasibility of the relaxed constraint set where equations of the form $g(\boldsymbol{x}) = 0$ are relaxed to $|g(\boldsymbol{x})| \leq \delta$. Commonly used SMT solvers include Z3 (De Moura and Bjørner, 2008) and MathSAT5 (Cimatti et al., 2013). Z3 implements linear and nonlinear, rational and real arthimetic theory solvers. MATHSAT5 implements linear, rational and real arithmetic theory solvers. An SMT solver that supports δ-completeness is dReal (Gao et al., 2013).

2.1.3 Constraint Programming

SMT supports Boolean variables, integer variables, mathematical inequalities and propositional constraints. With these elements, we may model the constraint set of combinatorial optimisation problems. However, for some combinatorial optimisation constraint sets, being limited to mathematical inequalities can result in large formulations when modelling problem structure that is conceptually simple. Constraint programming (CP) (Van Hentenryck, 1989; Tsang, 1993; Rossi et al., 2006) addresses constraint satisfiability by designing specialised constraints and associated handling methods that are tailored for concepts that are difficult to model consisely using only propositional constraints and mathematical inequalities.

CP approaches generally apply (i) backtracking search, and (ii) constraint propagation. Backtracking search in CP is similar to that of the DPLL algorithm in SAT (Davis and Putnam, 1960; Davis et al., 1962), however CP variable domains may be larger than two or be continuous (SAT variable domains are all Boolean). Hence, CP may employ alternative approaches to divide the problem into subproblems (van Beek, 2006). Constraint propagation infers whether we can reason over a constraint, given the current state of the (sub-)problem, to reduce the domains of variables (Apt, 1999). For a constraint propagation example, consider CP constraint

`all_different(`x_1`, ..., `x_n`)` that models all variables in $\{x_1, \ldots, x_n\}$ taking different values. This constraint is applicable in the travelling salesman problem (Bellmore and Nemhauser, 1968), a problem that may need many variables and constraints if only using mathematical inequalities (Kulkarni and Bhave, 1985; Orman and Williams, 2007). Assume that all variables initially have domain $\{1, \ldots, n\}$ and x_1 is set to n, e.g. set during the search, then constraint propagation over the `all_different` constraint reduces the domains of $x_2, \ldots, x_n$ to $\{1, \ldots, n-1\}$.

To assess feasibility of constraint set I, CP solvers often explore a search tree that enumerates all solutions. At each node of the search tree, CP applies constraint propagation to reduce variable domains. The result at each node is either (i) a feasible solution is found, (ii) a variable's domain becomes empty after propagation, or (iii) constraint propagation cannot further reduce variable domains and all variable domains are non-empty. Case (i) terminates the algorithm, case (ii) corresponds to the local instance being infeasible and the search algorithm backtracks, and case (iii) employs branching, i.e. splitting the instance, to further the search. Enhancements to the CP solution algorithm include *nogood recording* to propagate local infeasibilities globally (Stallman and Sussman, 1977), *constraint propagation ordering heuristics* to reduce the time spent applying constraint propagation (Wallace and Freuder, 1992; Schulte and Stuckey, 2004, 2008), and *symmetry breaking* to prevent variables assignments that are equivalent in the search from being revisited (Puget, 1993, 2002, 2005a,b; Backofen and Will, 1999).

Difference between CP and SMT. CP and SMT differ in their handling of Boolean variables, propositional constraints and their solution strategies. Consider the disjunction

$$Y_1 \vee C_1(\boldsymbol{x}) \vee C_2(\boldsymbol{x}) \tag{2.4}$$

where $Y_1 \in \mathbb{B}$ and C_1, C_2 are constraints that do not involve Boolean variables or propositional connectives. SMT expects access to corresponding theory solvers for C_1 and C_2. SMT assesses feasibility of Equation (2.4) by relaxing to a propositional formula and testing the relaxation assignment against the theory solver. CP expects that corresponding constraint propagation methods are available for C_1, C_2 and propositional calculus. CP assesses feasibility of Equa-

tion (2.4) using search and constraint propagation. Specialised disjunction handling techniques include Van Hentenryck et al. (1998), Würtz and Müller (1996) and Lhomme (2003). As shown above, SMT hands all aspects related to propositional calculus to a SAT solver and invokes the remaining theory solver on a set of theory specific constraints. This constraint set may be smaller than the entire set of theory constraints, given the propositional assignment, e.g. $Y_1 = \text{True}$ in Equation (2.4). CP handles all constraint types directly in a single search tree. Hence, CP addresses the entire instance and can propagate or branch on theory elements before the propositional elements.

2.2 Mathematical Optimisation

Optimisation is often a natural follow-up question after constraint satisfaction. For example, after finding a feasible schedule for some set of tasks, we may then aim to find an optimal schedule that minimises the total completion time.

A mathematical optimisation problem is a constraint satisfaction problem combined with an optimisation objective. An optimisation objective is a function $f : \mathbb{R}^{n_C} \times \mathbb{B}^{n_B} \times \mathbb{Z}^{n_I} \to \mathbb{R}$. The optimisation objective assigns scores to the feasible solutions. We assume that the goal is always to minimise the objective function. Mathematical optimisation problems are often formulated as

$$\begin{aligned}
\min \quad & f(\boldsymbol{x}, \boldsymbol{Y}, \boldsymbol{z}) && && \text{(2.5a)}\\
\text{subject to} \quad & C_i(\boldsymbol{x}, \boldsymbol{Y}, \boldsymbol{z}), && \forall i \in [p] && \text{(2.5b)}\\
& \boldsymbol{x} \in \mathbb{R}^{n_c} && && \text{(2.5c)}\\
& \boldsymbol{Y} \in \mathbb{B}^{n_b} && && \text{(2.5d)}\\
& \boldsymbol{z} \in \mathbb{Z}^{n_I}, && && \text{(2.5e)}
\end{aligned}$$

where $C_i : \mathbb{R}^{n_C} \times \mathbb{B}^{n_B} \times \mathbb{Z}^{n_I} \to \mathbb{B}$ are constraints. Note that the conjunction between constraint functions is not explicitly stated.

Algorithm 1 (Sebastiani and Tomasi, 2015; Callia D'Iddio and Huth, 2017)

S: initial problem
if $S \neq \emptyset$ **then**
  return *infeasible*
end if
while $S \neq \emptyset$ **do**
  Select $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{Y}}, \hat{\boldsymbol{z}}) \in S$
  $S \leftarrow S \wedge (f(\boldsymbol{x}, \boldsymbol{Y}, \boldsymbol{z}) < f(\hat{\boldsymbol{x}}, \hat{\boldsymbol{Y}}, \hat{\boldsymbol{z}}))$
end while
return $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{Y}}, \hat{\boldsymbol{z}})$

2.2.1 Adapting a Constraint Satisfaction Solver

Assume that an instance of Problem (2.5) has an optimal solution with objective value f^*. Then the constraint set [Equations (2.5b) to (2.5e), $(f(\boldsymbol{x}, \boldsymbol{Y}, \boldsymbol{z}) < f^*)$]. On this principle, we may adapt a constraint satisfaction solver using Algorithm 1 (Sebastiani and Tomasi, 2015; Callia D'Iddio and Huth, 2017). Algorithm 1 treats the objective function as a bounded constraint and iteratively tightens the bound using the best found feasible objective. If the underlying constraint satisfaction solver does not support strict inequalities, we may add $(f(\boldsymbol{x}, \boldsymbol{Y}, \boldsymbol{z}) \leq f(\hat{\boldsymbol{x}}, \hat{\boldsymbol{Y}}, \hat{\boldsymbol{z}}) - \delta)$ on each iteration in Algorithm 1 where $\delta > 0$ is an appropriately chosen value to enforce ε-global optimality. Algorithm 1 is a simple approach for adapting a constraint satisfaction solver for optimisation, alternative approaches maintain both upper and lower bounds (Sebastiani and Tomasi, 2015; Callia D'Iddio and Huth, 2017).

2.2.2 Branch-and-Bound

Branch-and-bound (B&B) (Land and Doig, 1960) is versatile algorithmic framework capable of solving Problem (2.5) instances. Using a divide-and-conquer principle, B&B forms a tree of subproblems and searches the domain of feasible solutions. Key aspects of B&B are: (i) rigorous lower (upper) bounding methods for minimisation (maximisation) subproblems, (ii) branch selection, and (iii) feasible solution generation. In the worst case, B&B enumerates all solutions, but generally it avoids complete enumeration by pruning subproblems, i.e. removing infeasible subproblems or nodes with lower bound exceeding the best found feasible solution

Algorithm 2 Branch-and-Bound(S^{root}, f) (Morrison et al., 2016)

1: **Assumption:** f is bounded below over S^{root}
2: $Q \leftarrow \{S^{\text{root}}\}$ ▷ Nodes to explore
3: UB $\leftarrow \infty$
4: **while** $Q \neq \emptyset$ **do**
5: Select $S \in L$
6: **if** $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{Y}}, \hat{\boldsymbol{z}}) \in \{(\boldsymbol{x}, \boldsymbol{Y}, \boldsymbol{z}) \in S \mid f(\boldsymbol{x}, \boldsymbol{Y}, \boldsymbol{z}) < \text{UB}\}$ can be found **then** ▷ Heuristics
7: $(\boldsymbol{x}^*, \boldsymbol{Y}^*, \boldsymbol{z}^*) \leftarrow (\hat{\boldsymbol{x}}, \hat{\boldsymbol{Y}}, \hat{\boldsymbol{z}})$
8: UB $\leftarrow f(\hat{\boldsymbol{x}}, \hat{\boldsymbol{Y}}, \hat{\boldsymbol{z}})$
9: **end if**
10: **if** $\text{lb}(S, f) <$ UB **then** ▷ Bounding and Fathoming
11: Create partition P of S ▷ Branching
12: $Q \leftarrow Q \cup P$
13: **end if**
14: $Q \leftarrow Q \setminus \{S\}$
15: **end while**
16: **if** UB $< \infty$ **then**
17: **return** $(\boldsymbol{x}^*, \boldsymbol{Y}^*, \hat{\boldsymbol{z}^*})$
18: **else**
19: **return** *infeasible*
20: **end if**

(Morrison et al., 2016).

Algorithm 2 (Morrison et al., 2016) lists the main steps in B&B algorithms. Lines 6 to 8 search for the optimal solution. Line 7 derives rigorous lower bounds and, if possible, prunes the current node being explored. Line 11 constructs the new branches.

Upper bounds B&B Algorithm 2 initialises an upper bound of ∞. Assuming a feasible solution exists, Algorithm 2 generates a sequence of feasible solutions $\{(\boldsymbol{x}^{(i)}, \boldsymbol{Y}^{(i)}, \boldsymbol{z}^{(i)})\}_{i=1}^{k}$ satisfying $f\left(\boldsymbol{x}^{(i)}, \boldsymbol{Y}^{(i)}, \boldsymbol{z}^{(i)}\right) > f\left(\boldsymbol{x}^{(i+1)}, \boldsymbol{Y}^{(i+1)}, \boldsymbol{z}^{(i+1)}\right)$, $i \in [k-1]$ and returns the solution $(\boldsymbol{x}^{(k)}, \boldsymbol{Y}^{(k)})$ on line 17. The solution $(\boldsymbol{x}^{(k)}, \boldsymbol{Y}^{(k)}, \boldsymbol{z}^{(k)})$ satisfies ε-global optimality. General purpose heuristics for Problem (2.5) often derive an assignment $(\hat{\boldsymbol{Y}}, \hat{\boldsymbol{z}})$ and perform a local search on the remaining continuous problem to populate $\boldsymbol{x}$ (Fischetti and Lodi, 2011; Berthold, 2014; Sharma et al., 2016; Belotti and Berthold, 2017; Berthold, 2017).

Lower bounds Line 10 of Algorithm 2 derives a rigorous lower bound on $\min_{(\boldsymbol{x}, \boldsymbol{Y}, \boldsymbol{z}) \in S} f(\boldsymbol{x}, \boldsymbol{Y}, \boldsymbol{z})$. These lower bounds are calculated by relaxing a mathematical programming reformulation of

the problem instance. A mathematical programming reformulation of Problem (2.5) constructs an equivalent instance that replaces Boolean variables with binary variables (domain $\{0, 1\}$) and propositional constraints with mathematical constraints using reformulation techniques (Nemhauser and Wolsey, 1988; McCormick, 1976; Grossmann and Trespalacios, 2013). Section 2.2 discusses mathematical programming reformulations of Problem (2.5) instances. A relaxation to the mathematical programming reformulation is given by an auxiliary problem whose (i) feasible region contains the reformulation feasible region and (ii) objective bounds the reformulation objective from below. Relaxations approaches include continuous relaxations, Lagrangian relaxations (Geoffrion, 1974; Fisher, 1981; Tanaka and Araki, 2008; Rostami and Bagherpour, 2017), and Benders master problems (Benders, 1962; Hooker, 2007; Gendron et al., 2016; Moreno et al., 2019).

Fathoming and Branching Lines 10 to 12 of Algorithm 2 apply fathoming and branching. Algorithm 2 fathoms node S if $\text{lb}(S, f) \geq \text{UB}$ where $\text{lb}(S, f) \leq \min_{(\boldsymbol{x},\boldsymbol{Y},\boldsymbol{z})\in S} f(\boldsymbol{x}, \boldsymbol{Y}, \boldsymbol{z})$ ($\text{lb}(\cdot)$ is calculated using a relaxation). If a node cannot be fathomed then Algorithm 2 creates partition P of S, the subproblems $S' \in P$ are branches. Furthermore, every subset of P belongs to the same class of problems as S, i.e. B&B applies the same relaxation and branching strategy recursively.

Optimality Tolerances Algorithm 2 assumes all computation is exact. In practice, this assumption may hinder an optimality proof, e.g. using a finite precision representation might not be sufficient. Implementations often introduce tolerances to assess optimality, e.g. Ryoo and Sahinidis (1996); Tawarmalani and Sahinidis (2005); Misener and Floudas (2014).

Definition 2.2. *Let $\varepsilon > 0$ be a small constant, e.g. 10^{-6}. The absolute optimality gap is defined as* $\text{absgap}(l, u) = u - l$. *The relative optimality gap is defined as* $\text{relgap}_\varepsilon(l, u) = \frac{\text{absgap}(l,u)}{|l|+\varepsilon}$.

Algorithm 2 implementations often introduce an optimality tolerance for the fathoming decision on Line 10. Let $\varepsilon_{\text{opt}} > 0$ be a small constant, e.g. 10^{-6}. Algorithm 2 fathoms with an optimality tolerance by replacing the Line 10 condition with $\text{gap}(\text{lb}(S), \text{UB}) \leq \varepsilon_{\text{opt}}$ where

gap $\in \{\text{absgap}, \text{relgap}_\varepsilon\}$ for small $\varepsilon > 0$.

2.3 Mathematical Optimisation Frameworks

This section discusses frameworks that are capable of formulating (sub-)instances of Problem (2.5) and associated algorithms.

2.3.1 Mixed-Integer Convex Programming

Mixed-integer convex programming (convex MINLP) considers optimisation instances that can be formulated using continuous and integral variables, convex constraints and a convex objective function. Convex MINLP can capture many Problem (2.5) instances by considering binary variable equivalents, i.e. domain $\{0, 1\}$, for Boolean variables. Without loss of generality, convex MINLP is defined as (Kronqvist et al., 2019):

$$\begin{aligned}
\min \quad & \boldsymbol{c_1}^\top \boldsymbol{x} + \boldsymbol{c_2}^\top \boldsymbol{z} && && \text{(2.6a)}\\
\text{subject to} \quad & g_j(\boldsymbol{x}, \boldsymbol{z}) \leq 0, && j \in [p] && \text{(2.6b)}\\
& \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} \leq \boldsymbol{b}, && && \text{(2.6c)}\\
& \boldsymbol{z^L} \leq \boldsymbol{z} \leq \boldsymbol{z^U}, && && \text{(2.6d)}\\
& \boldsymbol{x} \in \mathbb{R}^{n_C}, \boldsymbol{z} \in \mathbb{Z}^{n_I}, && && \text{(2.6e)}
\end{aligned}$$

where n_C, and n_I are the number of continuous and integer variables, respectively. Parameters $\boldsymbol{c_1} \in \mathbb{R}^{n_C}$, $\boldsymbol{c_2} \in \mathbb{R}^{n_I}$ are cost coefficients on the continous and integer variables, respectively. Functions $g_j : \mathbb{R}^{n_C+n_I} \to \mathbb{R}$ are nonlinear convex functions. Parameters $\boldsymbol{A} \in \mathbb{R}^{q \times n_C}$, $\boldsymbol{B} \in \mathbb{R}^{q \times n_I}$, and $\boldsymbol{b} \in \mathbb{R}^q$ are linear constraint coefficients on the continuous variables, coefficients on the integer variables, and right-hand side, respectively. Parameters $\boldsymbol{z^L} \in \mathbb{Z}^{n_I}$, and $\boldsymbol{z^U} \in \mathbb{Z}^{n_I}$ are lower and upper bounds on the integer variables, respectively. Mixed-integer nonlinear programming (MINLP) generalises Problem (2.6) by allowing nonconvex nonlinearities in Equation (2.6b). Global MINLP solution algorithms may apply systematic convexification techniques (McCormick,

1976; Maranas and Floudas, 1992; Liu and Floudas, 1993) which construct convex MINLP approximations when deriving rigorous bounds on optimality.

Let S be a Problem (2.5) instance. Reformulate S to an equivalent instance S' that (i) does not contain logical operators $\rightarrow$ or $\leftrightarrow$ by applying logical equivalences, and (ii) propagate any $\neg$ operators by using distributivity and De Morgans laws such that all applications of $\neg$ in S' only occur on Boolean atoms or a mathematical constraint. If the $\neg$ operator does not apply to any occurence of a mathematical constraint in S', all mathematical constraints are convex, and all inequalities are non-strict, then an equivalent Problem (2.6) formulation is constructed using systematic reformulation techniques, e.g. McCormick (1976); Grossmann and Trespalacios (2013). Generally reformulation involves (i) replacing Boolean variables by binary equivalents, i.e. variables with domain $\{0, 1\}$, and propositional constraints with arithmetic constraints over the new binary variables, and (ii) decomposing complex interactions between logical and mathematical constraints into simpler separate propositional constraints and mathematical constraints by introducing fresh variables.

Continuous Relaxation

B&B implementations that solve Problem (2.6) instances often utilise a continuous relaxation. The continuous relaxation of Problem (2.6) is given by replacing $\boldsymbol{z} \in \mathbb{Z}^{n_I}$ with $\boldsymbol{z} \in \mathbb{R}^{n_I}$ in Problem (2.6e). Let $R^C(S)$ denote the continuous relaxation of Problem (2.6) instance S. We have that $\min_{(\boldsymbol{x},\boldsymbol{z})\in R^C(S)} \boldsymbol{c_1}^\top \boldsymbol{x} + \boldsymbol{c_2}^\top \boldsymbol{z} \leq \min_{(\boldsymbol{x},\boldsymbol{z})\in S} \boldsymbol{c_1}^\top \boldsymbol{x} + \boldsymbol{c_2}^\top \boldsymbol{z}$. Futhermore, if $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{z}}) \in \arg\min_{(\boldsymbol{x},\boldsymbol{z})\in R^C(S)} \boldsymbol{c_1}^\top \boldsymbol{x} + \boldsymbol{c_2}^\top \boldsymbol{z}$ and $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{z}}) \in S$ then $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{z}})$ an optimal solution of S.

The continuous relaxation of a Problem (2.6) instance is a convex optimisation problem. Classes of convex optimisation problems may be solved efficiently (Khachiyan, 1979; Karmarkar, 1984; Nesterov and Nemirovskii, 1994; Ben-Tal and Nemirovski, 2001; Boyd and Vandenberghe, 2004).

Cutting Plane Methods

B&B algorithms approach convex MINLP with divide-and-conquer, i.e. the problem is split into disjoint subproblems which are handled independently. The theory of convex optimisation (Boyd and Vandenberghe, 2004), allows for an alternative class of global optimisation algorithms called cutting plane methods.

Definition 2.3. *Let $S \subset \mathbb{R}^n$. A cutting plane for S is a pair $\boldsymbol{a} \in \mathbb{R}^n$, $b \in \mathbb{R}$ with $\boldsymbol{a} \neq \mathbf{0}$ such that for all $\boldsymbol{x} \in S$, $\boldsymbol{a}^\top \boldsymbol{x} \leq b$.*

Cutting plane methods solve convex MINLP by strengthening a single relaxation iteratively with cutting planes (Gomory, 1958; Duran and Grossmann, 1986b; Fletcher and Leyffer, 1994; Westerlund and Pettersson, 1995; Kronqvist et al., 2016).

Theorem 2.1 (Boyd and Vandenberghe (2004))**.** *Let $g : [\boldsymbol{x^L}, \boldsymbol{x^U}] \to \mathbb{R}$. The function g is convex if and only if*

$$g(\boldsymbol{x}) \geq g(\hat{\boldsymbol{x}}) + \nabla g(\hat{\boldsymbol{x}})^\top (\boldsymbol{x} - \hat{\boldsymbol{x}}), \quad \forall \boldsymbol{x} \in [\boldsymbol{x^L}, \boldsymbol{x^U}]. \tag{2.7}$$

Let $S(g) = \{\boldsymbol{x} \in [\boldsymbol{x^L}, \boldsymbol{x^U}] \,|\, g(\boldsymbol{x}) \leq 0\}$. The following consequences of Theorem 2.1 motivate cutting plane algorithms for convex MINLP:

$$S(g) = \{\boldsymbol{x} \in [\boldsymbol{x^L}, \boldsymbol{x^U}] \,|\, g(\hat{\boldsymbol{x}}) + \nabla g(\hat{\boldsymbol{x}})^\top (\boldsymbol{x} - \hat{\boldsymbol{x}}) \leq 0, \hat{\boldsymbol{x}} \in [\boldsymbol{x^L}, \boldsymbol{x^U}]\} \tag{2.8}$$

$$S(g) \subseteq \{\boldsymbol{x} \in [\boldsymbol{x^L}, \boldsymbol{x^U}] \,|\, g(\hat{\boldsymbol{x}}) + \nabla g(\hat{\boldsymbol{x}})^\top (\boldsymbol{x} - \hat{\boldsymbol{x}}) \leq 0, \hat{\boldsymbol{x}} \in X \subseteq [\boldsymbol{x^L}, \boldsymbol{x^U}]\}, \tag{2.9}$$

i.e. the first-order Taylor approximations of g at $\hat{\boldsymbol{x}} \in X \subseteq [\boldsymbol{x^L}, \boldsymbol{x^U}]$ forms a linear relaxation of $g(\boldsymbol{x}) \leq 0$. When X is finite, we may construct a finite relaxation for $g(\boldsymbol{x}) \leq 0$ by replacing the nonlinear convex constraint with $|X|$ linear inequalities as defined by Equation (2.9). The relaxation may be tightened by considering further linearisation points from $[\boldsymbol{x^L}, \boldsymbol{x^U}] \setminus X$.

Duran and Grossmann (1986b) present an outer approximation approach for convex MINLPs where the integer variables only participate linearly. The outer approximation algorithm

iterates between solving an MILP master problem and an NLP subproblem. The MILP master approximates all nonlinear functions with a linear outer approximation. Solving the MILP master derives lower bounds on the optimum and provides a candidate assignment $\hat{\boldsymbol{z}}$ on the integer variables. The original problem where $\boldsymbol{z} = \hat{\boldsymbol{z}}$ forms a convex NLP which is solved to optimality. If the NLP is infeasible, a cut excluding $\hat{\boldsymbol{z}}$ is added to the MILP master. Otherwise, a linearisation at $(\bar{\boldsymbol{x}}, \hat{\boldsymbol{y}})$, the optimal solution of the NLP, for each nonlinear function is added to the MILP master. Fletcher and Leyffer (1994) extend the Duran and Grossmann (1986b) algorithm to handle convex MINLP instances where integer variables participate nonlinearly.

Applying cutting plane approaches in a B&B algorithm gives the branch-and-cut framework. Branch-and-cut generates cutting planes at nodes of the search tree. Cut generation at a node level can improve local lower bounds and reduce the size of the search tree. Branch-and-cut has been extensively applied in solving MILPs (Balas et al., 1993, 1996; Ceria et al., 1998). Stubbs and Mehrotra (1999) develop a branch-and-cut algorithm for bounded convex MINLPs where all integral variables are binary, i.e. domain $\{0, 1\}$. Their cutting plane generation extends the Balas et al. (1993) cutting plane generation for 0-1 MILPs to the 0-1 convex MINLP case. Bonami (2011) develops lift-and-project cuts for convex MINLP. Branch-and-cut is also applied to nonconvex MINLP (Tawarmalani and Sahinidis, 2005).

Mixed-Integer Linear Programming

When $p = 0$, Problem (2.6) also belongs to the subclass of mixed-integer linear programming (MILP). MILP continuous relaxations are linear programs (LPs). We may solve LPs with interior point methods, e.g. Karmarkar (1984), or with the simplex algorithm (Dantzig, 1998). While the simplex algorithm exhibits poor theoretical worst-case performance (Klee and Minty, 1972), average performance is competetive. Robust commercial codes often implement both interior point and simplex methods for LPs.

Mixed-Integer Second-Order Cone Programming Mixed-integer second-order cone programming (MISOCP) considers Problem (2.6) instances where the Equation (2.6b) are second-

order cone constraints (Ben-Tal and Nemirovski, 2001), i.e. of the form:

$$\|\boldsymbol{A}_{\boldsymbol{x},i}\boldsymbol{x} + \boldsymbol{A}_{\boldsymbol{z},i}\boldsymbol{z} + \boldsymbol{b}_{\boldsymbol{i}}\|_2 \leq \boldsymbol{d}_{\boldsymbol{x},\boldsymbol{i}}^\top \boldsymbol{x} + \boldsymbol{d}_{\boldsymbol{z},\boldsymbol{i}}^\top \boldsymbol{z} + e_i, \quad i \in [p]. \tag{2.10}$$

An MISOCP continuous relaxation is a second-order cone program (SOCP). SOCPs can be efficiently solved using interior point methods (Nesterov and Nemirovskii, 1994). Let S be an SOCP instance. Interior point methods solve a sequence $\{S^{(j)}\}$ of auxiliary problems whose corresponding optimal solutions form a sequence that converges to an optimal solution of S (Nesterov and Nemirovskii, 1994; Ben-Tal and Nemirovski, 2001; Boyd and Vandenberghe, 2004). In particular, when an interior point method solves an SOCP, each auxiliary problem iterate will always satisfy any corresponding Equation (2.10) constraint strictly. We explain using optimisation Problem (2.11) below:

$$\begin{aligned}
\min \quad & \boldsymbol{c}^\top \boldsymbol{x} && && \text{(2.11a)}\\
\text{subject to} \quad & \boldsymbol{A}^{(\boldsymbol{i})}\boldsymbol{x} + \boldsymbol{s}^{(\boldsymbol{i})} = \boldsymbol{b}^{(\boldsymbol{i})}, && i \in [p] && \text{(2.11b)}\\
& \boldsymbol{a}_{\boldsymbol{0}}^{(\boldsymbol{i})\top}\boldsymbol{x} + s_0^{(i)} = b_0^{(i)}, && i \in [p] && \text{(2.11c)}\\
& \left\|\boldsymbol{s}^{(\boldsymbol{i})}\right\|_2 \leq s_0^{(i)}, && i \in [p] && \text{(2.11d)}\\
& \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{s}^{(\boldsymbol{i})} \in \mathbb{R}^{m_i}, s_0^{(i)} \in \mathbb{R}, && && \text{(2.11e)}
\end{aligned}$$

where $\boldsymbol{A}^{(\boldsymbol{i})} \in \mathbb{R}^{m_i \times n}$, $\boldsymbol{a}_{\boldsymbol{0}}^{(\boldsymbol{i})} \in \mathbb{R}^n$, $\boldsymbol{b}^{(\boldsymbol{i})} \in \mathbb{R}^{m_i}$ and $b_0^{(i)} \in \mathbb{R}$. An interior point method constructs an auxiliary problem of Problem (2.11) using a barrier function.

Definition 2.4 (Ben-Tal and Nemirovski (2001))**.** *Let $\mathcal{X}$ be a closed convex set and $\mathcal{F} : \operatorname{int}\mathcal{X} \to \mathbb{R}$ a well-defined, smooth, strongly convex function. We call $\mathcal{F}$ a barrier function if for all sequences $\{\boldsymbol{x}^{(\boldsymbol{i})}\}_{i=1}^{\infty}$, $\boldsymbol{x}^{(\boldsymbol{i})} \in \operatorname{int}\mathcal{X}$:*

$$\lim_{i\to\infty} \boldsymbol{x}^{(\boldsymbol{i})} \in \partial\mathcal{X} \implies \lim_{i\to\infty} \mathcal{F}\left(\boldsymbol{x}^{(\boldsymbol{i})}\right) = \infty.$$

In particular, $\mathcal{F}_L\left(s_0^{(i)}, \boldsymbol{s}^{(\boldsymbol{i})}\right) = -\ln\left(s_0^{(i)2} - \boldsymbol{s}^{(\boldsymbol{i})\top}\boldsymbol{s}^{(\boldsymbol{i})}\right)$ is a barrier function for the constraint $\left\|\boldsymbol{s}^{(\boldsymbol{i})}\right\|_2 \leq s_0^{(i)}$. The auxiliary problem associated with Problem (2.11) removes the Equa-

tion (2.11d) constraints and replaces the Equation (2.11a) objective with

$$\min \boldsymbol{c}^\top \boldsymbol{x} + t \sum_{i \in [p]} \mathcal{F}_L \left(s_0^{(i)}, \boldsymbol{s^{(i)}} \right)$$

for some $t > 0$. Clearly, as $\|\boldsymbol{s^{(i)}}\|_2 \to s_0^{(i)}$, $\mathcal{F}_L(s_0^{(i)}, \boldsymbol{s^{(i)}}) \to \infty$. Hence, to apply an interior point method, the feasible region must be strictly feasible with respect to constraints that are associated with a barrier (Slater, 1959).

An alternative to interior point methods is a cutting plane approach. Vielma et al. (2017) study polyhedral relaxations of SOCP constraints and propose an outer approximating strategy that supports iterative improvement in its approximation quality.

2.3.2 Disjunctive Programming

Balas developed disjunctive programming in the 1970s (Balas, 1974, 1975, 1977, 1979, 2018). A disjunctive program is given by a linear objective and the disjunction of systems of linear constraints:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{s.t.} \quad & \bigvee_{i=1}^{k} \boldsymbol{A}_i \boldsymbol{x} \leq \boldsymbol{b}_i \\ & \boldsymbol{x} \in \mathbb{R}^n, \end{aligned} \tag{2.12}$$

with parameters $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{A}_i \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b}_i \in \mathbb{R}^m$ $\forall i = 1, \ldots, k$. The disjunctive program in Equation (2.12) may be written equivalently as a mixed-integer linear program (MILP) by introducing binary variables $\boldsymbol{y}$ and auxiliary continuous disaggregated variables $\boldsymbol{\nu_i}$ (Hooker,

2002):

$$\begin{aligned}
\min \quad & \boldsymbol{c}^\top \boldsymbol{x} && && \text{(2.13a)}\\
\text{s.t.} \quad & \boldsymbol{A_i \nu_i} \leq \boldsymbol{b}_i y_i, && \forall i = 1, \ldots, k && \text{(2.13b)}\\
& \boldsymbol{x} = \boldsymbol{\nu_1} + \ldots + \boldsymbol{\nu_k} && && \text{(2.13c)}\\
& \sum_{i=1}^{k} y_i = 1 && && \text{(2.13d)}\\
& \boldsymbol{x^L} y_i \leq \boldsymbol{\nu_i} \leq \boldsymbol{x^U} y_i && \forall i = 1, \ldots, k && \text{(2.13e)}\\
& \boldsymbol{\nu_i} \in \mathbb{R}^n && \forall i = 1, \ldots, k && \text{(2.13f)}\\
& \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \{0,1\}^k. && && \text{(2.13g)}
\end{aligned}$$

Disjunctive programming allows model developers to write certain problems more concisely and/or more meaningfully. For example, selecting one element i out of a set $i \in \{\, 1, \ldots, n \,\}$, i.e. set partitioning, is a common constraint in process systems engineering:

$$\sum_{i=1}^{n} y_i = 1 \text{ where } y_i \in \{\, 0, 1 \,\}. \tag{2.14}$$

As a disjunctive constraint, Equation (2.14) is easily identified as a selection constraint:

$$\bigvee_{i=1}^{n} \left(y_i = 1 \wedge \bigwedge_{i \neq j} y_j = 0 \right).$$

Most approaches for solving disjunctive programs replace propositional variables with binary variables. In the resulting model, $y_i = 1$ implies a set of active constraints and $y_i = 0$ implies associated inactive constraints. Disjunctive programming has a rich cutting plane and duality theory (Balas, 1974, 1975, 1979; Jeroslow, 1977). Furthermore, Balas (1998) characterises the convex hull of a disjunctive program. A disjunctive constraint is satisfied if at least one of its disjuncts is satisfied, Beaumont (1990) develops a branch-and-bound algorithm that branches on disjunctions directly, Beaumont (1990) considers a more general version of Equation (2.12) that allows for multiple disjunctive constraints.

2.3.3 Convex Generalized Disjunctive Programming

Convex generalized disjunctive programming (convex GDP) extends convex NLP with Boolean variables, disjunctions of groups of mathematical constraints and propositional variables (Raman and Grossmann, 1994; Grossmann and Ruiz, 2012; Grossmann and Trespalacios, 2013).

Without loss of generality, a convex GDP formulation is (Grossmann and Trespalacios, 2013):

$$\begin{aligned}
\min \quad & \boldsymbol{c}^\top \boldsymbol{x} && && \text{(2.15a)}\\
& g_j(\boldsymbol{x}) \leq 0, && j \in [p] && \text{(2.15b)}\\
& \bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ r_{ikj}(\boldsymbol{x}) \leq 0, \quad j \in [p_{ik}] \end{bmatrix}, && k \in K && \text{(2.15c)}\\
& \veebar_{i \in D_k} Y_{ik}, && k \in K && \text{(2.15d)}\\
& \Omega(\boldsymbol{Y}), && && \text{(2.15e)}\\
& \boldsymbol{x}^{\boldsymbol{L}} \leq \boldsymbol{x} \leq \boldsymbol{x}^{\boldsymbol{U}}, && && \text{(2.15f)}\\
& \boldsymbol{x} \in \mathbb{R}^n, && && \text{(2.15g)}\\
& Y_{ik} \in \{\text{True}, \text{False}\}, && k \in K, i \in D_k, && \text{(2.15h)}
\end{aligned}$$

where $g_j : \mathbb{R}^{n_C} \to \mathbb{R}$, $r_{ikj} : \mathbb{R}^{n_C} \to \mathbb{R}$ are convex, $\Omega(\boldsymbol{Y})$ is a propositional formula, and $\boldsymbol{c} \in \mathbb{R}^{n_C}$. Equation (2.15a) is the optimisation objective. Equation (2.15b) are global constraints. Equation (2.15c) are disjunctive constraints. Equation (2.15d) selects that exactly one disjunct in each disjunction K. Equation (2.15e) defines a propositional constraint on the Boolean variables. Equation (2.15f) are bounds on the continuous variables. Equations (2.15g) and (2.15h) are the domains of the model variables. The more general GDP framework allows for nonconvex g_j or r_{ikj} in Problem (2.15) (Jackson and Grossmann, 2001; Grossmann and Ruiz, 2012). Nonconvex GDP solution algorithms may consider convex GDP relaxations (Lee and Grossmann, 2001).

Basic Steps

A Problem (2.15) instance with $|K| \geq 1$ and $|K| + p \geq 2$ has more than one equivalent representation (Balas, 1985; Sawaya and Grossmann, 2012; Ruiz and Grossmann, 2012). These equivalent representations are constructed by applying basic steps. Basic steps in convex GDP are an application of distributivity of $\wedge$ over $\vee$. Let K' be a subset of disjunctions and global constraints (here we treat global constraints as a disjunctions of length 1). We define the following for notational convenience:

$$\begin{aligned} d_{00} &\equiv \Big[g_j(\boldsymbol{x}) \leq 0, \quad j \in p_0 \Big] \\ d_{ik} &\equiv \Big[r_{ikj}(\boldsymbol{x}) \leq 0, \quad j \in [p_{ik}] \Big], \qquad\qquad k \in K, i \in D_k \end{aligned}$$

where $p_0 \subseteq [p]$. Let $D_0 = \{0\}$, $K' \subseteq \{0\} \cup K$, $|K| \geq 2$, and $|p_0| \geq 1$ if $0 \in K'$. The basic step over K' is:

$$\bigwedge_{k \in K'} \bigvee_{i \in D_k} d_{ik} \equiv \bigvee_{I \in \prod_{k \in K'} \mathrm{zip}(k, D_k)} \bigwedge_{(k,\{i\}) \in I} d_{ik} \tag{2.16}$$

where $\mathrm{zip}(k, D) = \{(k, \{i\}) \,|\, i \in D\}$. The basic step replaces the conjunction of constraints indexed by K' by a single equivalent disjunction.

Balas (1985) proposes basic steps for disjunctive programming as a means of utilising the logical structure to systematically construct tighter continuous relaxations. Sawaya and Grossmann (2012) and Ruiz and Grossmann (2012) extend this theory to linear GDP and convex GDP, respectively. While basic steps can improve continuous relaxation tightness, as Equation (2.16) shows, the resulting disjunction has $\prod_{k \in K'} |D_k|$ disjuncts. However, it may be possible to reduce the size of the disjunction if some of the resulting disjuncts are infeasible (Trespalacios and Grossmann, 2015a).

Mathematical Programming Reformulations of Convex GDPs

Convex GDPs are often reformulated as convex MINLPs when solving either the original convex GDP or a continuous relaxation (Caballero and Grossmann, 2001; Castro and Marques, 2015; Jonuzaj et al., 2016). Only Equations (2.15c) to (2.15e) and (2.15h) involve variables and operators that are not allowed in mathematical programming.

A mathematical programming reformulation takes the form:

$$\begin{aligned}
\min \quad & \boldsymbol{c}^\top \boldsymbol{x} && && \text{(2.17a)}\\
\text{subject to} \quad & \text{[reformulation of disjunction } k\text{]}, && k \in K, && \text{(2.17b)}\\
& \sum_{i \in D_k} y_{ik} = 1, && k \in K, && \text{(2.17c)}\\
& \boldsymbol{A}\boldsymbol{y} \leq \boldsymbol{b}, && && \text{(2.17d)}\\
& y_{ik} \in \{0, 1\}, && k \in K,\ i \in D_k, && \text{(2.17e)}\\
& \text{Equations (2.15b), (2.15f) and (2.15g).} && && \text{(2.17f)}
\end{aligned}$$

Equation (2.17a) is the unchanged objective. Problem (2.17) introduces a binary variable y_{ik} for each Boolean variable Y_{ik}. Equation (2.17b) introduces a set of constraints and variables depending on the reformulation strategy (discussed below). Equation (2.17c) is equivalent to Equation (2.15d). Equation (2.17d) is a transformation of Equation (2.15e), see Grossmann and Trespalacios (2013). Equation (2.17e) are variable domains. Equation (2.17f) are unchanged constraints. The Equation (2.17b) reformulation can involve big-M and hull reformulations for the disjunctions.

Big-M reformulation. Reformulating disjunction $k \in K$ with a big-M approach introduces

$$r_{ikj}(\boldsymbol{x}) \leq M_{ikj}(1 - y_{ik}), \quad i \in D_k,\ j \in [p_{ik}] \tag{2.18}$$

in place of Equation (2.17b) (Nemhauser and Wolsey, 1988; Trespalacios and Grossmann, 2014). The tightest value for Equation (2.18) parameter M_{ikj}, i.e. the smallest value that does not

exclude any feasible solutions, is given by:

$$\sup\left\{r_{ikj}(\boldsymbol{x}) \,\middle|\, g(\boldsymbol{x}) \leq \mathbf{0}, \boldsymbol{x} \in T, \boldsymbol{x} \in \left[\boldsymbol{x}^{\boldsymbol{L}}, \boldsymbol{x}^{\boldsymbol{U}}\right]\right\}, \quad j \in [p_{ik}], \tag{2.19}$$

where $\sup \emptyset = -\infty$, and

$$S = \left\{ \boldsymbol{y} \,\middle|\, \boldsymbol{A}\boldsymbol{y} \leq \boldsymbol{b};\ \sum_{i \in D_k} y_{ik} = 1,\ k \in K;\ y_{ik} \in \{0,1\},\ k \in K,\ i \in D_k \right\}, \tag{2.20}$$

$$T = \bigcup_{\boldsymbol{y} \in S} \left\{ \boldsymbol{x} \,|\, r_{ikj}(\boldsymbol{x}) \leq \mathbf{0},\ (i,k) \in \{(i,k) \,|\, y_{ik} = 1\},\ j \in [p_{ik}] \right\}. \tag{2.21}$$

The Equation (2.19) bound considers all (exponential) assignments. Hence, calculating the Equation (2.19) bound has the same time complexity as finding an optimal solution by enumerating all assignments. As M_{ikj} is a modelling parameter, it is often more efficient to consider an upper bound on Equation (2.19) that admits a simpler calculation, e.g. $\sup_{\boldsymbol{x} \in [\boldsymbol{x}^{\boldsymbol{L}}, \boldsymbol{x}^{\boldsymbol{U}}]} r_{ikj}(\boldsymbol{x})$.

Trespalacios and Grossmann (2015b) propose an improvement for GDP big-M reformulations by considering substituting Equation (2.17c) into Equation (2.18) for y_{ik}. This substitution gives (Trespalacios and Grossmann, 2015b):

$$r_{ikj}(\boldsymbol{x}) \leq \sum_{i' \in D_k \setminus \{i\}} M_{ii'kj} y_{i'k}.$$

A direct substitution gives $M_{ii'kj} = M_{ikj}$, however $M_{ii'kj} \leq M_{ikj}$ may also be attainable for some of the $i' \in D_k \setminus \{i\}$. Trespalacios and Grossmann (2015b) show that using this approach can result in tighter root node relaxations, reduced solve times and a lower number of explored nodes.

Hull reformulation. Reformulating disjunction $k \in K$ with a hull approach introduces (Grossmann and Lee, 2003):

$$y_{ik} r_{ikj}(\boldsymbol{\nu_{ik}}/y_{ik}) \leq 0, \qquad i \in D_k,\, j \in [p_{ik}] \tag{2.22a}$$

$$\boldsymbol{x} = \sum_{i \in D_k} \boldsymbol{\nu_{ik}} \tag{2.22b}$$

$$\boldsymbol{x^L} y_{ik} \leq \boldsymbol{\nu_{ik}} \leq \boldsymbol{x^U} y_{ik}, \qquad i \in D_k \tag{2.22c}$$

$$\boldsymbol{\nu_{ik}} \in \mathbb{R}^n, \qquad i \in D_k \tag{2.22d}$$

in place of Equation (2.17b). The left-hand side of Equation (2.22a) is the perspective transformation of r_{ikj}, defined as (Rockafellar, 1970; Hiriart-Urruty and Lemaréchal, 1993):

$$y_{ik} r_{ikj}(\boldsymbol{\nu_{ik}}/y_{ik}) = \begin{cases} y_{ik} r_{ikj}(\boldsymbol{\nu_{ik}}/y_{ik}), & \text{if } y_{ik} > 0 \\ 0, & \text{if } y_{ik} = 0 \text{ and } \boldsymbol{\nu_{ik}} = 0. \end{cases} \tag{2.23}$$

Note that Equation (2.22c) ensures that constraint set (2.22) evaluates the perspective transformation where it is defined.

Mathematical programming solvers derive bounds by considering the relaxation of a convex approximation of the reformulation. The relaxation of a convex GDP reformulation is given by relaxing the integrality requirement on binary variables y_{ik}, i.e. Equation (2.17e) becomes

$$y_{ik} \in [0, 1], \quad k \in K, i \in D_k. \tag{2.24}$$

For fixed $k \in K$, when r_{ikj}, $i \in D_k$, $j \in [p_{ik}]$ are convex, Equations (2.17c), (2.22) and (2.24) limited to k define the convex hull of disjunction k (Grossmann and Lee, 2003). Hence, for a given GDP instance, a hull relaxation is always at least as tight as a big-M relaxation.

While the hull reformulation does not require any additional parameters, it does require many additional variables $\boldsymbol{\nu_{ik}}$ and can cause numerical issues as $y_{ik} \to 0$ when used directly. The division-by-zero issue can be avoided for disjunctions that only contain linear or SOCP constraints (Balas, 1998; Ben-Tal and Nemirovski, 2001). For general nonlinear convex r_{ikj}, Grossmann and

Lee (2003) propose

$$(y_{ik} + \varepsilon) r_{ikj}(\boldsymbol{\nu_{ik}}/(y_{ik} + \varepsilon)) \leq 0 \tag{2.25}$$

for small $\varepsilon > 0$ as an approximation for Equation (2.22a) that avoids a division-by-zero. However, Equation (2.25) is incorrect for $y_{ik} \in \{0, 1\}$. Furthermore, Sawaya and Grossmann (2007) identify that ε may have to be very small to satisfy solver tolerances when $y_{ik} = 0$, which can cause numerical issues. Sawaya (2006) suggests replacing Equation (2.22a) with

$$((1 - \varepsilon) y_{ik} + \varepsilon) r_{ikj}(\boldsymbol{\nu_{ik}}/((1 - \varepsilon) y_{ik} + \varepsilon)) - \varepsilon r_{ik}(\mathbf{0})(1 - y_{ik}) \leq 0 \tag{2.26}$$

for small $\varepsilon > 0$. Equation (2.26) is correct for $y_{ik} \in \{0, 1\}$, however $r_{ikj}(0)$ must be defined and $\mathbf{0} \leq \boldsymbol{\nu_{ik}} \leq \boldsymbol{x}^U y_{ik}$ must hold. Furman et al. (2020) establish theoretical properties relating to correctness, convexity and tightness of the Equation (2.26) perspective reformulation. Alternatively, we may handle the perspective function directly in a solver. Frangioni and Gentile (2006) propose *perspective cuts* that outer approximate the perspective function. Hence, a cutting plane algorithm that lazily generates perspective cuts may be employed.

Since convex GDPs can be reformulated as convex MINLPs, any supporting convex MINLP solver can be used directly to solve the convex GDP instance. Alternatively, we can design convex GDP-specific cutting plane methods or B&B algorithms.

Cutting Plane Methods for Convex Generalized Disjunctive Programming

Let S be a convex GDP instance. We denote the hull relaxation of S with $R^H(S)$.. Let S^* be a Problem (2.15) convex GDP instance with $p = 0$ and $|K| = 1$. Then any $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \text{proj}_{(\boldsymbol{x},\boldsymbol{y})} \arg\min_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\nu}) \in R^H(S^*)} \boldsymbol{c}^\top \boldsymbol{x}$ can be converted into an optimal solution $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{Y}})$ of S^*. Consider an arbitrary convex GDP instance S. If S has global constraints or more than one disjunction then basic steps systematically build a finite sequence of equivalent instances that terminates at an instance satisfying the required property for S^*. We have that $R^H(S^*) \subseteq R(S)$ where $R(S)$ denotes a big-M or hull relaxation of S.

Given a solution $\hat{\boldsymbol{x}} \in \text{proj}_{\boldsymbol{x}} R(S)$, either $\hat{\boldsymbol{x}} \in \text{proj}_{\boldsymbol{x}} R^H(S^*)$ or $\hat{\boldsymbol{x}} \notin \text{proj}_{\boldsymbol{x}} R^H(S^*)$. For the

former case, $\hat{\boldsymbol{x}}$ can be converted into an optimal solution whereas for the latter we can generate a cutting plane for $R(S)$ to separate $\hat{\boldsymbol{x}}$. A cutting plane $\boldsymbol{a}^\top \boldsymbol{x} \leq b$ for $R(S)$ that separates $\hat{\boldsymbol{x}} \notin \text{proj}_{\boldsymbol{x}}\, R^H(S^*)$ must satisfy $\boldsymbol{a}^\top \hat{\boldsymbol{x}} > b$. We derive the pair $(\boldsymbol{a}, b)$ by solving a separation problem (Stubbs and Mehrotra, 1999; Vecchietti et al., 2003):

$$\min \quad \|\boldsymbol{x} - \hat{\boldsymbol{x}}\| \tag{2.27a}$$

$$\text{subject to} \quad \boldsymbol{x} \in \text{proj}_{\boldsymbol{x}}\, R'(S'), \tag{2.27b}$$

where S' is problem S after applying zero or more basic steps and $R'(\cdot)$ is a relaxation strategy. Equation (2.27a) minimises the distance of the solution from $\hat{\boldsymbol{x}}$ for some norm. Equation (2.27b) constrains $\hat{\boldsymbol{x}}$ to $R'(S')$. In practice $R'(S)$ is generally a constructed by taking a hull reformulation of S or applying basic steps to S (Vecchietti et al., 2003; Trespalacios and Grossmann, 2016a). When $\hat{\boldsymbol{x}} \in \text{proj}_{\boldsymbol{x}}\, R'(S')$, the optimal solution to Problem (2.27) is $\bar{\boldsymbol{x}} = \hat{\boldsymbol{x}}$ with optimal objective of 0. If $R'(S') = R^H(S)$ then $\hat{\boldsymbol{x}}$ is optimal for S. If $R'(S') \neq R^H(S)$ then we have to construct a tighter constraint set $R''(S'')$, e.g. by making further basic steps or considering a hull relaxation. When $\hat{\boldsymbol{x}} \notin \text{proj}_{\boldsymbol{x}}\, R'(S')$, the optimal solution to Problem (2.27) is $\bar{\boldsymbol{x}} \neq \hat{\boldsymbol{x}}$ with optimal objective greater than 0. The linear inequality:

$$\boldsymbol{\xi}^\top (\boldsymbol{x} - \hat{\boldsymbol{x}}) \geq 0, \tag{2.28}$$

where $\boldsymbol{\xi}$ is a subgradient of $\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|$ at $\bar{\boldsymbol{x}}$, can define a valid cutting plane for $R(S)$ that separates $\hat{\boldsymbol{x}}$ (Stubbs and Mehrotra, 1999). Among the subgradients, there exists $\boldsymbol{\xi}^*$ for which Equation (2.28) separates $\hat{\boldsymbol{x}}$, i.e. $\hat{\boldsymbol{x}} \notin \{\boldsymbol{x} \,|\, \boldsymbol{\xi}^{*\top}(\boldsymbol{x} - \hat{\boldsymbol{x}}) \geq 0\} \cap \text{proj}_{\boldsymbol{x}}\, R(S)$. Cutting plane methods maintain a convex GDP master problem that extends the Problem (2.17) constraint set with additional linear cuts. The number of cuts increases as the algorithm progresses.

Vecchietti et al. (2003) propose using cutting planes in a branch-and-cut framework that maintains a big-M reformulated GDP where cutting planes are generated against a hull reformulation. Sawaya and Grossmann (2005) investigate the effect of the Vecchietti et al. (2003) cutting plane approach and the choice of the Equation (2.27a) norm when applied at the root node of a

B&B algorithm for linear GDP. Trespalacios and Grossmann (2016a) extend the Vecchietti et al. (2003) approach by considering a tighter separation problem constraint set generated by applying basic steps.

Cutting plane strategies iteratively separate infeasible relaxation solutions with the explicit introduction of a cutting plane. As investigated by Trespalacios and Grossmann (2016a), the application of basic steps can induce tighter cuts when considering a hull relaxation. The application of basic steps combined with a hull relaxation may be viewed as an implicit simultaneous introduction of many cutting planes. Trespalacios and Grossmann (2015a) propose a convex GDP solution strategy that maintains a single hull reformulated disjunction and the remaining disjunctions are big-M reformulated. Their algorithm iteratively aggregates basic steps into the hull relaxed disjunction.

Branch-and-Bound Methods for Convex Generalized Disjunctive Programming

B&B methods for convex GDP branch over disjunctions, i.e. splitting a disjunctions into subdisjunctions. Candidates for branching are identified by considering fractionality in their continuous relaxations.

Definition 2.5 (Lee and Grossmann (2000))**.** *Let $\hat{\boldsymbol{y}}$ be an assignment on binary variables $\boldsymbol{y}$ for some relaxation solution of convex GDP problem S.*

Disjunction $k \in K$ (i) selects disjunct $i \in D_k$ under $\hat{\boldsymbol{y}}$ if $\hat{y}_{ik} = 1$; (ii) is integer feasible under $\hat{\boldsymbol{y}}$ if, for some $i \in D_k$, it selects disjunct i; and (iii) is fractional under $\hat{\boldsymbol{y}}$ if it is not integer feasible under $\hat{\boldsymbol{y}}$.

The assignment $\hat{\boldsymbol{y}}$ is: (i) integer feasible for S if every disjunction $k \in K$ is integer feasible under $\hat{\boldsymbol{y}}$; and (ii) fractional for S if some disjunction $k \in K$ is fractional under $\hat{\boldsymbol{y}}$.

Beaumont (1990) develops a B&B algorithm for disjunctive programming that, at each node with fractional assignment $\hat{\boldsymbol{y}}$, branches by partitioning fractional disjunction k to construct $|D_k|$ children. Lee and Grossmann (2000) generalise the Beaumont (1990) B&B algorithm to

convex GDP. The algorithm considers hull relaxations at each B&B node. If the relaxation binary assignment is not integer feasible, the branching strategy branches on the binary variable with the largest fractional assignment. Trespalacios and Grossmann (2016b) study Lagrangian relaxations for linear GDPs and develop a heuristic that is embedded into the Beaumont (1990); Lee and Grossmann (2000) B&B algorithm. B&B methods for nonconvex GDP utilise spatial branching and convexification techniques (McCormick, 1976; Smith and Pantelides, 1997; Lee and Grossmann, 2001; Kirst et al., 2017).

2.3.4 Mixed Logical-Linear Programming

Mixed logical-linear programming (MLLP) is formulated (Hooker and Osorio, 1999):

$$\begin{aligned} &\min \ \ \boldsymbol{c}^\top \boldsymbol{x} \\ &\text{s.t.} \quad \Omega_j^p(\boldsymbol{Y}, \boldsymbol{y}) \rightarrow (\boldsymbol{A_j x} \geq \boldsymbol{b_j}),\ j \in \mathcal{J} \ \Big|\ \Omega_i^q(\boldsymbol{Y}, \boldsymbol{y}),\ i \in \mathcal{I}\ . \end{aligned} \tag{2.29}$$

Equation (2.29) splits the constraints into continuous and logical parts, on the left and right of the bar, respectively. The logical part consists of formulas $\Omega_i^q(\boldsymbol{Y}, \boldsymbol{y})$ where $\boldsymbol{Y} \in \{\textit{True, False}\}^{n_B}$ and $\boldsymbol{y} \in \mathbb{Z}^{n_I}$. The continuous part is formulated as logical implications such that if $\Omega_j^p(\boldsymbol{Y}, \boldsymbol{y})$ is true then the constraint $\boldsymbol{A_j x} \geq \boldsymbol{b_j}$ is imposed.

MLLP models are solved by branching on the propositional variables $\boldsymbol{Y}$ and discrete variables $\boldsymbol{y}$. As branching takes place, MLLP progressively strengthens the relaxation by enforcing constraints $\boldsymbol{A_j x} \geq \boldsymbol{a_j}$ if the corresponding antecedent Ω_j^p is true. Since the logical part is separated from the continuous part, MLLP enables propositional satisfiability algorithms to derive further logical constraints and prune the search space.

An MLLP model may look different from the equivalent MILP. For many cases, e.g. where MILP binary variables model existence or assignment, MLLP may result in an easier-to-comprehend model with fewer variables. MLLP may be extended to models with nonlinear constraints, i.e. to mixed logical-nonlinear programming (Türkay and Grossmann, 1996; Bollapragada et al., 2001; Bemporad and Giorgetti, 2004, 2006; Carbonneau et al., 2011, 2012).

2.3.5 Optimisation Methods based on Satisfiability Modulo Theories

The SMT language enables a user to easily model applications with logical and continuous constraints. Indeed, such models are also GDP (Grossmann and Ruiz, 2012) representable. However, GDP formulations are more restrictive when modelling interactions between continuous and propositional variables since these interactions can only occur in disjunctive constraints. Therefore, a GDP equivalent of an SMT formulation may be less expressive, e.g. when modelling implications. SMT solvers support logical theories and dependencies, do precise arithmetic, and enable incremental solving. But SMT solvers may have performance issues with division, reasoning over integers, and only limited support for transcendental functions (de Moura and Passmore, 2013). MINLP tools support the transcendental functions and scale well for mixed-integer reasoning (Carvajal et al., 2014), but MINLP solvers cannot solve incrementally and have limited support for logical constraints and are sensitive to rounding errors.

Two of the most prominent SMT-based optimisation methods are optimisation modulo theories and integer linear programming modulo theories.

Optimisation modulo theories integrates optimisation and SMT with respect to the theory of linear arithmetic over the rationals (Sebastiani and Tomasi, 2015). Sebastiani and Tomasi (2015) consider different approaches to solve the MILP problems, e.g. offline and inline schemas with linear, binary or adaptive search. Sebastiani and Tomasi (2015) compare optimisation modulo theories versus linear GDP using both a convex hull and a big-M relaxation. The comparisons, based on strip packing and job shop scheduling case studies (Sawaya and Grossmann, 2005), show that an SMT solver may be used for optimisation.

Integer linear programming modulo theories is an optimisation framework where MILP, rather than SAT, is leveraged as the efficient solver (Manolios and Papavasileiou, 2013). Integer linear programming modulo theories is an optimisation framework in which difference logic is used to communicate with the solver. Manolios and Papavasileiou (2013) implement their framework as a constraint handler for the MILP solver SCIP (Achterberg, 2007b, 2009). A

weakness of an MILP-based approach is that floating point calculations may lead to wrong answers. Errors based on floating point do not happen in SMT because all formulae evaluate to true or false only.

2.3.6 Logic-Based Benders Decomposition

Hybrid optimisation/logic approaches have been developed combining mixed-integer linear programming (MILP) and constraint programming (CP), e.g. Jain and Grossmann (2001); Maravelias and Grossmann (2004); Li and Womer (2008); Sitek (2014), or multiple levels of MILP, e.g. Maravelias (2006). The hybrid formulations usually use logic-based Benders decomposition (LBBD) (Hooker and Ottoson, 2003), a generalisation of Benders decomposition (Benders, 1962). The principles of Benders decomposition remain: we have a master problem and a subproblem which generates cuts if the solution from the master problem is infeasible. The difference is that LBBD requires a logic proof deriving an objective bound. Other hybrid algorithms use branch-and-check (Thorsteinsson, 2001) or Lagrangian decomposition (Papageorgiou and Trespalacios, 2018).

Hybrid MILP/CP methods are typically applied to scheduling and its variants (Sitek, 2014). This is reasonable: CP is very good at assessing scheduling feasibility. The problem with hybrid MILP/CP is that, if the application does not have a suitable CP constraint, a hybrid method may be poor since bespoke CP constraints take full advantage of very specific mathematical structures.

2.4 Metaheuristics

2.4.1 Particle Swarm Optimisation

Kennedy and Eberhart (1995) introduce PSO for optimising continuous nonlinear functions. PSO computes a good heuristic solution by triggering m particles that collaboratively search the

Algorithm 3 Particle Swarm Optimisation

Compute initial position $\boldsymbol{x}_i^{(0)} \in \mathbb{R}^n$ and velocity $\boldsymbol{v}_i^{(0)} \in \mathbb{R}^n$ for each particle $i = 1, \ldots, m$.
$\boldsymbol{p}_i \leftarrow \boldsymbol{x}_i^{(0)}$
$\boldsymbol{g} \leftarrow \arg\min\{f(\boldsymbol{p}_i)\}$
$k \leftarrow 0$
while the time limit is not exceeded **do**
  for $i = 1, \ldots, m$ **do**
    Choose random values $r_1, r_2 \sim U(0, 1)$
    $\boldsymbol{v}_i^{(k+1)} \leftarrow \omega \boldsymbol{v}_i^{(k)} + c_1 \cdot r_1 \cdot (\boldsymbol{p}_i - \boldsymbol{x}_i^{(k)}) + c_2 \cdot r_2 \cdot (\boldsymbol{g} - \boldsymbol{x}_i^{(k)})$
    $\boldsymbol{x}_i^{(k+1)} \leftarrow \boldsymbol{x}_i^{(k)} + \boldsymbol{v}_i^{(k+1)}$
    if $f(\boldsymbol{x}_i^{(k+1)}) < f(\boldsymbol{p}_i)$ **then**
      $\boldsymbol{p}_i \leftarrow \boldsymbol{x}_i^{(k+1)}$
    end if
  end for
  $\boldsymbol{g} \leftarrow \arg\min\{f(\boldsymbol{p}_i)\}$
  $k \leftarrow k + 1$
end while

feasibility space. PSO picks the initial particle position $\boldsymbol{x}_i^{(0)}$ and search direction $\boldsymbol{v}_i^{(0)}$ of particle i randomly. The search occurs in a sequence of rounds. In round k, every particle chooses its next position $\boldsymbol{x}_i^{(k+1)}$ by following the direction specified by a weighted sum of: (i) the current trajectory direction $\boldsymbol{v}_i^{(k)}$, (ii) the particle's best found solution $\boldsymbol{p}_i$, (iii) the globally best found solution $\boldsymbol{g}$, and moving by a fixed step size. The *inertia term* $\omega \boldsymbol{v}_i^{(k)}$ controls how quickly a particle changes direction. The *cognitive term* $c_1 \cdot r_1 \cdot (\boldsymbol{p}_i - \boldsymbol{x}_i^{(k)})$ controls the particle tendency to move to the best observed solution by that particle. The *social term* $c_2 \cdot r_2 \cdot (\boldsymbol{g} - \boldsymbol{x}_i^{(k)})$ controls the particle tendency to move toward the best solution observed by any particle. Coefficients ω, c_1, and c_2 are tunable parameters. Termination occurs either when all particles are close, or within a specified time limit. Algorithm 3 lists the PSO algorithm.

2.4.2 Simulated Annealing

Simulated annealing (SA) is a metaheuristic method for producing a near-optimal solution to an optimisation problem (Metropolis et al., 1953). SA is inspired by a well-known analogy between the physical annealing process of metals commonly employed in metallurgy. Above their melting point, metals are in liquid states and their atoms are randomly arranged, relatively

free to move. By cooling, i.e. annealing, a metal transitions into lower energy states and settles into a solid state with ordered crystalline structures. At temperature T, the probability of an energy increase of magnitude ΔE is $\exp(-\Delta E/kT)$, where k is the Boltzmann's constant. By very slow cooling, the material settles into a low energy state and the material properties are improved.

If we interpret (i) material states as feasible solutions, (ii) state changes as transitions to neighboring solutions, (iii) energy as objective function cost, and (iv) temperature as control parameter, we may use SA as a metaheuristic approach for solving combinatorial optimisation problems. SA starts with an initial solution $\boldsymbol{x}^{(0)}$, temperature $T^{(0)}$, and iteratively moves to new solutions $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots$, until it ends up with its last, good heuristic solution $\boldsymbol{x}^{(\ell)}$ while the temperature is dropped. During this process, transitions to certain solutions may be rejected. As opposed to local search that only accepts transitions to solutions with strictly better objective values, i.e. $f(\boldsymbol{x}^{(k-1)}) > f(\boldsymbol{x}^{(k)})\ \forall k \in \{1, \ldots, \ell\}$, SA may probabilistically accept transitions to worse solutions with the aim of escaping local minima. Initially, the temperature is high enough so that all transitions are allowable and SA performs a sort of random search in the feasibility space. As the algorithm progresses and the temperature decreases, the tendency to move to worse solutions decreases.

Algorithm 4 lists the simulated annealing algorithm (Kirkpatrick et al., 1983). The initial temperature is typically set $T^{(0)} = 1$. We may naively set the probability constant $c = 1$, or select c to normalise the objective function in a way that all transitions are accepted in the starting temperature $T^{(0)} = 1$. Temperature is gradually decreased according to a fixed rule such that $\lim_{t\to\infty} T^{(t)} = 0$. Typically, the rule $T^{(t)} = \alpha T^{(t-1)}$ is adopted, where $\alpha \in [0.8, 0.99]$. At a given temperature level $T^{(t)}$, a number $r \in [100, 1000]$ of iterations (repetitions) is performed before a temperature decrease. In each iteration, we investigate a randomly chosen solution $\boldsymbol{x}$ in the neighborhood $\mathcal{N}(\boldsymbol{x}^{(k)})$ of the current solution $\boldsymbol{x}^{(k)}$. If the new solution improves, i.e. $f(\boldsymbol{x}) < f(\boldsymbol{x}^{(k)})$, the algorithm moves to the new solution. If the new solution is of worse objective value than the current one, then it is accepted with probability $\exp(-(f(\boldsymbol{x}) - f(\boldsymbol{x}^{(k)}))/cT^{(t)})$. The algorithm terminates when the temperature is sufficiently small, i.e. when $|T^{(t)}| \leq \varepsilon$, or equivalently when a certain number of temperature decreases has been performed.

Algorithm 4 Simulated Annealing

Compute an initial solution $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$.
Set initial temperature $T^{(0)} = 1$ and probability constant $c = 1$.
Set temperature factor $\alpha \in [0.80, 0.99]$.
$t = 0$, $k = 0$
while $T^{(t)} > \varepsilon$ **do**
    for r iterations **do**
        Select a neighboring solution $\boldsymbol{x} \in \mathcal{N}(\boldsymbol{x}^{(k)})$ randomly.
        if $f(\boldsymbol{x}) < f(\boldsymbol{x}^{(k)})$ **then**
            $\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}$
            $k \leftarrow k + 1$
        else
            Choose $p \sim U(0, 1)$
            if $\exp(-(f(\boldsymbol{x}) - f(\boldsymbol{x}^{(k)}))/cT^{(t)}) > p$ **then**
                $\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}$
                $k \leftarrow k + 1$
            end if
        end if
    end for
    $T^{(t+1)} \leftarrow \alpha T^{(t)}$
    $t \leftarrow t + 1$
end while

Chapter 3

Application Background - Gradient-Boosted Trees

Consider integrating an unknown function into an optimisation problem, i.e. without a closed-form formula, but with a data set representing evaluations over a box-constrained feasibility domain. Optimisation in the machine learning literature usually refers to the training procedure, e.g. model accuracy maximisation (Sra et al., 2012; Snoek et al., 2012). This chapter considers optimising after the training procedure, where the trained predictive model is embedded in the optimisation problem. We consider optimisation methods for problems with gradient-boosted tree (GBT) models embedded (Friedman, 2001; Hastie et al., 2009). Advantages of GBTs are myriad (Chen and Guestrin, 2016; Ke et al., 2017), e.g. they are robust to scale differences in the training data features, handle both categorical and numerical variables, and can minimise arbitrary, differentiable loss functions.

We consider an optimisation problem that incorporates an additional convex penalty term in the objective, accounting for risky predicted values in parts of the feasibility domain where the machine learning model is not well trained due to missing data. However, penalising a greater distance from the candidate solution to the existing data may not be the only reason to add a convex penalty function, e.g. our numerical tests consider an instance with an additional soft constraint. Duran and Grossmann (1986a) document several other convex terms commonly

appearing in process engineering, any of those convex nonlinear equations could be incorporated into this framework. Another possible application area for this work is in portfolio optimisation, e.g. as an extension to the Markowitz model with cardinality constraint and buy-in threshold constraints (Bienstock, 1996). Several authors have considered more elaborate extensions, e.g. by integrating uncertainty in the expected return estimate (Bonami and Lejeune, 2009) or considering concave transaction costs (Konno and Wijayanayake, 2001). But the framework we present here could use GBT models to develop data-driven uncertainty or cost models. Meanwhile, the well known Markowitz model is convex.

Related Work Lombardi and Milano (2018) survey approaches for embedding machine learning models as parts of decision-making problems. We *encode the ML model using the native language* (Lombardi and Milano, 2018), i.e. in an optimisation modelling framework, namely, mathematical programming. Resulting optimisation models may be addressed using local (Nocedal and Wright, 2006) or deterministic global (Schweidtmann and Mitsos, 2019) methods. The value of global optimisation is known in engineering (Boukouvala et al., 2016), e.g. local minima can lead to infeasible parameter estimation (Singer et al., 2006) or misinterpreted data (Bollas et al., 2009). For applications where global optimisation is less relevant, we still wish to develop optimisation methods for discrete and non-smooth machine learning models, e.g. regression trees. Discrete optimisation methods allow repurposing a legacy model, originally built for prediction, into an optimisation framework. In a closely related line of work, Donti et al. (2017) investigate training machine learning models in a manner that captures the task for which they will be used. Here, we focus on generating optimal decisions once the ML model has been trained.

This chapter formulates a problem that is closely related to Mišić (2017), the contents of this chapter and Chapter 4 differs from Mišić (2017) as follows.

1. Mišić (2017) studies the problem of optimising over a tree ensemble whereas our problem formulation includes a convex penalty in the objective as well.

2. Both Mišić (2017) and Chapter 4 develop approaches for relaxing the tree ensemble

optimisation problem to derive rigorous bounds. Mišić (2017) proposes a hierarchy of relaxations that approximate each tree in the ensemble up to a certain depth. Furthermore, Mišić (2017) quantifies an approximation guarantee on their relaxation. Instead of reducing the instance size with respect to tree depth, our tree ensemble relaxation partitions the ensemble into smaller tree ensembles. The bound is calculated by considering these smaller instances independently. We show that a partial ordering over the partitions guarantees improvement in the relaxation tightness.

3. In terms of solution methodologies, Mišić (2017) develops a Benders decomposition (Benders, 1962) strategy and a split generation approach that lazily reintroduces split nodes that can be used as part of an iterative method that tightens a depth d relaxation. Our work develops a B&B strategy and heuristics for the problem that we consider. In particular, we develop a pseudocost initialisation strategy, a strong branching approach and an iterative mixed-integer convex programming-based heuristic. The pseudocost initialisation strategy may be used to guide branch selection for the optimisation problem that Mišić (2017) considers as well.

Chapter Organisation Section 3.1 introduces GBTs, Section 3.2 introduces the optimisation problem and Section 3.3 formulates the optimisation problem as a convex MINLP.

3.1 Gradient-Boosted Trees

This section describes gradient-boosted trees (GBTs) (Friedman, 2001, 2002). In this work, GBTs are embedded into the Section 3.2 optimisation problem. GBTs are a subclass of boosting methods (Freund, 1995). Boosting methods train many weak learners iteratively that collectively produce a strong learner, where a weak learner is at least better than random guessing. Each boosting iteration trains a new weak learner against the residual of the previously trained learners by minimising a loss function. For GBTs, the weak learners are classification and regression trees (Breiman et al., 1984).

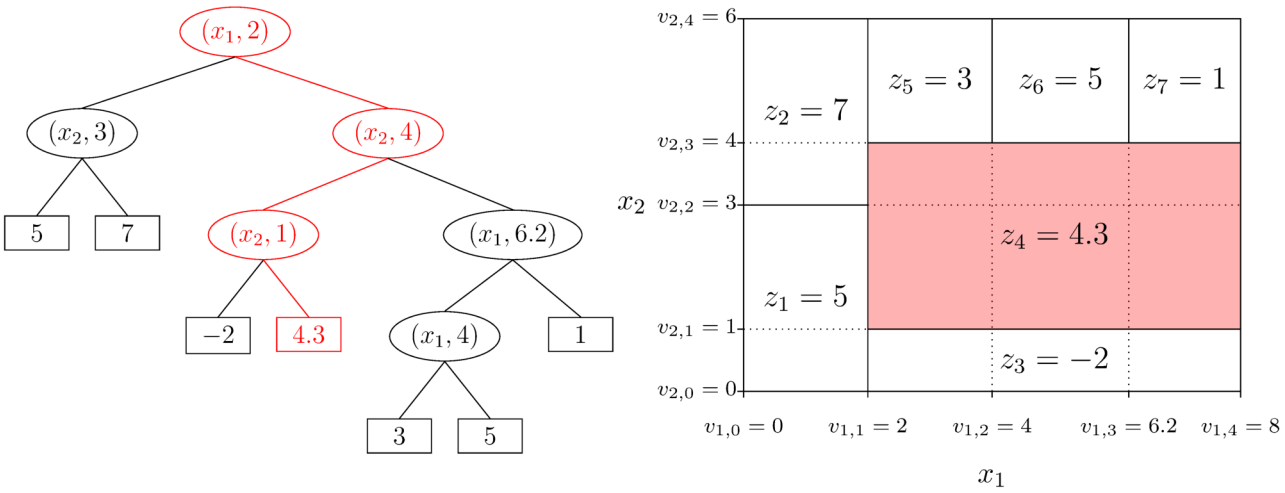


Figure 3.1: Gradient boosted tree, see Definition 3.1, trained in two dimensions. Left: gradient boosted tree. Right: recursive domain partition defined by tree on left. The highlighted path and region corresponds to the result of evaluating at $\boldsymbol{x} = (4.2, 2.8)^\top$ as in Example (1).

This chapter restricts its analysis to regression GBTs without categorical input variables. A trained GBT function is a collection of binary trees and each of these trees provides its own independent contribution when evaluating at $\boldsymbol{x}$.

Definition 3.1. *A trained GBT function is defined by sets* $(\mathcal{T}, \mathcal{V}_t, \mathcal{L}_t)$ *and values* $(i(t,s), v(t,s), F_{t,l})$. *The set* $\mathcal{T}$ *indexes the trees. For a given tree* $t \in \mathcal{T}$, $\mathcal{V}_t$ *and* $\mathcal{L}_t$ *index the split and leaf nodes, respectively. At split node* $t \in \mathcal{T}$, $s \in \mathcal{V}_t$, $i(t,s)$ *and* $v(t,s)$ *return the split variable and value, respectively. At leaf node* $t \in \mathcal{T}$, $l \in \mathcal{L}_t$, $F_{t,l}$ *is its contribution.*

Tree $t \in \mathcal{T}$ evaluates at $\boldsymbol{x}$ by following a root-to-leaf path. Beginning at the root node of t, each encountered split node $s \in \mathcal{V}_t$ assesses whether $x_{i(t,s)} < v(t,s)$ or $x_{i(t,s)} \geq v(t,s)$ and follows the left or right child, respectively. The leaf $l \in \mathcal{L}_t$ corresponding to $\boldsymbol{x}$ returns t's contribution $F_{t,l}$. Figure 3.1 shows how a single gradient-boosted tree recursively partitions the domain. The overall output, illustrated in Figure 3.2, sums all individual tree evaluations:

$$\mathrm{GBT}(\boldsymbol{x}) = \sum_{t \in \mathcal{T}} \mathrm{GBT}_t(\boldsymbol{x}).$$

Example 3.1. *Consider a trained GBT that approximates a two-dimensional function with* $\mathcal{T} = \{t_1, \ldots, t_{|\mathcal{T}|}\}$. *To evaluate* $\mathrm{GBT}(\boldsymbol{x})$ *where* $\boldsymbol{x} = (4.2, 2.8)^\top$, *let* t_1 *be the tree given by Figure 3.1, the highlighted path corresponds to evaluating at* $\boldsymbol{x}$. *The root split node query*

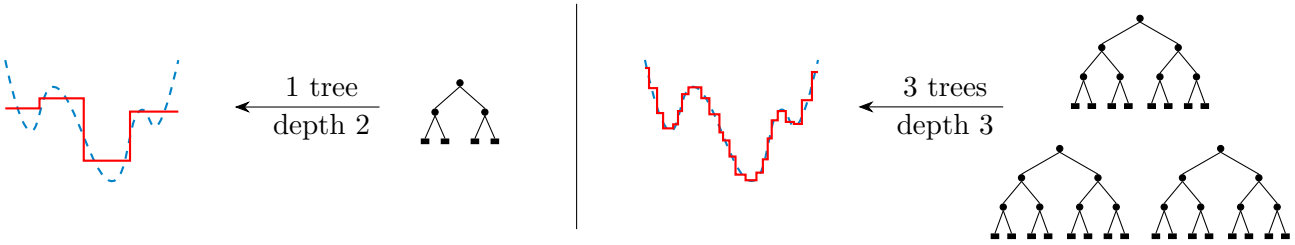


Figure 3.2: GBT approximations to the dashed function: 1 tree of depth 2 (left) and 3 trees of depth 3 (right).

Table 3.1: Mixed-integer convex programming model sets, parameters and variables.

Symbol	Description
v_i^L, v_i^U	Lower and upper bound of variable x_i
x_i	Continuous variable, $i \in \{1, \ldots, n_C\}$
$t \in \mathcal{T}$	Indices of GBTs
$l \in \mathcal{L}_t$	Indices of leaves for tree t
$s \in \mathcal{V}_t$	Indices of split nodes for tree t
m_i	Number of variable x_i splitting values
$v_{i,j}$	Variable i's j-th breakpoint, $j \in \{1, \ldots, m_i\}$
$F_{t,l}$	Value of leaf (t, l)
$y_{i,j}$	Binary variable indicating whether variable $x_i < v_{i,j}$
$z_{t,l}$	Nonnegative variable that activates leaf (t, l)

of $x_1 < 2$ *is false, since* $x_1 = 4.2$*, so we follow the right branch. Following this branch encounters another split node. The next query of* $x_2 < 4$ *is true, since* $x_2 = 2.8$*, so we follow the left branch. The final branch reaches a leaf with value 4.3, hence* $\mathrm{GBT}_{t_1}(\boldsymbol{x}) = 4.3$*. The remaining trees also return a value after making similar queries on* $\boldsymbol{x}$*. This results in* $\mathrm{GBT}(\boldsymbol{x}) = \sum_{i=1}^{|\mathcal{T}|} \mathrm{GBT}_{t_i}(\boldsymbol{x}) = 4.3 + \sum_{i=2}^{|\mathcal{T}|} \mathrm{GBT}_{t_i}(\boldsymbol{x})$.

3.2 Optimisation Problem

This chapter considers an optimisation problem that includes a GBT-trained function in the objective. The choice of using GBTs for the machine-learning task leverages two strategies: boosting and decision trees. Boosting (Freund, 1995), in the context of GBTs, trains many relatively low depth decision trees in sequence where consecutive trees are trained against the residual of the previously trained trees. Decision trees (Breiman et al., 1984) partition the domain and assign each disjoint region a score. This partitioning of the domain, allows

each tree to consider the disjoint regions in isolation, i.e. the scores assigned to the disjoint regions do not influence each other. This disjointedness allows the decision trees to capture sudden changes, e.g. nonconvex behaviour, in the underlying training data without fitting a continuous function. However, while GBT-trained functions may result in good predictors, we may want to optimise over the value being predicted. For example, the Chapter 4 numerical tests consider a GBT-trained function that predicts the compressive strength of concrete. Using this GBT-trained function for prediction questions: *given input* $\boldsymbol{x}$*, what is the compressive strength?* Whereas in the optimisation context, the question becomes: *what input* $\boldsymbol{x}$ *predicts the maximal compressive strength?* This move from prediction to optimisation motivates box-constrained optimisation Problem (3.1), where the objective is the sum of a convex nonlinear function and a GBT-trained function:

$$\min_{\boldsymbol{v}^L \leq \boldsymbol{x} \leq \boldsymbol{v}^U} \underbrace{\text{cvx}(\boldsymbol{x})}_{\textit{Convex Part}} + \underbrace{\text{GBT}(\boldsymbol{x})}_{\textit{GBT Part}}, \tag{3.1}$$

where $\boldsymbol{x} = (x_1, \ldots, x_{n_C})^\top$ is the variable vector. $\text{GBT}(\boldsymbol{x})$ is the GBT-trained function value at $\boldsymbol{x}$. Table 3.1 defines the model sets, parameters and variables. Problem (3.1) is relevant, for example, in cases where a GBT function has been trained to data but we may trust an optimal solution close to regions with many training points. A convex penalty term may penalise solutions further from training data. For instance, consider historical data from a manufacturing process for quality maximisation. The data may exhibit correlation between two process parameters, e.g. the temperature and the concentration of a chemical additive. A machine learned model of the system assigns weights to these parameters for future predictions. Lacking additional information, numerical optimisation may produce candidate solutions with temperature and concentration combinations that (possibly incorrectly) suggest temperature is responsible for an observed effect. The convex penalty term helps control the optimiser's adventurousness by penalising deviation from the training data subspace and is parameterised using principal component analysis (Vaswani et al., 2018). Large values of this risk control term generate conservative solutions. Smaller penalty values explore regions with greater possible rewards but also additional risk. Further to modelling distance to training data, the convex penalty may be used to characterise additional soft constraints.

A given problem instance may sum independently-trained GBT functions. Without loss of generality, we equivalently optimise a single GBT function which is the union of all original GBTs.

3.3 Mixed-Integer Convex Formulation

Problem (3.1) consists of a continuous convex function and a discrete GBT function. The discrete nature of the GBT function arises from the left/right decisions at the split nodes. So we consider a mixed-integer nonlinear program with convex nonlinearities (convex MINLP) formulation. The main ingredient of the convex MINLP model is a mixed-integer linear programming (MILP) formulation of the GBT part which merges with the convex part via a linking constraint. The high level convex MINLP is:

$$\min_{\boldsymbol{v}^L \leq \boldsymbol{x} \leq \boldsymbol{v}^U} \quad \text{cvx}(\boldsymbol{x}) + [\text{GBT MILP objective}] \tag{3.2a}$$

$$\text{s.t.} \quad [\text{GBT MILP constraints}], \tag{3.2b}$$

$$[\text{Variable linking constraints}]. \tag{3.2c}$$

3.3.1 Gradient-Boosted Trees Mixed-Integer Linear Programming Formulation

We form the GBT MILP using the Mišić (2017) approach, which recalls the state-of-the-art in modelling piecewise linear functions (Misener et al., 2009; Misener and Floudas, 2010; Vielma et al., 2010). Alternative modelling approaches include the Verwer et al. (2017) MILP formulation, the Bonfietti et al. (2015) constraint programming formulation, and the Lombardi et al. (2017) satisfiability modulo theories formulation.

Figure 3.1 shows how a GBT partitions the domain $[\boldsymbol{v}^L, \boldsymbol{v}^U]$ of $\boldsymbol{x}$. Optimising a GBT function

reduces to optimising the leaf selection, i.e. finding an optimal interval, opposed to a specific $\boldsymbol{x}$ value. Aggregating over all GBT split nodes produces a vector of ordered breakpoints $v_{i,j}$ for each x_i variable: $v_i^L = v_{i,0} < v_{i,1} < \cdots < v_{i,m_i} < v_{i,m_i+1} = v_i^U$. Selecting a consecutive pair of breakpoints for each x_i defines an interval where the GBT function is constant. Each point $x_i \in [v_i^L, v_i^U]$ is either on a breakpoint $v_{i,j}$ or in the interior of an interval. Binary variable $y_{i,j}$ models whether $x_i < v_{i,j}$ for $i \in [n_C] = \{1, \ldots, n\}$ and $j \in [m_i] = \{1, \ldots, m_i\}$. Binary variable $z_{t,l}$ is 1 if tree $t \in \mathcal{T}$ evaluates at node $l \in \mathcal{L}_t$ and 0 otherwise. Denote by $\mathcal{V}_t$ the set of split nodes for tree t. Moreover, let $\text{Left}_{t,s}$ and $\text{Right}_{t,s}$ be the sets of subtree leaf nodes rooted in the left and right children of split node s, respectively.

MILP Problem (3.3) formulates the GBT (Mišić, 2017). Equation (3.3a) minimises the total value of the active leaves. Equation (3.3b) selects exactly one leaf per tree. Equations (3.3c) and (3.3d) activates a leaf only if all corresponding splits occur. Equation (3.3e) ensures that if $x_i \leq v_{i,j-1}$, then $x_i \leq v_{i,j}$. Without loss of generality, we drop the $z_{t,l}$ integrality constraint because any feasible assignment of $\boldsymbol{y}$ specifies one leaf, i.e. a single region in Figure 3.1.

$$\min \sum_{t \in \mathcal{T}} \sum_{l \in \mathcal{L}_t} F_{t,l} z_{t,l} \tag{3.3a}$$

$$\text{s.t.} \sum_{l \in \mathcal{L}_t} z_{t,l} = 1, \qquad \forall t \in \mathcal{T}, \tag{3.3b}$$

$$\sum_{l \in \text{Left}_{t,s}} z_{t,l} \leq y_{i(s),j(s)}, \qquad \forall t \in \mathcal{T}, s \in \mathcal{V}_t, \tag{3.3c}$$

$$\sum_{l \in \text{Right}_{t,s}} z_{t,l} \leq 1 - y_{i(s),j(s)}, \qquad \forall t \in \mathcal{T}, s \in \mathcal{V}_t, \tag{3.3d}$$

$$y_{i,j} \leq y_{i,j+1}, \qquad \forall i \in [n_C],\ j \in [m_i - 1], \tag{3.3e}$$

$$y_{i,j} \in \{0, 1\}, \qquad \forall i \in [n_C],\ j \in [m_i], \tag{3.3f}$$

$$z_{t,l} \geq 0, \qquad \forall t \in \mathcal{T},\ l \in \mathcal{L}_t. \tag{3.3g}$$

Equation (3.3g) models the domain of leaf selection variables $z_{t,l}$ as $\mathbb{R}_{\geq 0}$ opposed to $\{0, 1\}$. We show correctness by considering a feasible assignment $\hat{\boldsymbol{y}}$ on binary variables $\boldsymbol{y}$. Let $t \in \mathcal{T}$ and let $s_t^{\text{root}} \in \mathcal{V}_t$ be the root split node of t. If $\hat{y}_{i(s_t^{\text{root}}),j(s_t^{\text{root}})} = 0$ then Equation (3.3c) sets $z_{t,l} = 0$, $\forall l \in \text{Left}_{t,s_t^{\text{root}}}$, otherwise Equation (3.3d) sets $z_{t,l} = 0$, $\forall l \in \text{Right}_{t,s_t^{\text{root}}}$. These two

cases correspond to following the right and left branch of the tree t root split node, respectively. Repeating this reasoning for non-root nodes of tree t results in $z_{t,l} = 0 \; \forall l \in \mathcal{L}_t \setminus \{l^*\}$ for some $l^* \in \mathcal{L}_t$. Leaf l^* corresponds to the leaf that $\hat{\boldsymbol{y}}$ selects, see the Figure 3.1 recursive partition interpretation which indicates why a feasible assignment on $\boldsymbol{y}$ always corresponds to a unique leaf. Since, only z_{t,l^*} is unassigned, Equation (3.3b) sets $z_{t,l^*} = 1$. Hence, the domain of $z_{t,l}$ can be made continuous. The $z_{t,l}$ have to be non-negative since Equations (3.3c) and (3.3d) only enforce upper bounds.

3.3.2 Linking Constraints

We complete the mixed-integer representation of Problem (3.1) by linking continuous variables $\boldsymbol{x}$ with the Problem (3.3) binary variables. Consider feasible assignment $\hat{\boldsymbol{y}}$ to binary variables $\boldsymbol{y}$ in Problem (3.3) and let $\hat{\boldsymbol{y}}_{\boldsymbol{i}} = (\hat{y}_{i,1}, \ldots, \hat{y}_{i,m_i})^\top$. Equation (3.3e) orders $\hat{\boldsymbol{y}}_{\boldsymbol{i}}$ such that any feasible assignment is a sequence of zeroes followed by a sequence of ones. Let $\hat{j} = \max(\{j \in [m_i] | \hat{y}_{i,j} = 0\} \cup \{0\})$, i.e. $\hat{j}$ is either equal to the index of the last zero in $\hat{\boldsymbol{y}}_{\boldsymbol{i}}$ or equal to zero if no such index exists ($\hat{\boldsymbol{y}}_{\boldsymbol{i}}$ can be the vector of all ones). Since $y_{i,j} = 0$ corresponds to $x_i \geq v_{i,j}$ and $\hat{\boldsymbol{y}}_{\boldsymbol{i}}$ begins with a sequence of zeroes, we have that $\hat{\boldsymbol{y}}_{\boldsymbol{i}}$ corresponds to $x_i \geq v_{i,\hat{j}}$. We model the sequential lower bounding behaviour exhibited by $\boldsymbol{y}_{\boldsymbol{i}}$ with:

$$x_i \geq v_{i,0} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j-1})(1 - y_{i,j}).$$

For $\boldsymbol{y} = \hat{\boldsymbol{y}}$, the constraint becomes $x_i \geq v_{i,0} + \sum_{j=1}^{\hat{j}} v_{i,j} - v_{i,j-1} = v_{i,\hat{j}}$, since consecutive terms in the sum have elements that cancel. For the particular case of $\hat{j} = 0$, we have that $x_i \geq v_{i,0}$ is enforced, i.e. the global lower bound on x_i. We derive the upper bound on x_i that $\hat{\boldsymbol{y}}_{\boldsymbol{i}}$ corresponds to, i.e. $v_{i,\hat{j}+1}$, with a similar argument. This gives Equations (3.4a) and (3.4b) which link the continuous x_i variables, from the original Problem (3.1) definition, to the binary $y_{i,j}$ variables:

$$x_i \geq v_{i,0} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j-1})(1 - y_{i,j}), \tag{3.4a}$$

$$x_i \leq v_{i,m_i+1} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j+1}) y_{i,j}, \tag{3.4b}$$

for all $i \in [n_C]$. We express the linking constraints using non-strict inequalities to avoid computational issues when optimising with strict inequalities. Combining Equations (3.2) to (3.4) defines the mixed-integer nonlinear program with convex nonlinearities (convex MINLP) formulation to Problem (3.1), i.e. Problem (3.5).

$$\min_{\boldsymbol{v}^L \leq \boldsymbol{x} \leq \boldsymbol{v}^U} \ \mathrm{cvx}(\boldsymbol{x}) + \sum_{t \in \mathcal{T}} \sum_{l \in \mathcal{L}_t} F_{t,l} z_{t,l} \tag{3.5a}$$

$$\text{s.t.} \sum_{l \in \mathcal{L}_t} z_{t,l} = 1, \qquad \forall t \in \mathcal{T}, \tag{3.5b}$$

$$\sum_{l \in \mathrm{Left}_{t,s}} z_{t,l} \leq y_{i(s),j(s)}, \qquad \forall t \in \mathcal{T}, s \in \mathcal{V}_t, \tag{3.5c}$$

$$\sum_{l \in \mathrm{Right}_{t,s}} z_{t,l} \leq 1 - y_{i(s),j(s)}, \qquad \forall t \in \mathcal{T}, s \in \mathcal{V}_t, \tag{3.5d}$$

$$y_{i,j} \leq y_{i,j+1}, \qquad \forall i \in [n_C],\ j \in [m_i - 1], \tag{3.5e}$$

$$x_i \geq v_{i,0} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j-1})(1 - y_{i,j}), \qquad \forall i \in [n_C], \tag{3.5f}$$

$$x_i \leq v_{i,m_i+1} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j+1}) y_{i,j}, \qquad \forall i \in [n_C], \tag{3.5g}$$

$$y_{i,j} \in \{0, 1\}, \qquad \forall i \in [n_C],\ j \in [m_i], \tag{3.5h}$$

$$z_{t,l} \geq 0, \qquad \forall t \in \mathcal{T},\ l \in \mathcal{L}_t. \tag{3.5i}$$

When $\mathrm{cvx}(\boldsymbol{x})$ is linear, Problem (3.5) belongs to the MILP problem class and may be solved by commercial MILP solvers, e.g. $\mathrm{cvx}(\boldsymbol{x})$ could involve the L_1-norm. In general, Problem (3.5) may be solved using a convex MINLP solver, e.g. Westerlund and Pettersson (1995); Tawarmalani and Sahinidis (2005); Vigerske (2012); Misener and Floudas (2014); Lundell et al. (2017). Chapter 4 assesses this approach numerically for our tested instances.

Considering the structure of Problem (3.5), we have the majority of the constraints come from modelling the GBT aspect, i.e. feasibility in Problem (3.5) is more influenced by the heavily constrained mixed-integer aspect opposed to the convex aspect. Hence, it a solver may still expend a fair amount of effort in finding integer feasible assignments for the binary $y_{i,j}$ variables. The constraints of Problem (3.5) extends the Problem (3.3) constraints with Equation (3.4). Mišić (2017) discusses relaxation tightness of Problem (3.3) and shows that it is tighter than an

alternative formulation. The relaxation of Problem (3.5) also includes Equation (3.4a). We may rewrite this constraint as

$$x_i \geq v_{i,0}y_{i,1} + \sum_{j=1}^{m_i-1} v_{i,j}(y_{i,j+1} - y_{i,j}) + v_{i,m_i}(1 - y_{i,m_i}), \quad \forall i \in [n_C], \tag{3.6}$$

i.e. Equation (3.6) expands factors in Equation (3.4a) and regroups the elements by common $v_{i,j}$'s. By $y_{i,j} \in [0,1]$ and $y_{i,j} \leq y_{i,j+1}$, we have that

$$y_{i,1} + \sum_{j=1}^{m_i-1} (y_{i,j+1} - y_{i,j}) + (1 - y_{i,m_i}) = 1, \quad \forall i \in [n_C].$$

Hence, Equation (3.6) corresponds to a convex combination of the lower interval endpoints. Furthermore, in an integer feasible assignment to $y_{i,j}$, $y_{i,1} = 1$ corresponds to selecting $x_i \in [v_{i,0}, v_{i,1}]$, $(y_{i,j+1} - y_{i,j}) = 1$ corresponds to selecting $x_i \in [v_{i,j}, v_{i,j+1}]$, and $(1 - y_{i,m_i}) = 1$ corresponds to selecting $x_i \in [v_{i,m_i}, v_{i,m_i+1}]$, i.e. the disjunctive choice between intervals. As Equation (3.4a) is a convex combination over the lower limits, the constraint is at least as tight as a big-M approach, such as:

$$\begin{aligned}
&x_i \geq v_{i,0}, && \forall i \in [n_C] \\
&x_i \geq v_{i,j} - (v_{i,j} - v_{i,0})(1 - (y_{i,j+1} - y_{i,j})), && \forall i \in [n_C], j \in [m_i - 1] \\
&x_i \geq v_{i,m_i} - (v_{i,m_i} - v_{i,0})y_{i,m_i}, && \forall i \in [n_C].
\end{aligned}$$

A similar analysis holds for Equation (3.4b).

3.3.3 Worst Case Analysis

The difficulty of Problem (3.1) is primarily justified by the fact that optimising a GBT-trained function, i.e. Problem (3.3), is an $\mathcal{NP}$-hard problem (Mišić, 2017). This section shows that the number of continuous variable splits and tree depth affect complete enumeration methods. These parameters motivate the branching scheme in our branch-and-bound algorithm.

In a GBT ensemble, each continuous variable x_i is associated with $m_i + 1$ intervals (splits).

Picking one interval $j \in \{1, \ldots, m_i + 1\}$ for each x_i sums to a total of $\prod_{i=1}^{n}(m_i + 1)$ distinct combinations. A GBT-trained function evaluation selects a leaf from each tree. However, not all leaf combinations are valid evaluations. In a feasible leaf combination where one leaf enforces $x_i < v_1$ and another enforces $x_i \geq v_2$, it must be that $v_2 < v_1$. Let d be the maximum tree depth in $\mathcal{T}$. Then, the number of leaf combinations is upper bounded by $2^{d|\mathcal{T}|}$. Since the number of feasibility checks for a single combination is $\frac{1}{2}|\mathcal{T}|(|\mathcal{T}| - 1)$, an upper bound on the total number of feasibility checks is $2^{d|\mathcal{T}|-1}|\mathcal{T}|(|\mathcal{T}| - 1)$. This observation implies that the worst case performance of an exact method improves as the number of trees decreases.

Chapter 4

Mixed-Integer Convex Nonlinear Optimisation with Gradient-Boosted Trees Embedded

This chapter designs exact methods computing either globally optimal solutions, or solutions within a quantified distance from the global optimum for Chapter 3 mixed-integer nonlinear convex Problem (3.5). The Problem (3.5) objective sums a discrete GBT-trained function and a continuous convex penalty function. We develop a branch-and-bound method exploiting both the GBT's combinatorial structure and the penalty function convexity. Numerical results substantiate our approach.

This chapter primarily develops our methodology in the context of using GBTs in Problem (3.1), the discussion is in a GBT context because our our numerical results are also all in the context of GBTs. However, aspects of our methodology can be generalised to other trained machine-learning models, e.g. assuming only decision-tree ensemble or simply ensemble structure. We define an ensemble to be a set of independent functions that all evaluate over $\boldsymbol{x} \in [\boldsymbol{L}, \boldsymbol{U}]$ and are summed to calculate the value predicted by the ensemble. Sections 4.1.2 and 4.1.3 develop decomposition-based lower bounding strategies, a refinement procedure for improving the decomposition-based lower bound and a strong branching method. These approaches are still

applicable to Problem (3.1) formulations where the learnt function is an *ensemble* (opposed to a *GBT ensemble*), e.g. the learnt function could be the result of applying boosting (Freund, 1995) where the weak learner is given by an alternative machine-learning approach. A requirement to make such a change would be the existence of an oracle that can optimise over the ensemble, e.g. if the ensemble can be formulated as a MILP, using a MILP solver as the oracle. Section 4.1.3 derives branches from the GBT split nodes and develops a pseudocost intialisation to order the branches. These aspects and the B&B algorithm is generally applicable to tree ensembles, e.g. we may use random forests or extremely randomised trees (Breiman, 2001; Geurts et al., 2006) in place of GBTs in the Problem (3.5) formulation.

Chapter Organisation Section 4.1 describes our branch-and-bound method. Section 4.2 defines the convex penalty term. Section 4.3 presents numerical results.

4.1 Branch-and-Bound Algorithm

This section designs an exact branch-and-bound (B&B) approach. The B&B algorithm exploits *spatial branching* that splits on continuous variables (Belotti et al., 2013). Table 4 defines the symbols in this section.

4.1.1 Overview

B&B Algorithm 5 spatially branches over the $[\boldsymbol{v}^L, \boldsymbol{v}^U]$ domain. It selects a variable x_i, a point v and splits interval $[v_i^L, v_i^U]$ into intervals $[v_i^L, v]$ and $[v, v_i^U]$. Each interval corresponds to an independent subproblem and a new B&B node. To avoid redundant branches, all GBT splits define the B&B branching points. At a given node, denote the reduced node domain by $S = [\boldsymbol{L}, \boldsymbol{U}]$. Algorithm 5 solves Problem (3.1) by relaxing the Equation (3.4) linking constraints and thereby separating the convex and GBT parts. Using this separation, Algorithm 5 computes corresponding bounds $b^{\text{cvx},S}$ and $b^{\text{GBT},S,P}$ independently, where the latter bound requires a tree ensemble partition P initialised at the root node and dynamically refined at each non-root node.

Algorithm 5 Branch-and-Bound (B&B) Algorithm Overview

1: $S = [\boldsymbol{L}, \boldsymbol{U}] \leftarrow [\boldsymbol{v}^L, \boldsymbol{v}^U]$
2: $b^{\text{cvx},S} \leftarrow \textsc{ConvexBound}(S)$ ▷ Lemma 1, Section 4.1.2
3: $P_{\text{root}} \leftarrow \textsc{RootNodePartition}(N)$ ▷ Section 4.1.2
4: $b^{\text{GBT},S,P_{\text{root}}} \leftarrow \textsc{GbtBound}(S, P_{\text{root}})$ ▷ Lemma 2, Section 4.1.2
5: $B \leftarrow \textsc{BranchOrdering}()$ ▷ Section 4.1.3
6: $Q = \{S\}$
7: **while** $Q \neq \emptyset$ **do**
8: Select $S \in Q$
9: **if** S is not leaf **then**
10: $S' \leftarrow S$
11: **repeat**
12: $S', (x_i, v) \leftarrow \textsc{StrongBranch}(S', B)$ ▷ Algorithm 7, Section 4.1.3
13: **until** strong branch not found
14: **if** S' is not leaf **then**
15: $(S_{\text{left}}, S_{\text{right}}) \leftarrow \textsc{Branch}(S', (x_i, v))$
16: P: tree ensemble partition of node S
17: $P' \leftarrow \textsc{PartitionRefinement}(P)$ ▷ Algorithm 6, Section 4.1.2
18: $b^{\text{GBT},S',P'} \leftarrow \textsc{GbtBound}(S', P')$ ▷ Lemma 2, Section 4.1.2
19: **for** $S_{\text{child}} \in \{S_{\text{left}}, S_{\text{right}}\}$ **do**
20: **if** S_{child} cannot be pruned **then** ▷ Section 4.1.2
21: $Q \leftarrow Q \cup \{S_{\text{child}}\}$
22: **end if**
23: **end for**
24: **end if**
25: **end if**
26: $Q \leftarrow Q \setminus \{S\}$
27: **end while**

Algorithm 5 begins by constructing the root node, computing a global lower bound, and determining a global ordering of all branches (lines 1–5). A given iteration: (i) extracts a node S from the unexplored node set Q, (ii) strong branches at S to cheaply identify branches that tighten the domain resulting in node S', (iii) updates the GBT lower bound at S', (iv) branches to obtain the child nodes S_{left} and S_{right}, (v) assesses if each child node $S_{\text{child}} \in \{S_{\text{left}}, S_{\text{right}}\}$ may now be pruned and, if not, (vi) adds S_{child} to the unexplored node set Q (lines 8–25).

The remainder of this section is structured as follows. Section 4.1.2 lower bounds Problem (3.1). Section 4.1.3 introduces a GBT branch ordering and leverages strong branching for cheap node pruning. Section 4.1.4 discusses heuristics for computing efficient upper bounds.

4.1.2 Lower Bounding

Global lower bound

The convex MINLP Problem (3.2) objective function consists of a *convex* (penalty) part and a *mixed-integer linear* (GBT) part. Lemma 1 computes a lower bound on the problem by handling the convex and GBT parts independently.

Lemma 1. *Let $S = [\boldsymbol{L}, \boldsymbol{U}] \subseteq [\boldsymbol{v}^L, \boldsymbol{v}^U]$ be a sub-domain of optimisation Problem (3.2). Denote by W^S the optimal objective value, i.e. the tightest relaxation, over the sub-domain S. Then, it holds that $W^S \geq \hat{W}^S$, where:*

$$\hat{W}^S = \underbrace{\left[\min_{\boldsymbol{x} \in S} \ \mathrm{cvx}(\boldsymbol{x})\right]}_{b^{\mathrm{cvx},S}} + \underbrace{\left[\min_{\boldsymbol{x} \in S} \sum_{t \in \mathcal{T}} \mathrm{GBT}_t(\boldsymbol{x})\right]}_{b^{\mathrm{GBT},S,*}}.$$

Proof. Proof Let $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in S}\{\mathrm{cvx}(\boldsymbol{x}) + \mathrm{GBT}(\boldsymbol{x})\}$ and observe that $\mathrm{cvx}(\boldsymbol{x}^*) \geq b^{\mathrm{cvx},S}$ and $\mathrm{GBT}(\boldsymbol{x}^*) \geq b^{\mathrm{GBT},S,*}$. □

We may compute $\hat{W}^S$ by removing the Equation (3.4) linking constraints and solving the mixed-integer model consisting of Equations (3.2) and (3.3). Computationally, the Lemma 1 separation leverages efficient algorithms for the convex part and commercial codes for the MILP GBT part. Lemma 1 treats the two Problem (3.1) objective terms independently, i.e. $\hat{W}^S$ separates the convex part from the GBT part. The Lemma 1 separation, while loose at the root node, may be leveraged to discard regions that are dominated by one of the objective terms. This approach resembles exact algorithms for multiobjective optimisation (Fernández and Tóth, 2009; Niebling and Eichfelder, 2016, 2019). An alternative approach, e.g. in line with augmented Lagrangian methods for stochastic optimisation (Bertsekas, 2014), would not separate the convex penalty term as in Lemma 1, but rather tighten the lower bound by keeping the convex penalty and GBTs integrated together. This would be an interesting alternative, but would eliminate the possibility of strong branching, i.e. the method proposed in Section 4.1.3.

GBT Lower Bound

While we may efficiently compute $b^{\text{cvx},S}$ (Boyd and Vandenberghe, 2004), deriving $b^{\text{GBT},S,*}$ is $\mathcal{NP}$-hard (Mišić, 2017). With the aim of tractability, we calculate a relaxation of $b^{\text{GBT},S,*}$. Lemma 2 lower bounds Problem (3.3), i.e. the GBT part of Problem (3.2), by partitioning the GBT ensemble into a collection of smaller ensembles.

Lemma 2. *Consider a sub-domain $S = [\boldsymbol{L}, \boldsymbol{U}] \subseteq [\boldsymbol{v}^L, \boldsymbol{v}^U]$ of the optimisation problem. Let $P = \{\mathcal{T}_1, \dots, \mathcal{T}_k\}$ be any partition of $\mathcal{T}$, i.e. $\cup_{i=1}^{k} \mathcal{T}_i = \mathcal{T}$ and $\mathcal{T}_i \cap \mathcal{T}_j = \emptyset$ $\forall 1 \leq i < j \leq k$. Then, it holds that $b^{\text{GBT},S,*} \geq b^{\text{GBT},S,P}$, where:*

$$b^{\text{GBT},S,P} = \sum_{\mathcal{T}' \in P} \left[\min_{\boldsymbol{x} \in S} \left\{ \sum_{t \in \mathcal{T}'} \text{GBT}_t(\boldsymbol{x}) \right\} \right].$$

Proof. When evaluating $\text{GBT}(\boldsymbol{x})$ at a given $\boldsymbol{x}$, each tree $t \in \mathcal{T}$ provides its own independent contribution $\text{GBT}_t(\boldsymbol{x})$, i.e. a single leaf. A feasible selection of leaves has to be consistent with respect to the GBT node splits, i.e. if one leaf splits on $x_i < v_1$ and another splits on $x_i \geq v_2$ then $v_1 > v_2$. Relaxing this consistency requirement by considering a partition P of $\mathcal{T}$ derives the lower bounds $b^{\text{GBT},S,P}$ for any partition P. □

The tightness of the Lemma 2 GBT lower bound is dependent on the partition P used. For example, the partition $P_1 = \{\{t\} \,|\, t \in \mathcal{T}\}$ corresponds to trivial GBT lower bound $\sum_{t \in \mathcal{T}} \min_{l \in \mathcal{L}_t} F_{t,l}$, i.e. summing the minimal leaves in each tree, whereas the partition $P_2 = \{\mathcal{T}\}$ is equivalent to solving Problem (3.3) to optimality, i.e. the tightest possible GBT lower bound. However, Lemma 2 corresponding to a partition that has subsets containing a large number of trees, e.g. P_2, may be more difficult to solve since the individual MILPs represent large GBTs. Section 4.3 numerically assesses this trade-off of tightness versus runtime. Depending on the partition used, a Lemma 2 GBT lower bound may be looser or tighter than a continuous (linear programming) relaxation solution of Problem (3.3). Clearly, a Lemma 2 GBT lower bound corresponding to P_2 is always at least as tight as a continuous relaxation solution. Example 4.1 shows that the Lemma 2 GBT lower bound can be looser than a Problem (3.3) continuous relaxation lower bound for some partitions of $\mathcal{T}$.

Example 4.1. *We describe a GBT instance whose continuous relaxation lower bound is tighter than the Lemma 2 GBT lower bound for some partitions of* $\mathcal{T}$*. Let* $\mathcal{T} = \{T_1, T_2\}$ *be a GBT instance with two trees where all* $F_{t,l}$*,* $t \in \mathcal{T}$*,* $l \in \mathcal{L}_t$ *are unique. Let* $l^*_{T_1} = \arg\min_{l \in \mathcal{L}_{T_1}} F_{T_1,l}$*. Then the weight of leaf* $l \in \mathcal{L}_{T_1}$ *can be written as* $F_{T_1,l} = F_{T_1,l^*_{T_1}} + \gamma_{T_1,l}$ *where* $\gamma_{T_1,l} \geq 0$*. Similarly, we can define* $l^*_{T_2}$*, and* $\gamma_{T_2,l}$ *that are defined from tree* T_2*. Assume that* $l^*_{T_1}$ *and* $l^*_{T_2}$ *cannot be selected together in an integer feasible solution. The objective of Problem (3.3) can be rewritten as* $F_{T_1,l^*_{T_1}} + F_{T_2,l^*_{T_2}} + \sum_{t\in\mathcal{T}} \sum_{l\in\mathcal{L}_t} \gamma_{t,l} z_{t,l}$*, since Equation* (3.3b) *holds. The minimum of this equivalent objective, subject to Equations* (3.3b) *and* (3.3g)*, occurs when* $\sum_{t\in\mathcal{T}} \sum_{l\in\mathcal{L}_t} \gamma_{t,l} z_{t,l} = 0$*. This case only arises when* $z_{T_1,l^*_{T_1}} = z_{T_2,l^*_{T_2}} = 1$*, since all leaf weights are unique. But this is an integer feasible solution that contradicts the assumption that leaves* $l^*_{T_1}$ *and* $l^*_{T_2}$ *cannot be selected together. Hence, a feasible assignment to continously relaxed* $\boldsymbol{y}$ *does not correspond to* $z_{T_1,l^*_{T_1}} = z_{T_2,l^*_{T_2}} = 1$*, since, by assumption, it would contradict with the constraint set given by Equations* (3.3c) *to* (3.3e)*. All other (possibly fractional) assignments to* $z_{t,l}$ *are such that* $\sum_{t\in\mathcal{T}} \sum_{l\in\mathcal{L}_t} \gamma_{t,l} z_{t,l} > 0$*, as all leaf values are unique. Hence, the linear programming relaxation of this instance yields a tighter bound than that of the Lemma 2 GBT lower bound when using the partition* $P = \{\{T_1\}, \{T_2\}\}$*.*

Root Node Partition B&B Algorithm 5 chooses an initial root node partition P_{root} with subsets of size N and calculates the associated Lemma 2 lower bound. Section 4.3 numerically decides the partition size N for the considered instances. The important factors for a subset size N are the tree depth, the number of continuous variable splits and their relation with the number of binary variables.

Non-Root Node Partition Refinement Any non-root B&B node has reduced domain $\boldsymbol{x} \in S = [\boldsymbol{L}, \boldsymbol{U}] \subset [\boldsymbol{v}^L, \boldsymbol{v}^U]$. B&B Algorithm 5 only branches on GBT node splits, so modelling the reduced domain S in MILP Problem (3.3) is equivalent to setting $y_{i,j} = 0$ or $y_{i,j} = 1$ for any $y_{i,j}$ that corresponds to $x_i \leq L_i$ or $x_i \geq U_i$, respectively. Reducing the box-constrained domain at the node level equates to reducing the GBT instance size. In particular, we may reduce the number and height of trees by assigning fixed variable values and cancelling redundant

constraints similarly to Mišić (2017).

Assume that, at some non-root node with domain S, the algorithm is about to update $b^{\text{GBT},S',P'}$ which was calculated at the parent node with domain $S' \supset S$. Fixing binary variables $y_{i,j}$ subject to domain S reduces the worst case enumeration cost of calculating $b^{\text{GBT},S,P'}$. The GBT lower bound may further improve at S by considering an alternative partition P such that $|P| < |P'|$, i.e. reducing the number of subsets. However, reducing the number of subsets has challenges because: (i) choosing any partition P does not necessarily guarantee $b^{\text{GBT},S,P} \geq b^{\text{GBT},S',P'}$, and (ii) a full Lemma 2 calculation of $b^{\text{GBT},S,P}$ may still be expensive when considering the cumulative time across all B&B nodes. Refinability Definition 4.1 addresses the choice of P such that $b^{\text{GBT},S,P} \geq b^{\text{GBT},S',P'}$.

Definition 4.1. *Given two partitions P' and P'' of set $\mathcal{T}$, we say that P'* refines *P'' if and only if $\forall \mathcal{T}' \in P'$, $\exists \mathcal{T}'' \in P''$ such that $\mathcal{T}' \subseteq \mathcal{T}''$. This definition of refinement implies a partial ordering between different partitions of $\mathcal{T}$. We express the refinement relation by $\preceq$, i.e. $P' \preceq P''$ if and only if P' refines P''.*

Example 4.2. *Let $P = \{\{1,2,3\},\{4,5\}\}$, $P' = \{\{1\},\{2\},\{3\},\{4\},\{5\}\}$ and $P'' = \{\{1,2\},\{3,4,5\}\}$ be partitions of $\{1,\ldots,5\}$. Here P' refines P since every subset in P' is contained in one of the P subsets. Similarly P' refines P''. Partition P does not refine P'' nor does P'' refine P.*

Lemma 3 allows bound tightening by partition refinements. Its proof is similar to Lemma 2.

Lemma 3. *Let P and P' be two partitions of $\mathcal{T}$. If $P' \preceq P$, then $b^{\text{GBT},P'} \leq b^{\text{GBT},P}$.*

In general, for two partitions P and P', we do not know a priori which partition results in a superior GBT lower bound. However, by Lemma 3, P' refining P suffices for $b^{\text{GBT},P} \geq b^{\text{GBT},P'}$. Therefore, given partition P' for the parent node, constructing P for the child node S by unifying subsets of P' will not result in inferior lower bounds.

Algorithm 6 improves $b^{\text{GBT},S',P'}$ at node S by computing a refined partition P. Suppose that $P' = \{\mathcal{T}_1,\ldots,\mathcal{T}_k\}$. Each GBT ensemble subset $\mathcal{T}' \in P'$ corresponds to a smaller subproblem with $n^{\mathcal{T}',S}$ leaves ($z_{t,l}$ variables) over the domain S. Initially, Algorithm 6 sorts the subsets of

Algorithm 6 Non-Root Node Partition Refinement

P': parent node partition
Sort $P' = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ so that $n^{\mathcal{T}_1} \leq \ldots \leq n^{\mathcal{T}_k}$
$P \leftarrow \emptyset$
$i = 1$
while $i < \lfloor n/2 \rfloor$ and the time limit is not exceeded **do**
    $P \leftarrow P \cup \{\mathcal{T}_{2i-1} \cup \mathcal{T}_{2i}\}$
    $i \leftarrow i + 1$
end while
$P \leftarrow P \cup \{\mathcal{T}_j \in P' : j > i\}$
return P

P' in non-decreasing order of $n^{\mathcal{T}',S}$. Then, it iteratively takes the union of consecutive pairs and calculates the associated lower bound, i.e. the first calculation is for $b^{\text{GBT},S,\{\mathcal{T}_1 \cup \mathcal{T}_2\}}$, the second is for $b^{\text{GBT},S,\{\mathcal{T}_3 \cup \mathcal{T}_4\}}$ and so forth. The iterations terminate when all unions have been recalculated, or at user defined time limit q resulting in two sets of bounds: those that are combined and recalculated, and those that remain unchanged. Assuming that the final subset that is updated has index $2l$, the new partition of the trees at node S is $P = \{\mathcal{T}_1 \cup \mathcal{T}_2, \ldots, \mathcal{T}_{2l-1} \cup \mathcal{T}_{2l}, \mathcal{T}_{2l+1}, \ldots, \mathcal{T}_k\}$ with GBT bound $b^{\text{GBT},S,P} = \sum_{i=1}^{l} b^{\text{GBT},S,\{\mathcal{T}_{2i-1} \cup \mathcal{T}_{2i}\}} + \sum_{i=2l+1}^{k} b^{\text{GBT},S',\{\mathcal{T}_i\}}$. The second sum is a result of placing time limit q on updating the GBT lower bound. Time limit q maintains a balance between searching and bounding. Unifying any number of subsets satisfies Lemma 3, but Algorithm 6 unifies pairs to keep the resulting subproblems manageable. One may speed up our lower bounding procedure by reducing the height of the GBTs, thus relaxing feasibility, and converting each partition subset $\mathcal{T}_k$ solution into a feasible one for $\mathcal{T}_k$ using the Mišić (2017) split generating procedure for fixing violated constraints.

Node Pruning

In the B&B algorithm, each node can access: (i) the current best found feasible objective f^*, (ii) a lower bound on the convex penalties $b^{\text{cvx},S}$, and (iii) a lower bound on the GBT part $b^{\text{GBT},S}$. The algorithm prunes node S if:

$$b^{\text{cvx},S} + b^{\text{GBT},S} > f^*, \tag{4.1}$$

i.e. if all feasible solutions in S have objective inferior to f^*.

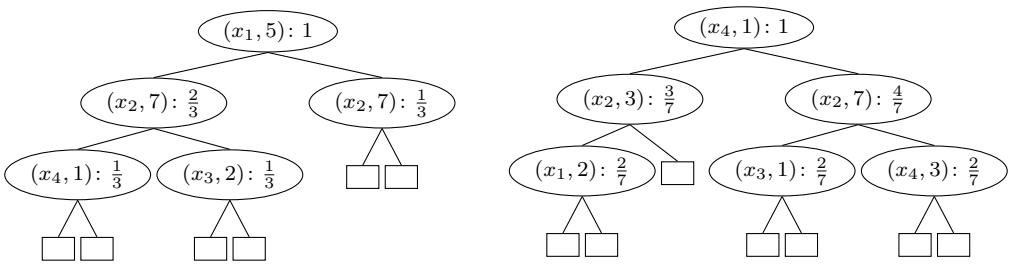


Figure 4.1: Example 4.3 node contributions to Equation (4.2) weight calculation. Each split node contains '$(x_i, v) : w'$ where (x_i, v) is the split pair and w is the node's contribution to (x_i, v)'s weight. We calculate w as the proportion of leaves covered relative to the total number of leaves.

4.1.3 Branching

Branch Ordering

Next branch selection is a critical element of B&B Algorithm 5. Each branch is a GBT split (x_i, v) choice and eliminates a certain number of GBT leaves. Branching with respect to a GBT split that covers a larger number of leaves may lead to a smaller number of subsequent B&B iterations by reducing the GBT size.

Selecting an ideal (x_i, v) split that most improves the GBT lower bound is challenging as it may require solving multiple expensive MILPs. So, we heuristically approximate this task by quantifying splits that occur often among all trees and influence a larger number of leaves in participating trees. Let $r((x_i, v), t)$ and cover$(s,\ t)$ return the set of nodes in tree t that split on (x_i, v) and the set of leaves that node $s \in t$ covers, respectively. We initialise pseudocosts by weighting the (x_i, v) splits as:

$$\text{weight}((x_i, v), t) = |\mathcal{L}_t|^{-1} \sum_{s \in r((x_i, v), t)} |\operatorname{cover}(s, t)|, \tag{4.2a}$$

$$\text{weight}((x_i, v), \mathcal{T}) = \sum_{t \in \mathcal{T}} \text{weight}((x_i, v), t). \tag{4.2b}$$

Equation (4.2a) weights (x_i, v) as the fraction of leaves covered by nodes splitting on (x_i, v) in tree t. Equation (4.2b) sums all weights calculated by Equation (4.2a) for split (x_i, v) in each tree $t \in \mathcal{T}$.

Example 4.3. *Figure 4.1 shows the weight given to each node for two trees. The left tree contains 6 leaves and the right tree contains 7 leaves. Consider split* $(x_2, 7)$*. The left tree contains two nodes splitting on* $(x_2, 7)$ *one of which covers 4 out of 6 leaves and the other covers 2 out of 6 leaves therefore these nodes contribute* $\frac{2}{3}$ *and* $\frac{1}{3}$*, respectively, to the weight. Similarly, the right tree contains a single node splitting on* $(x_2, 7)$ *which covers 4 out of 7 leaves therefore this node contributes* $\frac{4}{7}$ *to the weight. We obtain the weight for* $(x_2, 7)$ *by summing these values, i.e.* $\text{weight}((x_2, 7), \mathcal{T}) = \frac{2}{3} + \frac{1}{3} + \frac{4}{7} = 1\frac{4}{7}$.

We note that the Equation (4.2) weight function initialises pseudocosts satisfying the following properties:

1. for each tree t, $\text{weight}((x_i, v), t)$ is increasing with $|\,\text{cover}((x_i, v), t)|$,
2. if (x_i, v) and $(x_{i'}, v')$ cover the same set of leaves then $\text{weight}((x_i, v), s') = \text{weight}((x_i, v), s)$.

Strong Branching

Branch selection is fundamental to any B&B algorithm. *Strong branching* selects a branch that enables pruning with low effort computations and achieves a non-negligible speed-up in the algorithm's performance (Morrison et al., 2016). Strong branching increases the size of efficiently solvable large-scale mixed-integer problems and is a major solver component (Klabjan et al., 2001; Anstreicher et al., 2002; Anstreicher, 2003; Easton et al., 2003; Belotti et al., 2009; Misener and Floudas, 2013; Kılınç et al., 2014). Here, strong branching leverages the easy-to-solve convex penalty term for pruning.

At a B&B node S, branching produces two children S_{left} and S_{right}. Strong branching Algorithm 7 considers the branches in their Section 4.1.3 pseudo-cost ordering and assesses each branch by computing the associated convex bound. Under the strong branching test, one node among S_{left} and S_{right} inherits the convex bound $b^{\text{cvx},S}$ from the parent, while the other requires a new computation. Suppose that $S' \in \{S_{\text{left}}, S_{\text{right}}\}$ does not inherit $b^{\text{cvx},S}$. If $b^{\text{cvx},S'}$ satisfies the Equation (4.1) pruning condition without GBT bound improvement, then S' is immediately

Algorithm 7 Strong Branching

S: B&B node with bounds $b^{\text{GBT},S}$ and $b^{\text{cvx},S}$
$B^S = [(x_{i_1}, v_1), \ldots, (x_{i_l}, v_l)]$: l next branches list w.r.t. Section 4.1.3 pseudo-cost order
for $(x_i, v) \in B^S$ **do**
    $S_{\text{left}}, S_{\text{right}}$: S children by branching on (x_i, v)
    Compute $b^{\text{cvx},S_{\text{left}}}$ and $b^{\text{cvx},S_{\text{right}}}$
    if $\max\{b^{\text{cvx},S_{\text{left}}}, b^{\text{cvx},S_{\text{right}}}\} + b^{\text{GBT},S} < f^*$ **then**
        return $\arg\min\{b^{\text{cvx},S_{\text{left}}}, b^{\text{cvx},S_{\text{right}}}\}, (x_i, v)$
    end if
end for
return $S, (x_{i_1}, v_1)$

selected as the strong branch and strong branching repeats at the other child node S''. Figure 4.2 illustrates strong branching. When Algorithm 5 does not find a strong branch, it performs a GBT lower bound update and branches on the first item of the branch ordering. Algorithm 5 then adds this node's children to a set of unexplored nodes and continues with the next B&B iteration. Strong branching Algorithm 7 may also be viewed as a form of bound contraction. Bound contraction solves auxiliary problems that aim to tighten variable upper and lower bounds by utilising the relaxation solutions or feasible solutions (Quesada and Grossmann, 1993; Maranas and Floudas, 1995; Tawarmalani and Sahinidis, 2002; Karuppiah and Grossmann, 2006; Faria and Bagajewicz, 2012; Castro and Grossmann, 2014). Bound contraction techniques are used in global optimisation of non-convex MINLP as convex relaxations for non-convex constraints are often dependent on variable bounds, hence tightening variable bounds can tighten relaxed constraints as well (Belotti et al., 2009; Castro, 2015). Strong branching Algorithm 7 results in a sequence of branches whose sibling branches are all infeasible. Hence, each strong branch may be viewed as tightening the domain of the node at which strong branching was applied, i.e. bound contraction. Also, the strong branches may tighten the MILPs that calculate the Lemma 2 GBT lower bound, this is similar to the effect that bound contraction has on convex relaxations of non-convex constraints in global optimisation.

Strong branching allows efficient pruning when the convex objective part is significant. Strong branching may reduce the computational overhead incurred by GBT bound recalculation when Algorithm 7 selects multiple strong branches between GBT bound updates. While a single strong branch assessment is negligible, the cumulative cost of calculating convex bounds for all

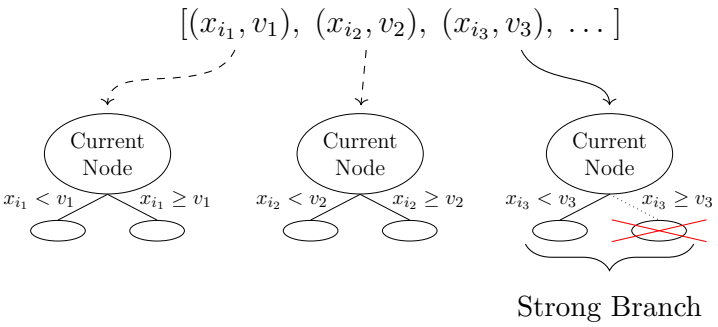


Figure 4.2: Strong branching for selecting the next spatial branch. A strong branch leads to a node that is immediately pruned, based on a convex bound computation.

branches may be high. Section 4.1.3 orders the branches according to a measure of effectiveness aiding GBT bounding, so the time spent deriving strong branches with small weighting function may be better utilised in improving the GBT bound. Opposed to full strong branching, i.e. assessing all branches, strong branching Algorithm 7 uses a lookahead approach (Achterberg et al., 2005). Parameterised by a lookahead value $l \in \mathbb{Z}_{>0}$, Algorithm 7 investigates the first l branches. If Algorithm 7 finds a strong branch, Algorithm 5 repeats Algorithm 7, otherwise the B&B Algorithm 5 updates the GBT bound $b^{\mathrm{GBT},S,P}$ at the current node. Algorithm 7 keeps strong branching checks relatively cheap and maintains a balance between searching and bounding.

4.1.4 Heuristics

To prune, i.e. satisfy Equation (4.1), consider two heuristic methods generating good feasible solutions to Problem (3.1): (i) a mixed-integer convex programming (convex MINLP) approach, and (ii) particle swarm optimisation (PSO) (Eberhart and Kennedy, 1995; Kennedy and Eberhart, 1995). The mixed-integer approach uses the decomposability of GBT ensembles, i.e. while convex MINLP solvers provide weak feasible solutions for large-scale instances of Problem (3.1), they may efficiently solve moderate instances to global optimality (Westerlund and Pettersson, 1995; Tawarmalani and Sahinidis, 2005; Vigerske, 2012; Misener and Floudas, 2014; Lundell et al., 2017). The PSO approach exploits trade-offs between the convex and objective GBT parts. Metaheuristics like particle swarm optimisation and simulated annealing (Kirkpatrick et al., 1983) may produce good heuristic solutions, as a preprocessing step, before

the branch-and-bound algorithm begins. Simpler convex MINLP heuristics may improve upper bounds at the branch-and-bound node level because of their efficient running times.

Mixed-Integer Convex Programming Heuristic

For a given a subset $\mathcal{T}' \subseteq \mathcal{T}$ of trees, let $f_{\mathcal{T}'}(\cdot)$ be the objective function obtained by ignoring the trees $\mathcal{T} \setminus \mathcal{T}'$. Then, $\min_{\boldsymbol{v}^L \leq \boldsymbol{x} \leq \boldsymbol{v}^U}\{f_{\mathcal{T}'}(\boldsymbol{x})\}$ may be significantly more tractable than the original problem instance when $|\mathcal{T}'| << |\mathcal{T}|$. So, the Algorithm 8 heuristic solves the original convex MINLP by sequentially solving smaller convex MINLP sub-instances of increasing size. A sub-instance is restricted to a subset $\mathcal{T}' \subseteq \mathcal{T}$ of GBTs. Let $\mathcal{T}^{(k)}$ be the subset of trees when the k-th heuristic iteration begins. Initially, $\mathcal{T}^{(0)} = \emptyset$, i.e. $f_{\mathcal{T}^{(0)}}(\cdot)$ consists only of the convex part. Denote by $\boldsymbol{x}^{(k)}$ the sub-instance optimal solution minimising $f_{\mathcal{T}^{(k)}}(\cdot)$. Note that $\boldsymbol{x}^{(k)}$ is feasible for the full instance. Each iteration k chooses a set of N additional trees $\mathcal{T}^{\text{next}} \subseteq \mathcal{T} \setminus \mathcal{T}^{(k)}$ and constructs $\mathcal{T}^{(k+1)} = \mathcal{T}^{(k)} \cup \mathcal{T}^{\text{next}}$, i.e. $\mathcal{T}^{(k)} \subseteq \mathcal{T}^{(k+1)}$. Consider two approaches for picking the N trees between consecutive iterations: (i) training-aware selection and (ii) best improvement selection. Termination occurs when the time limit is exceeded and Algorithm 8 returns the best computed solution.

Algorithm 8 Mixed-integer convex programming heuristic

1: $k \leftarrow 0$
2: $\mathcal{T}^{(k)} \leftarrow \emptyset$
3: **while** the time limit is not exceeded **do**
4: $\quad \boldsymbol{x}^{(k)} \leftarrow \underset{\boldsymbol{v}^L \leq \boldsymbol{x} \leq \boldsymbol{v}^U}{\arg\min} f_{T^{(k)}}(\boldsymbol{x})$
5: $\quad$ Choose $\mathcal{T}^{\text{next}}$ from $\left\{\mathcal{T}' \mid \mathcal{T}' \subseteq \mathcal{T} \setminus \mathcal{T}^{(k)}, |\mathcal{T}'| = \min\{N, |\mathcal{T} \setminus \mathcal{T}^{(k)}|\}\right\}$
6: $\quad \mathcal{T}^{(k+1)} \leftarrow \mathcal{T}^{(k)} \cup \mathcal{T}^{\text{next}}$
7: $\quad k \leftarrow k + 1$
8: **end while**
9: **return** $\underset{k \in \{0,\ldots,k-1\}}{\arg\min} f\left(\boldsymbol{x}^{(k)}\right)$

Training-aware selection Let $T_1, T_2, \ldots, T_m$ be the tree generation order during training. This approach selects the trees $\mathcal{T}^{\text{next}}$ according to this predefined order. That is, in the k-th iteration, $\mathcal{T}^{(k)} = \{T_1, \ldots, T_{kN}\}$ and $\mathcal{T}^{\text{next}} = \{T_{kN+1}, \ldots, T_{(k+1)N}\}$. A GBT training algorithm constructs the trees iteratively, so each new tree reduces the current GBT ensemble error

with respect to the training data. Thus, we expect that the earliest-generated trees better approximate the learned function than the latest-generated trees. Specifically, for two subsets $\mathcal{T}_A, \mathcal{T}_B \subseteq \mathcal{T}$ with the property that $t_a < t_b$ for each $T_{t_a} \in \mathcal{T}_A$ and $T_{t_b} \in \mathcal{T}_B$, we expect that $|f_{\mathcal{T}_A}(\boldsymbol{x}) - f^*(\boldsymbol{x})| \leq |f_{\mathcal{T}_B}(\boldsymbol{x}) - f^*(\boldsymbol{x})|$, for each $\boldsymbol{v}^L \leq \boldsymbol{x} \leq \boldsymbol{v}^U$, where f^* is the original objective function, i.e. the optimal approximation. Intuitively, earlier trees place the GBT function within the correct vicinity, while later trees have a fine tuning role.

Best improvement selection In this approach, the k-th iteration picks the N trees with the maximum contribution when evaluating at $\boldsymbol{x}^{(k)}$. We select $\mathcal{T}^{\text{next}} \subseteq \mathcal{T} \setminus \mathcal{T}^{(k)}$ so that, for each pair of trees $T_t \in \mathcal{T}^{\text{next}}$ and $T_{t'} \in \mathcal{T} \setminus (\mathcal{T}^{(k)} \cup \mathcal{T}^{\text{next}})$, it holds that $f_t(\boldsymbol{x}^{(k)}) \geq f_{t'}(\boldsymbol{x}^{(k)})$. Assuming that approximation $\mathcal{T}^{(k)}$ is poor, then $\mathcal{T}^{\text{next}}$ contains the trees that refute optimality of $\boldsymbol{x}^{(k)}$ the most, from the perspective of $f_t(\boldsymbol{x}^{(k)})$ $t \in \mathcal{T} \setminus \mathcal{T}^{(k)}$.

Particle Swarm Optimisation

For Problem (3.1), we improve the PSO performance by avoiding initial particle positions in feasible regions strictly dominated by the convex term. We project the initial random points close to regions where the GBT term is significant compared to the convex term.

4.2 Case Studies

Our case studies consider GBT instances where training data is not evenly distributed over the $[\boldsymbol{v}^L, \boldsymbol{v}^U]$ domain. So, while $\boldsymbol{x} \in [\boldsymbol{v}^L, \boldsymbol{v}^U]$ is feasible, GBT($\boldsymbol{x}$) may be less meaningful for $\boldsymbol{x}$ far from training data. The Problem (3.1) cvx($\boldsymbol{x}$) function, for the case studies, is a penalty function constructed with principal component analysis (PCA) (Jolliffe, 2002).

PCA characterises a large, high-dimensional input data set $D = \{\boldsymbol{d}^{(1)}, \ldots, \boldsymbol{d}^{(p)}\}$ with a low-dimensional subspace capturing most of the variability (James et al., 2013). PCA defines a set of n ordered, orthogonal *loading vectors*, ϕ_i, such that ϕ_i captures more variability than $\phi_{i'}$, for

$i < i'$. PCA on D defines parameters $\boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^n$ and $\boldsymbol{\Phi} = [\phi_1 \ \dots \ \phi_n] \in \mathbb{R}^{n \times n}$, i.e. the sample mean, sample standard deviation and loading vectors, respectively. Vectors $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ standardise D since PCA is sensitive to scaling. Often, only a few ($k < n$) leading loading vectors capture most of the variance in D and $\boldsymbol{\Phi'} = [\phi_1 \ \dots \ \phi_k]$ may effectively replace $\boldsymbol{\Phi}$. $\boldsymbol{P} = \boldsymbol{\Phi'}\boldsymbol{\Phi'}^\top$ defines a projection matrix to the subspace spanned by $\{\phi_1, \dots, \phi_k\}$. Penalising solutions further from training data with PCA defined projection matrix $\boldsymbol{P}$:

$$\text{cvx}_\lambda(\boldsymbol{x}) = \lambda \left\| (\boldsymbol{I} - \boldsymbol{P}) \, \text{diag}(\boldsymbol{\sigma})^{-1} (\boldsymbol{x} - \boldsymbol{\mu}) \right\|_2^2 \tag{4.3}$$

where $\lambda > 0$ is a penalty parameter, $\boldsymbol{I}$ is the identity matrix and $\text{diag}(\cdot)$ is a matrix with the argument on the diagonal. Larger λ is more conservative with respect to PCA subspace $\boldsymbol{P}$. Note in Equation (4.3) that our specific nonlinear convex penalty is a convex quadratic.

The parameter λ in Equation (4.3) scales the distance to the PCA subspace defined by $\boldsymbol{P}$. If λ is large then the corresponding Problem (3.1) objective is more influenced by Equation (4.3) over large regions of the $[\boldsymbol{L}, \boldsymbol{U}]$ domain. Hence, the GBT function is more relevant, with respect to optimality, in regions where Equation (4.3) evaluates to a small value. If λ is small then the $[\boldsymbol{L}, \boldsymbol{U}]$ domain is less dominated by Equation (4.3) and the corresponding Problem (3.1) may more closely resemble optimising over the GBT function alone. Parameter λ captures conservativeness with respect to subspace $\boldsymbol{P}$ because, for large values, solutions that deviate from subspace $\boldsymbol{P}$ may equate to large increases in the Problem (3.1) objective evaluations regardless of the contribution from the GBT function. Small λ values give more freedom to deviate from subspace $\boldsymbol{P}$, as the Problem (3.1) the contribution of Equation (4.3) is less dominating. In practice, the effect of λ on a given Problem (3.1) instance may not be known and λ may need to be tuned. An approach to tuning parameter λ considers a sequence of decreasing values $\lambda^{(i)}$ and the corresponding sequence of solutions to Problem (3.1), $\boldsymbol{x}^{(*,i)}$. The value $\text{cvx}(\boldsymbol{x}^{(\boldsymbol{*,i})})/\lambda^{(i)}$, gives a measure of distance from subspace $\boldsymbol{P}$. If $\text{cvx}(\boldsymbol{x}^{(\boldsymbol{*,i+1})})/\lambda^{(i+1)} - \text{cvx}(\boldsymbol{x}^{(\boldsymbol{*,i})})/\lambda^{(i)}$ is increasing significantly then we have an indicator that the optimal Problem (3.1) solutions are becoming less conservative. Our Section 4.3 numerical tests assess how λ affects the performance of the B&B algorithm. Assessing how to tune λ efficiently is an interesting direction of future research.

Equation (4.3) aims to characterise the region containing the training data with an affine subspace. Points in the subspace are not penalised and points close to the subspace are not heavily penalised. However, Equation (4.3) may be qualitatively less effective when the standardised training data is not evenly distributed within subspace $\boldsymbol{P}$.

Example 4.4. *Consider a data set* $\left\{\boldsymbol{x}^{(i)}\right\}_{i=1}^{2m}$, $\boldsymbol{x}^{(i)} \in \mathbb{R}^3$ *where* $x_1^{(i)} \sim U(0,1), \forall\ i \in [2m]$, *and* $x_2^{(i)} = x_3^{(i)} = 0$, $x_2^{(m+i)} = x_3^{(m+i)} = 1, \forall\ i \in [m]$. *The 2D subspace containing these points contains the origin and directions* $(1,0,0)^T, (0,1,1)^T$. *Equation* (4.3) *does not penalise points in this subspace. But the point* $(0.5, 0.5, 0.5)$, *which is contained in the subspace, is far from the training data when considering the subspace distribution. Having* $x_2^{(i)}, x_3^{(i)} \sim N(0, \varepsilon)$, $x_2^{(m+i)}, x_3^{(m+i)} \sim N(1, \varepsilon), \forall\ i \in [m]$ *and small* $\varepsilon > 0$, *introduces an error term to the second and third variables while retaining the same clustered distribution over the subspace.*

Clustering, e.g. Example 4.4, may be handled by the Section 4.1 B&B. We could instantiate a separate instance for each cluster using a penalty that only considers training data in a given cluster and limit the solve to a reduced box domain. A single problem formulation considering more complex training data relationships may negatively affect the strong branching aspect of B&B Algorithm 5.

4.3 Numerical Results

This section compares the Section 4.1 heuristic, lower bounding and branch-and-bound algorithms to black-box solvers. Section 4.3.1 provides information about the system specifications and the solvers. Sections 4.3.2 and 4.3.3 investigate two GBT instances for engineering applications, namely: (i) concrete mixture design and (ii) chemical catalysis. Section 4.3.4 discusses observations from the Sections 4.3.2 and 4.3.3 results. The concrete mixture design instance is from the UCI machine learning repository (Dheeru and Karra Taniskidou, 2017). The industrial chemical catalysis instance is provided from BASF. Table 4.1 presents information about these instances. For both instances, we model closeness to training data using the PCA-based function $\text{cvx}(\boldsymbol{x})$ defined in Equation (4.3).

Table 4.1: Instance sizes

	Concrete Mixture Design	Chemical Catalysis
GBT attributes:		
Number of trees	7,750	8,800
Maximum depth	16	16
Number of leaves ($z_{t,l}$)	131,750	93,200
Number of x_i continuous variables	8	42
Convex MINLP (3.2) attributes:		
Number of continuous variables ($\#z_{t,l} + \#x_i$)	131,758	93,242
Number of $y_{i,j}$ binary variables	8,441	2,061
Number of constraints	281,073	183,791

4.3.1 System and Solver Specifications

Experiments are run on an Ubuntu 16.04 HP EliteDesk 800 G1 TWR with 16GB RAM and an Intel Core i7-4770@3.40GHz CPU. Implementations are in Python 3.5.3 using Pyomo 5.2 (Hart et al., 2011, 2017) for mixed-integer programming modelling and interfacing with solvers. We use CPLEX 12.7 and Gurobi 7.5.2 as: (i) black-box solvers for the entire convex MINLP (3.2), (ii) heuristic components for solving convex MINLP (3.2) instances in the Section 4.1.4 convex MINLP heuristic, and (iii) branch-and-bound algorithm components for solving MILP (3.3) instances in the Section 4.1.2 GBT lower bounding procedure. The R package GenSA (Xiang et al., 2013) runs the simulated annealing (SA) metaheuristic. The Python module PySwarms (Miranda, 2018) implements the Section 4.1.4 particle swarm optimisation (PSO) metaheuristic. Note that current versions of CPLEX and Gurobi cannot solve general convex MINLP, so we would use a more general solver if we had non-quadratic penalty functions. All results report wall clock times.

This chapter evaluates the (i) objective lower bounding procedure, and (ii) branch-and-bound algorithm, both of which use CPLEX or Gurobi as a black-box MILP solver. We also apply CPLEX and Gurobi to: the entire MINLP for evaluating branch-and-bound Algorithm 5 and the mixed-integer convex programming heuristic. Figures 5-12 append labels -C and -G to indicate CPLEX and Gurobi, respectively, and use different line types for displaying the results. At nodes immediately following a GBT bound update, the B&B algorithm assesses solutions from

solving the convex part of Problem (3.1) as heuristics solutions. We use the default CPLEX 12.7 and Gurobi 7.5.2 tolerances, i.e. relative MIP gap, integrality and barrier convergence tolerances of 10^{-4}, 10^{-5} and 10^{-8}, respectively. We use the default SA parameters. We parameterise PSO with inertia term $\omega = 0.5$, cognitive term $c_1 = 0.7$, social term $c_2 = 0.3$, 500 particles and an iteration limit of 100. Each particle takes a randomly generated point, $\boldsymbol{x}^{(0)} \in [\boldsymbol{v}^L, \boldsymbol{v}^U]$, and its projection, $\boldsymbol{x}^{(p)}$ on $\boldsymbol{P}$ and initialises at $\boldsymbol{x} = h \cdot \boldsymbol{x}^{(0)} + (1 - h) \cdot \boldsymbol{x}^{(p)}$. For our tests, we use $h = 0.15$.

4.3.2 Concrete Mixture Design

In concrete mixture design, different ingredient proportions result in different properties of the concrete, e.g. compressive strength. The relationship between ingredients and properties is complex, so black-box machine learning is well suited for the function estimation task (Chou et al., 2011; Erdal, 2013; DeRousseau et al., 2018).

Instance

We maximise concrete compressive strength where GBTs are used for modelling. Since we maximise concrete compressive strength, negating all leaf weights $F_{t,l}$ forms an equivalent GBT instance that fits the Problem (3.1) minimisation formulation. We use the Yeh (1998) concrete compressive strength dataset from the UCI machine learning repository (Dheeru and Karra Taniskidou, 2017). This dataset has $n = 8$ continuous variables. R packages gbm (Ridgeway, 2017) and caret (Kuhn, 2008) are used for GBT training. Root-mean-square error is used for model selection. The resulting GBT instance has 7,750 trees with max depth 16. The PCA based convex penalty has $\text{rank}(\boldsymbol{P}) = 4$, i.e. we select the first four loading vectors.

Heuristic Solutions

Table 4.2 compares the CPLEX 12.7, Gurobi 7.5.2, SA, and PSO computed solutions for the entire convex MINLP, under 1 hour time limit. SA performs the best. PSO solution is relatively

Table 4.2: Concrete mixture design instance: black-box solver solutions (upper bounds) by solving the entire mixed-integer convex programming (convex MINLP) model using: (i) CPLEX 12.7, (ii) Gurobi 7.5.2, (iii) Simulated Annealing (SA), and (iv) Particle Swarm Optimisation (PSO), with 1 hour timeout.

λ	CPLEX 12.7	Gurobi 7.5.2	PSO	SA
1	−14.2	−17.7	−88.7	**−91.3**
10	422.7	112.6	−86.0	**−86.6**
100	4,791.6	1,413.7	−80.1	**−80.3**
1000	48,480.6	14,425.1	**−75.9**	−71.6

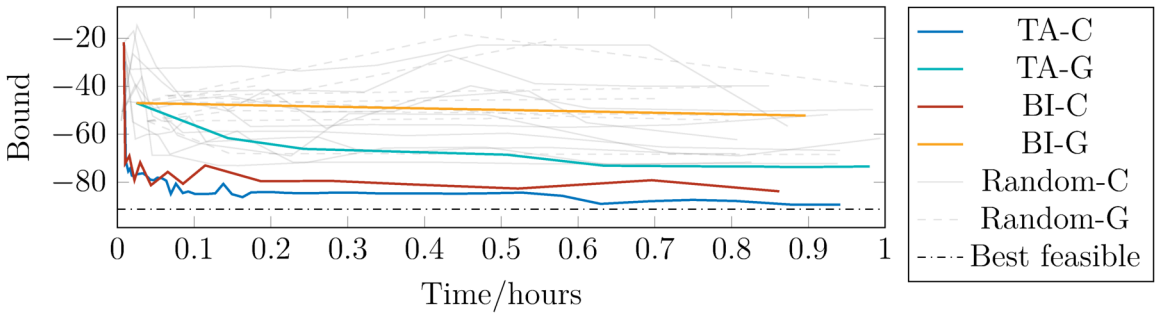


Figure 4.3: Concrete mixture design instance ($\lambda = 1$): Convex MINLP heuristic using training-aware (TA), best improvement (BI), or random strategies for choosing the next trees. Each iteration selects 10 new trees. The suffixes -C and -G denote using CPLEX 12.7 and Gurobi 7.5.2 as subsolvers, respectively. Best feasible is the simulated annealing solution.

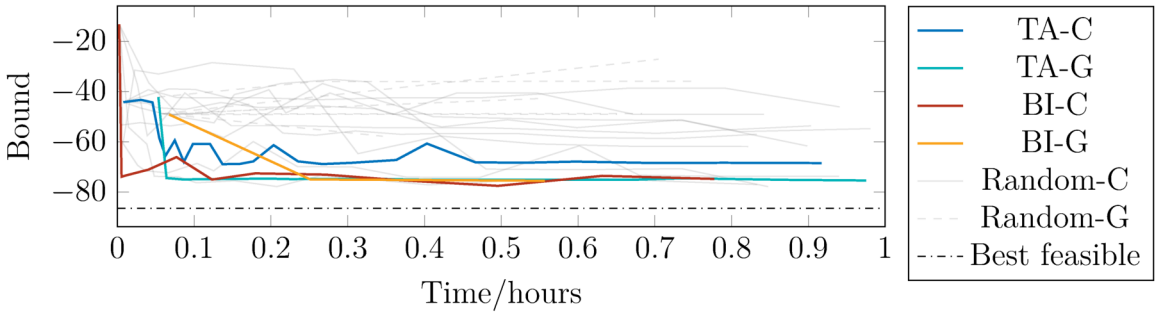


Figure 4.4: Concrete mixture design instance ($\lambda = 10$): Convex MINLP heuristic using training-aware (TA), best improvement (BI), or random strategies for choosing the next trees. Each iteration selects 10 new trees. The suffixes -C and -G denote using CPLEX 12.7 and Gurobi 7.5.2 as subsolvers, respectively. Best feasible is the simulated annealing solution.

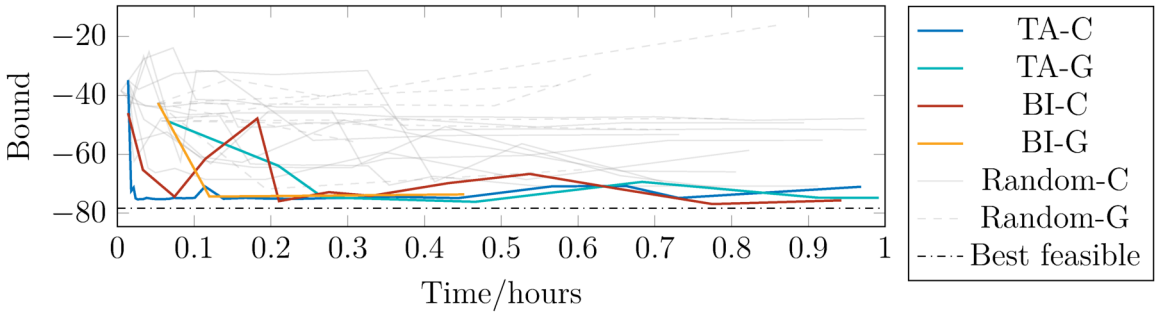


Figure 4.5: Concrete mixture design instance ($\lambda = 100$): Convex MINLP heuristic using training-aware (TA), best improvement (BI), or random strategies for choosing the next trees. Each iteration selects 10 new trees. The suffixes -C and -G denote using CPLEX 12.7 and Gurobi 7.5.2 as subsolvers, respectively. Best feasible is the simulated annealing solution.

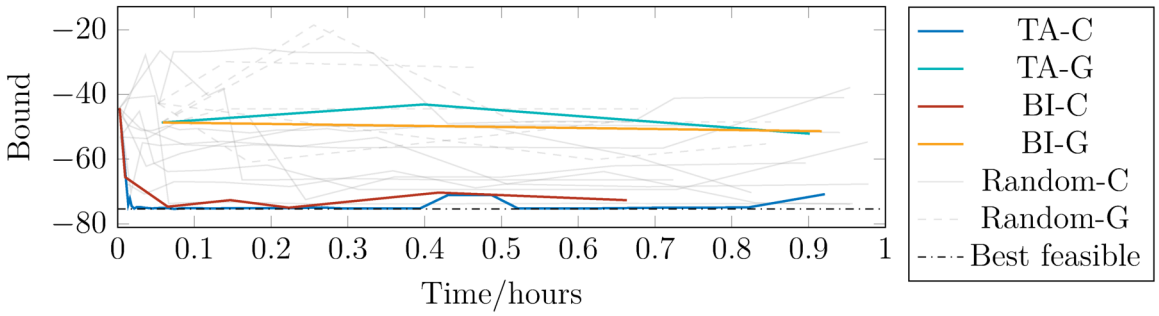


Figure 4.6: Concrete mixture design instance ($\lambda = 1000$): Convex MINLP heuristic using training-aware (TA), best improvement (BI), or random strategies for choosing the next trees. Each iteration selects 10 new trees. The suffixes -C and -G denote using CPLEX 12.7 and Gurobi 7.5.2 as subsolvers, respectively. Best feasible is the simulated annealing solution.

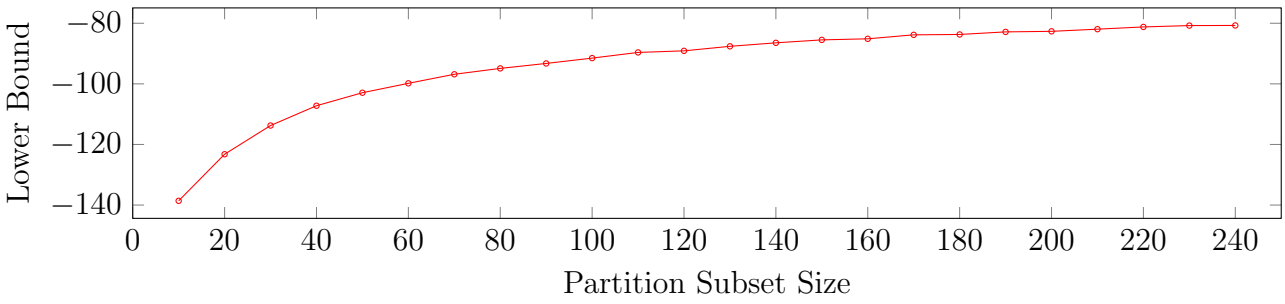


Figure 4.7: Concrete mixture design instance: Global GBT lower bound improvement using the Section 4.1.2 GBT lower bounding approach for different partition subset sizes.

close to the SA best found solution. CPLEX 12.7 reports poor heuristic solutions because it encounters out-of-memory errors prior to completing its root node relaxation solve. Gurobi 7.5.2 reports poor heuristic solutions because it does not complete the root node relaxation solve within an hour. The presolve of both solvers removes at most 43 constraints and 43 variables, hence the root node instance remains close in size to the original instance, i.e. the instance is still fairly large after presolve. Gurobi 7.5.2 performance can improve given more time as shown by the 24 hour time limit results in Table 4.3, the solver completes the root node relaxation solve in this time. Figures 4.3 to 4.6 evaluate the Section 4.1.4 augmenting convex MINLP heuristic using CPLEX 12.7, Gurobi 7.5.2, and the different tree selection approaches, i.e. (i) training-aware (TA), (ii) best improvement (BI), and (iii) random selection, for $\lambda \in \{1, 10, 100, 1000\}$, respectively. Figures 4.3 to 4.6 also plot the SA best-found solution. In general, both TA and BI perform better than random selection. Moreover, TA performs better than BI. Therefore, there is a benefit in choosing the earlier trees to find good heuristic solutions. Interestingly, the solution found in the first iteration of the augmenting convex MINLP heuristic, i.e. by solely minimising the convex part, is lower than -43, while the upper bounds reported by CPLEX 12.7 and Gurobi 7.5.2 after one hour of execution are greater than -18.

GBT Lower Bounding

Figures 4.7 and 4.8 evaluate the Section 4.1.2 GBT lower bounding approach for different partition subset sizes. Figure 4.7 illustrates the global GBT lower bound improvement as the partition subset size increases. Figure 4.8 compares run times with either CPLEX 12.7, or

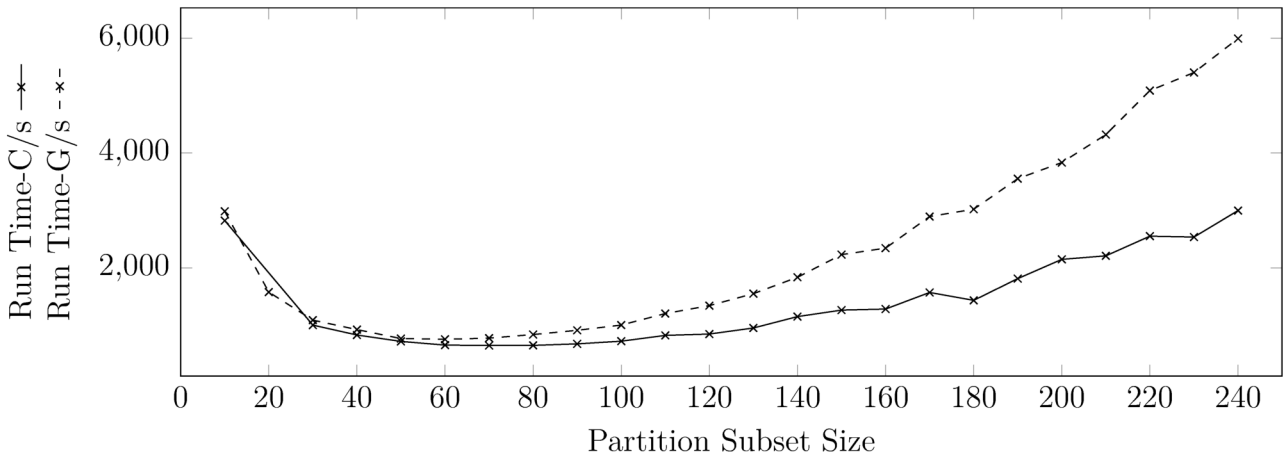


Figure 4.8: Concrete mixture design instance: Global GBT lower bounding wall clock time using the Section 4.1.2 approach for different partition subset sizes. Suffixes -C and -G denote subsolvers CPLEX 12.7 and Gurobi 7.5.2, respectively.

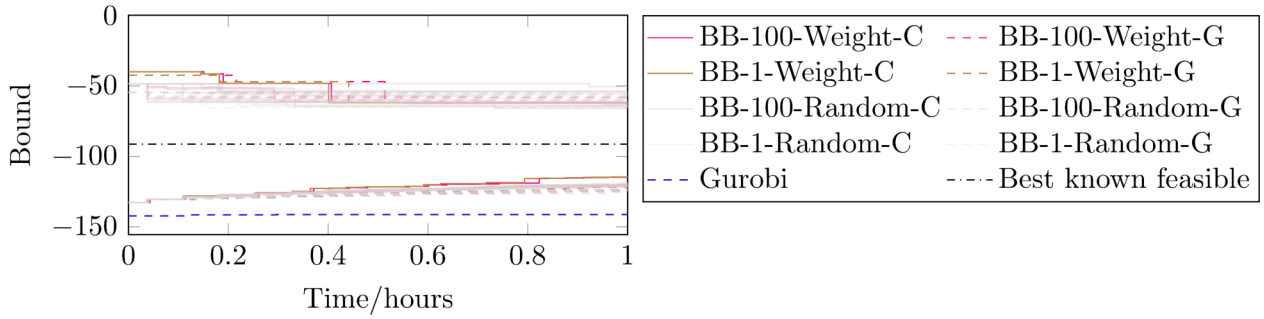


Figure 4.9: Concrete mixture design instance ($\lambda = 1$): B&B lower bound improvement compared to Gurobi 7.5.2, with a one hour timeout. The B&B Algorithm 5 is labeled BB-*a*-*b*-*c* where a, b and c denote the strong branching lookahead value, the pseudocost initialisation approach, and the solver used for lower bounding and solving convex quadratics, respectively. The BB-$*$ results sort the unexplored nodes in ascending lower bound order. The dashed-dotted line reports best found feasible solution (upper bound).

Gurobi 7.5.2 as subsolvers for each partition subset size. For the entire MILP instance, i.e. solving Problem (3.3), black-box solving with CPLEX 12.7 and Gurobi 7.5.2 achieve GBT lower bounds -97 and -547, respectively, within 1 hour. The Section 4.1.2 approach achieves a lower bound of -83 (partition size 190), in 1 hour, and improves upon black-box solver lower bounds in under 15 minutes (partition size 70).

Branch-and-Bound Algorithm

We instantiate the branch-and-bound algorithm with a root node partition of 70 trees, and non-root lower bounding time limit of 120 seconds. All branch-and-bound tests are run with

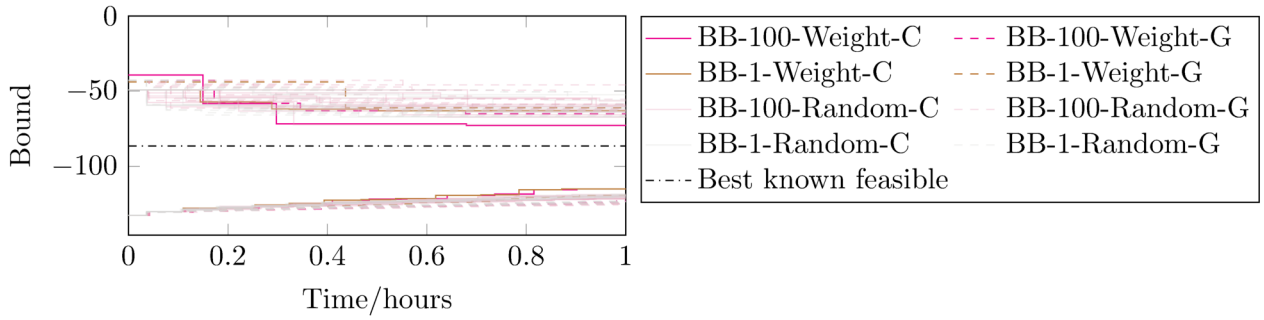


Figure 4.10: Concrete mixture design instance ($\lambda = 10$): B&B lower bound improvement compared to Gurobi 7.5.2, with a one hour timeout. The B&B Algorithm 5 is labeled BB-*a*-*b*-*c* where a, b and c denote the strong branching lookahead value, the pseudocost initialisation approach, and the solver used for lower bounding and solving convex quadratics, respectively. The BB-$*$ results sort the unexplored nodes in ascending lower bound order. The dashed-dotted line reports best found feasible solution (upper bound).

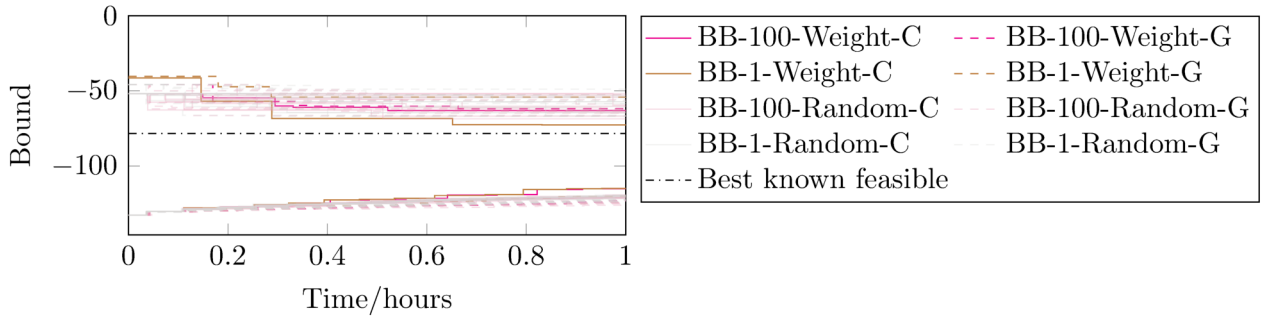


Figure 4.11: Concrete mixture design instance ($\lambda = 100$): B&B lower bound improvement compared to Gurobi 7.5.2, with a one hour timeout. The B&B Algorithm 5 is labeled BB-*a*-*b*-*c* where a, b and c denote the strong branching lookahead value, the pseudocost initialisation approach, and the solver used for lower bounding and solving convex quadratics, respectively. The BB-$*$ results sort the unexplored nodes in ascending lower bound order. The dashed-dotted line reports best found feasible solution (upper bound).

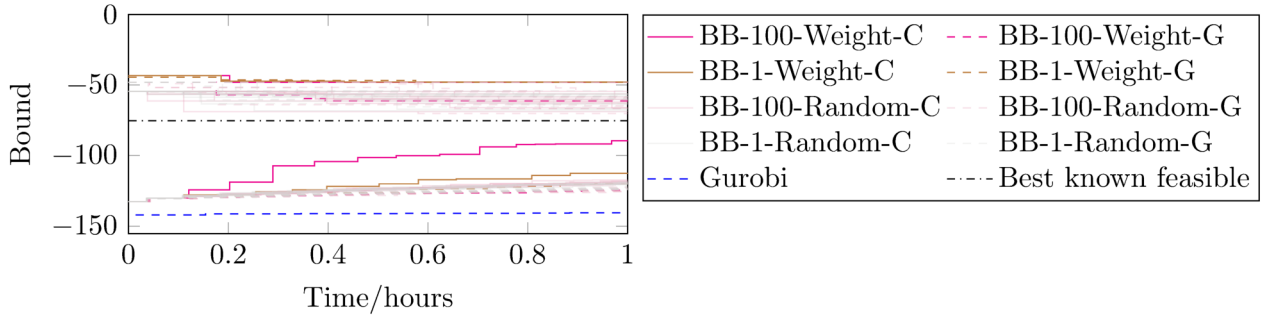


Figure 4.12: Concrete mixture design instance ($\lambda = 1000$): B&B lower bound improvement compared to Gurobi 7.5.2, with a one hour timeout. The B&B Algorithm 5 is labeled BB-*a*-*b*-*c* where a, b and c denote the strong branching lookahead value, the pseudocost initialisation approach, and the solver used for lower bounding and solving convex quadratics, respectively. The BB-$*$ results sort the unexplored nodes in ascending lower bound order. The dashed-dotted line reports best found feasible solution (upper bound).

CPLEX 12.7 and Gurobi 7.5.2 as MILP subsolvers. We assess the effect of strong branching by comparing lookahead list sizes $l = 1$ vs. $l = 100$. We assess the quality of feasible solutions by comparing with the Table 4.2 [electronic companion] best found feasible solution. We assess the pseudocost ordering by comparing with 10 independent tests of random branch orderings for each strong branch lookahead-subsolver combination. We compare all branch-and-bound results, which allocate 1 hour for GBT lower bounding at the root node and 1 hour for the B&B search, to 3 hour black-box runs of CPLEX 12.7 and Gurobi 7.5.2 for the entire convex MINLP.

Figures 4.9 to 4.12 plot the bound improvement for $\lambda \in \{1, 10, 100, 1000\}$, respectively. For the entire convex MINLP, the black-box CPLEX 12.7 bounds are outside the figure axis limits. For smaller λ, a larger strong branching lookahead value does not noticeably improve the lower bound, but a larger lookahead does significantly improve the lower bound for large λ, e.g. $\lambda = 1000$. Figures 4.9 to 4.12 depict lower bound improvement. The B&B algorithm lower bound improves over time, but there is still a non-negligible gap from the best-known feasible solution after 1 hour. This gap appears to be due to a cluster-like effect caused by the GBTs (Du and Kearfott, 1994; Wechsung et al., 2014; Kannan and Barton, 2017), where the variable split points are quite close. In the B&B algorithm, if the current lookahead list contains these clusters, strong branching is less effective. CPLEX 12.7 results in an out-of-memory error prior to beginning the branch-and-bound search therefore its lower bounds are relatively poor. Gurobi 7.5.2 returns an incumbent of -85 and a lower bound of -141, after 2 hours, and these do not improve further in the subsequent hour. The B&B algorithm, at 2 hours, i.e. prior to tree search, has an incumbent of -91 and a lower bound not less than -133. Given an additional hour for tree search, the gap reduces further. Table 4.3 compares the B&B algorithm to Gurobi 7.5.2 with 24 hours time limit. The Gurobi heuristics generally outperform the B&B algorithm, but the B&B algorithm derives better lower bounds. In all cases, $\geq 22\%$ optimality gap remains. Because regions close to training data have many GBT breakpoints, optimal solutions lie in highly discretised areas of the feasibility domain.

Table 4.3: Concrete mixture design instance: Results comparing 24 hour runs of the B&B algorithm with Gurobi 7.5.2. The B&B algorithm uses a strong branching lookahead value of 100, a root node partition of 70 trees, a non-root lower bounding time limit of 120 seconds, and CPLEX 12.7 as a subsolver.

	BB-C			Gurobi 7.5.2		
λ	UB	LB	Gap	UB	LB	Gap
1	−80.56	−99.87	24%	−85.48	−140.75	64%
10	−74.96	−99.39	33%	−85.06	−121.10	42%
100	−73.74	−96.43	31%	−77.98	−121.27	55%
1000	−74.86	−90.75	22%	−72.29	−121.23	67%

4.3.3 Chemical Catalysis

BASF uses catalysts to improve yield and operating efficiency. But, modelling catalyst effectiveness is highly nonlinear and varies across different applications. BASF has found GBTs effective for modelling catalyst behaviour. Capturing the high-dimensional nature of catalysis over the entire feasible domain requires many experiments, too many to run in practice. Running a fewer number of experiments necessitates penalising solutions further from where the GBT function is trained.

Instance

The BASF industrial instance contains $n = 42$ continuous variables. The convex part of the instance takes the following form:

$$\text{cvx}_\lambda(\boldsymbol{x}) = \lambda \left\| (\boldsymbol{I} - \boldsymbol{P}) \operatorname{diag}(\boldsymbol{\sigma})^{-1} (\boldsymbol{x} - \boldsymbol{\mu}) \right\|_2^2 + \left(100 - \sum_{i \in \mathcal{I}^{\%}} x_i \right)^2 \tag{4.4}$$

Equation (4.4) differs from Equation (4.3) in its addend which aims to generate solutions where $x_i \in \mathcal{I}^{\%}$, i.e. proportions of the chemicals being mixed, sum to 100%. The test instance has $\text{rank}(\boldsymbol{P}) = 2$ and $|\mathcal{I}^{\%}| = 37$. The GBT part contains 8,800 trees where 4,100 trees have max depth 16, the remaining trees have max depth 4, the total number of leaves is 93,200 and the corresponding Problem (3.3) MILP model has 2,061 binary variables (recall that only the $y_{i,j}$ variables are binary since integrality constraints on $z_{t,l}$ leaf selecting variables can be dropped

Table 4.4: Chemical catalysis BASF instance (with different λ values): Black-box solver solutions (upper bounds) by solving the entire mixed-integer convex programming (convex MINLP) model using: (i) CPLEX 12.7, (ii) Gurobi 7.5.2, (iii) Simulated Annealing (SA), and (iv) Particle Swarm Optimisation (PSO), with 1 hour timeout.

λ	CPLEX 12.7	Gurobi 7.5.2	PSO	SA
0	*	−158.5	−96.8	**−168.2**
1	*	−101.6	−89.8	**−130.7**
10	952	−100.1	−97.6	**−102.7**
100	1,040	11.5	−82.7	**−84.2**
1000	18,579	606.5	−76.5	**−81.3**

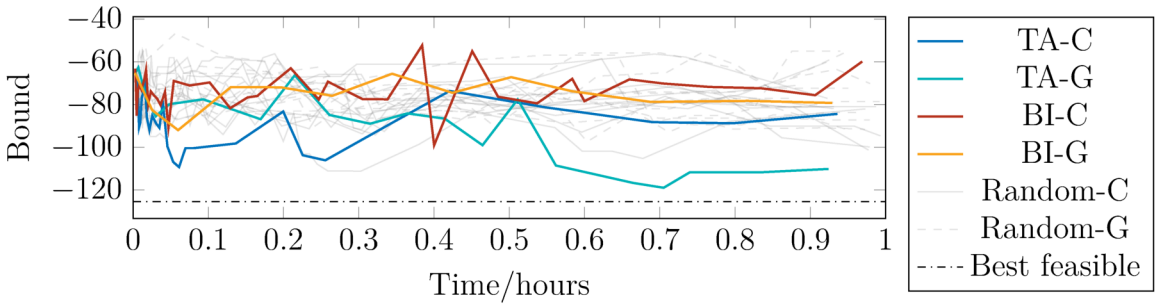


Figure 4.13: Chemical catalysis instance ($\lambda = 1$): Convex MINLP heuristic using training-aware (TA), best improvement (BI), or random strategies for choosing the next trees. Each iteration selects 10 new trees. The suffixes -C and -G denote using CPLEX 12.7 and Gurobi 7.5.2 as subsolvers, respectively. Best feasible is the simulated annealing solution.

without affecting correctness of the MILP model).

Heuristic Solutions

Table 4.4 compares the CPLEX 12.7, Gurobi 7.5.2, SA, and PSO computed solutions for the entire covex MINLP, under a 1 hour time limit. SA outperforms all others. PSO performs well for larger λ values, because it keeps the contribution of the convex part low at initialisation. Gurobi 7.5.2 also performs relatively well for smaller λ values, however due to solver tolerances it may report incorrect objective values. For example, using $\lambda = 0$ the solver reports an objective of -174.1, however a manual evaluation results in -158.5. In fact, both CPLEX 12.7 or Gurobi 7.5.2, may produce incorrect outputs due to solver tolerances, hence a specialised fixing method may be necessary.

Figures 4.13 to 4.16 evaluate the Section 4.1.4 augmenting convex MINLP heuristic for different

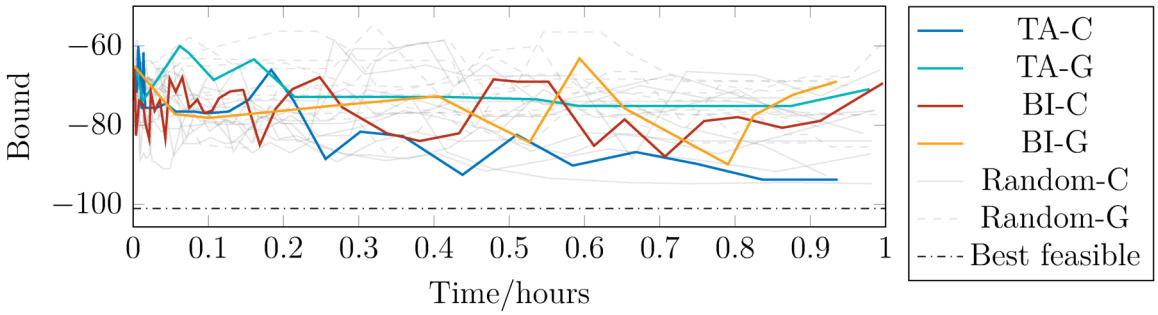


Figure 4.14: Chemical catalysis instance ($\lambda = 10$): Convex MINLP heuristic using training-aware (TA), best improvement (BI), or random strategies for choosing the next trees. Each iteration selects 10 new trees. The suffixes -C and -G denote using CPLEX 12.7 and Gurobi 7.5.2 as subsolvers, respectively. Best feasible is the simulated annealing solution.

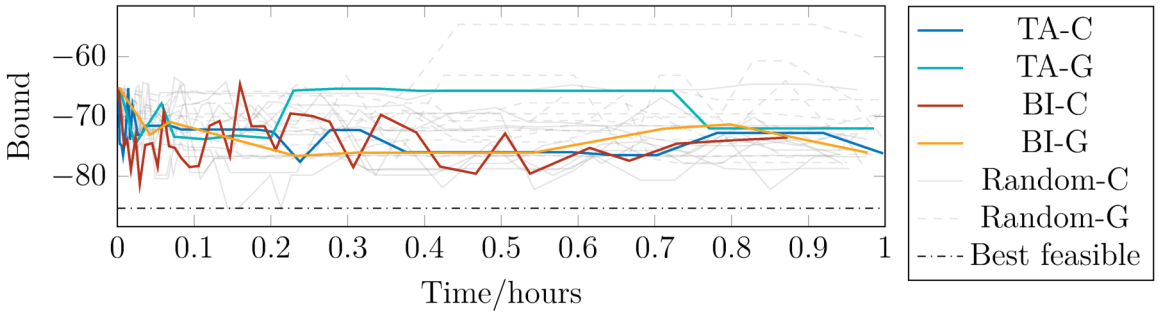


Figure 4.15: Chemical catalysis instance ($\lambda = 100$): Convex MINLP heuristic using training-aware (TA), best improvement (BI), or random strategies for choosing the next trees. Each iteration selects 10 new trees. The suffixes -C and -G denote using CPLEX 12.7 and Gurobi 7.5.2 as subsolvers, respectively. Best feasible is the simulated annealing solution.

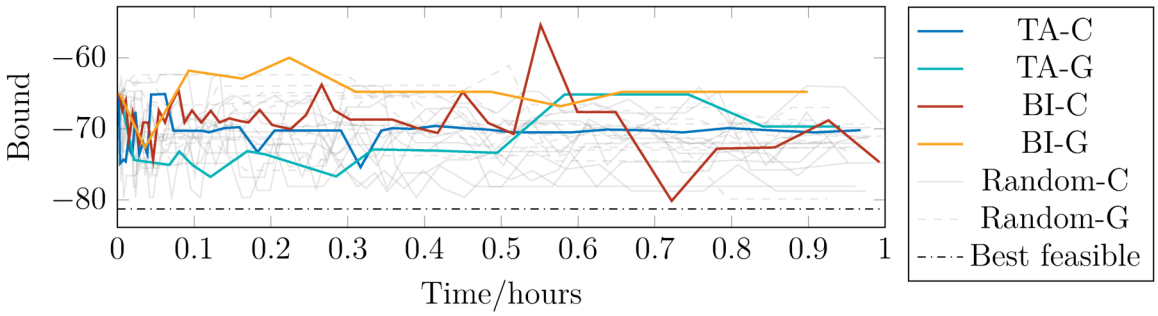


Figure 4.16: Chemical catalysis instance ($\lambda = 1000$): Convex MINLP heuristic using training-aware (TA), best improvement (BI), or random strategies for choosing the next trees. Each iteration selects 10 new trees. The suffixes -C and -G denote using CPLEX 12.7 and Gurobi 7.5.2 as subsolvers, respectively. Best feasible is the simulated annealing solution.

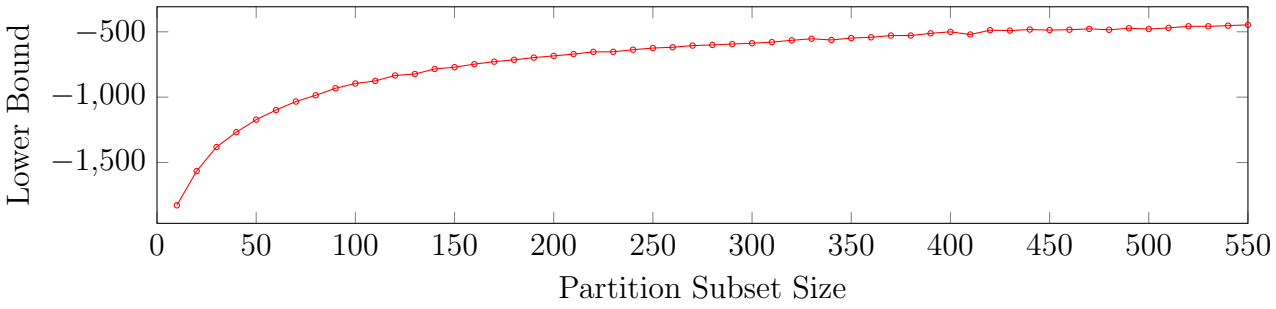


Figure 4.17: Chemical catalysis BASF instance: Global GBT lower bound improvement using the Section 4.1.2 GBT lower bounding approach for different partition subset sizes.

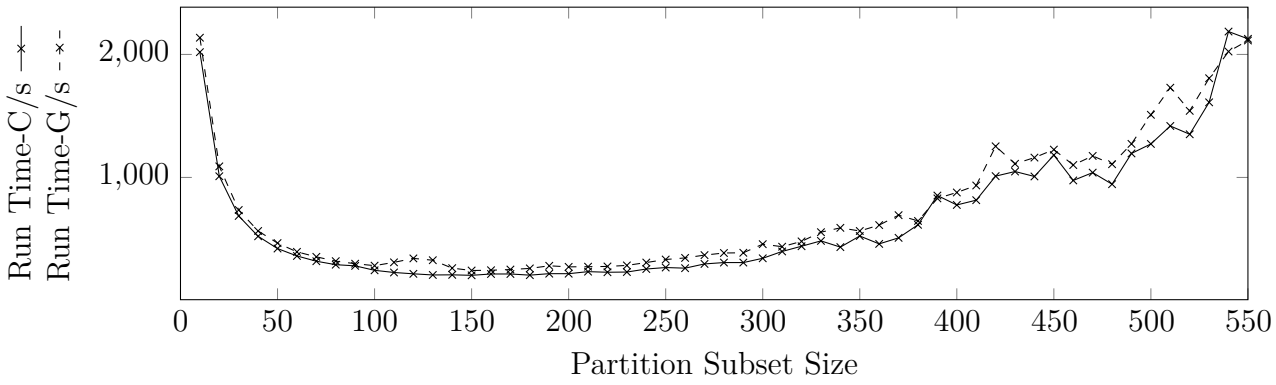


Figure 4.18: Chemical catalysis BASF instance: Global GBT lower bounding wall clock time using the Section 4.1.2 approach for different partition subset sizes. Suffixes -C and -G denote subsolvers CPLEX 12.7 and Gurobi 7.5.2, respectively.

values of the λ input parameter. We investigate the augmenting convex MINLP heuristic performance using either CPLEX 12.7, or Gurobi 7.5.2 for solving convex MINLP sub-instances and each of the: (i) training-aware (TA), (ii) best improvement (BI), and (iii) random selection strategies. The Figures 4.13 and 4.16 best feasible solution is the one produced by SA. For $\lambda = 1$, TA constructs several heuristic solutions that outperform both the BI and random selection ones. In this case, since the GBT part dominates the convex part, TA iteratively computes a better GBT approximation. As λ becomes larger, TA and BI exhibit comparable performance, with BI finding the best solution for $\lambda = 1000$. Random selection also performs well because the convex part dominates the GBT part.

GBT Lower Bounding

Figures 4.17 and 4.18 evaluate the Section 4.1.2 GBT lower bounding approach for different partition subset sizes. Figure 4.17 illustrates the global GBT lower bound improvement as the

partition subset size increases. Figure 4.18 compares run times when using either CPLEX 12.7, or Gurobi 7.5.2 as subsolvers for each partition subset size. These results resemble Figures 4.7 and 4.8. In particular, (i) the lower bound is improved with larger subset sizes, (ii) there is a time-consuming modelling overhead for solving many small MILPs for small subset sizes, and (iii) the running time increases exponentially, though non-monotonically, for larger subset sizes. We compare the lower bounding approach with solving the entire MILP (3.3) using CPLEX 12.7, or Gurobi 7.5.2 as black-box solvers. Our lower bounding approach exhibits a superior time-to-lower bound performance: (i) it improves the Gurobi 7.5.2 lower bound with subset size 140 and 4 minutes of execution, and (ii) it improves the CPLEX 12.7 lower bound with subset size 360 and 8 minutes of execution.

Branch-and-Bound Algorithm

We instantiate the branch-and-bound algorithm with a root node partition of 150 trees, and non-root lower bounding time limit of 120 seconds. All branch-and-bound tests are run with CPLEX 12.7 and Gurobi 7.5.2 as subsolvers. We assess the effect of strong branching by comparing lookahead list sizes $l = 1$ vs. $l = 100$. We assess the quality of feasible solutions by comparing with the Table 4.4 [electronic companion] best found feasible solution. We assess the pseudocost ordering by comparing with 10 independent tests of random branch orderings for each strong branch lookahead-subsolver combination. We compare all branch-and-bound results, which allocate 1 hour for GBT lower bounding at the root node and 1 hour for the B&B search, to 3 hour black-box runs of CPLEX 12.7 and Gurobi 7.5.2 for the entire convex MINLP.

Figures 4.19 to 4.22 plot the bound improvement for $\lambda \in \{1, 10, 100, 1000\}$, respectively. For the entire convex MINLP, CPLEX 12.7 reports a poor lower bound and does not find a feasible solution within 3 hours. The B&B algorithm terminates with a tighter lower bound and closes a larger gap than the black-box solvers, across all tested parameter combinations. The B&B algorithm performs better for $\lambda = 1000$ because the convex part dominates the GBT part more, making strong branching more effective. Finally, we see that the branch-and-bound algorithm finds a relatively good heuristic solution at the root node for $\lambda = 1000$. As λ becomes

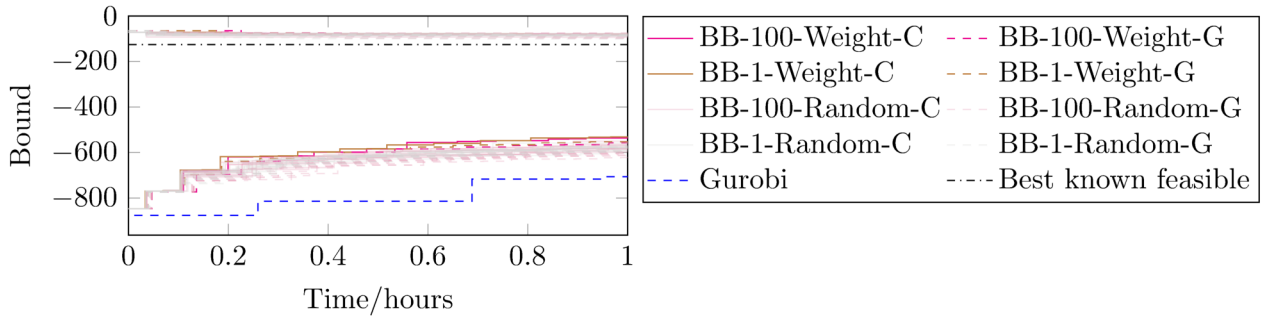


Figure 4.19: Chemical catalysis BASF instance ($\lambda = 1$): B&B lower bound improvement compared to Gurobi 7.5.2 with one hour timeout. The B&B Algorithm 5 is labeled BB-*a*-*b*-*c* where a, b and c denote the strong branching lookahead value, the pseudocost initialisation approach, and the solver used for lower bounding and solving convex quadratics, respectively. The BB-$*$ results sort the unexplored nodes in ascending lower bound order. The dashed-dotted line reports best found feasible solution (upper bound).

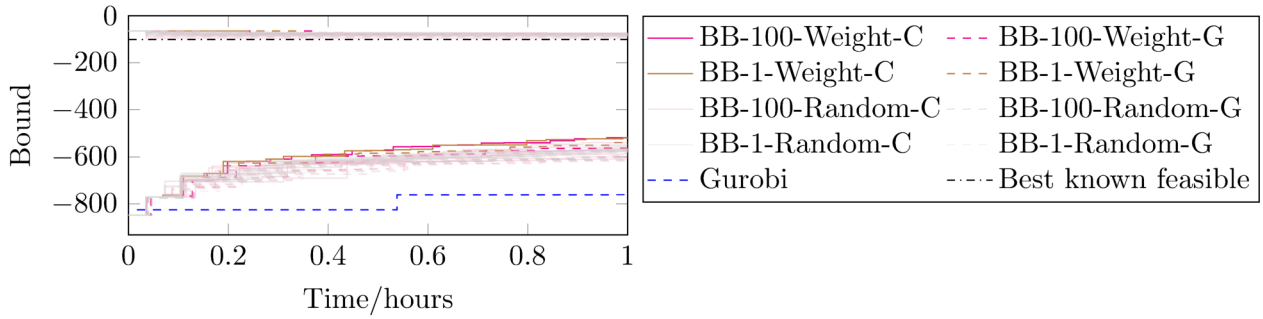


Figure 4.20: Chemical catalysis BASF instance ($\lambda = 10$): B&B lower bound improvement compared to Gurobi 7.5.2 with one hour timeout. The B&B Algorithm 5 is labeled BB-*a*-*b*-*c* where a, b and c denote the strong branching lookahead value, the pseudocost initialisation approach, and the solver used for lower bounding and solving convex quadratics, respectively. The BB-$*$ results sort the unexplored nodes in ascending lower bound order. The dashed-dotted line reports best found feasible solution (upper bound).

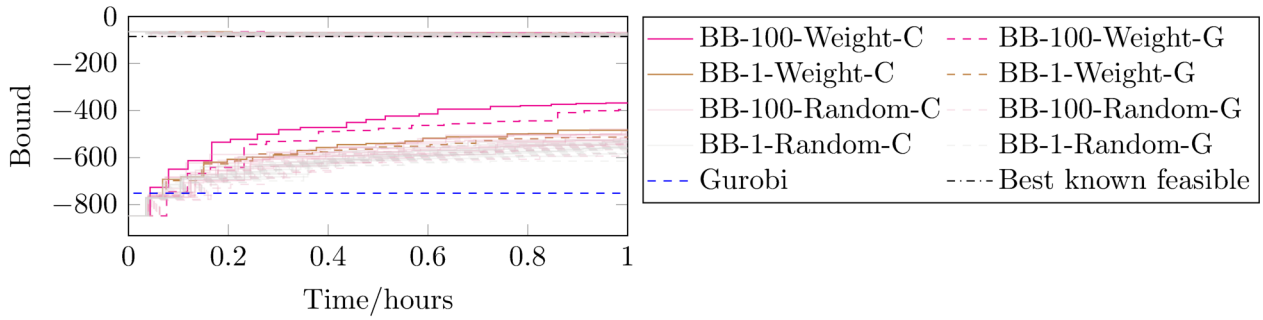


Figure 4.21: Chemical catalysis BASF instance ($\lambda = 100$): B&B lower bound improvement compared to Gurobi 7.5.2 with one hour timeout. The B&B Algorithm 5 is labeled BB-*a*-*b*-*c* where a, b and c denote the strong branching lookahead value, the pseudocost initialisation approach, and the solver used for lower bounding and solving convex quadratics, respectively. The BB-$*$ results sort the unexplored nodes in ascending lower bound order. The dashed-dotted line reports best found feasible solution (upper bound).

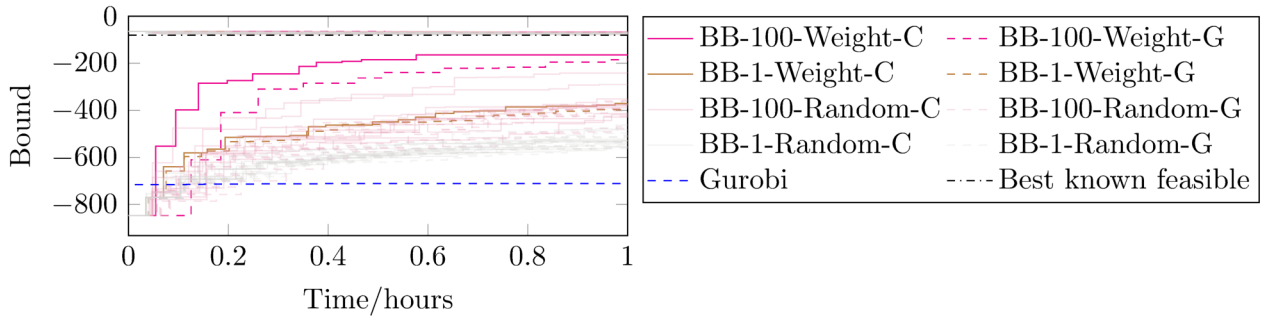


Figure 4.22: Chemical catalysis BASF instance ($\lambda = 1000$): B&B lower bound improvement compared to Gurobi 7.5.2 with one hour timeout. The B&B Algorithm 5 is labeled BB-*a*-*b*-*c* where a, b and c denote the strong branching lookahead value, the pseudocost initialisation approach, and the solver used for lower bounding and solving convex quadratics, respectively. The BB-$*$ results sort the unexplored nodes in ascending lower bound order. The dashed-dotted line reports best found feasible solution (upper bound).

Table 4.5: Chemical catalysis instance: Results comparing 24 hour runs of the B&B algorithm with Gurobi 7.5.2. The B&B algorithm uses a strong branching lookahead value of 100, a root node partition of 150 trees, a non-root lower bounding time limit of 120 seconds, and CPLEX 12.7 as a subsolver.

	BB-C			Gurobi 7.5.2		
λ	Upper Bound	Lower Bound	Gap	Upper Bound	Lower Bound	Gap
1	−81.7	−366.0	348%	−154.8	−580.6	275%
10	−80.6	−336.8	318%	−118.8	−577.1	386%
100	−87.3	−187.0	114%	−94.2	−424.5	350%
1000	−86.0	−86.0	0%	−85.9	−92.1	7%

smaller, there is a larger gap between the B&B upper bounds and the best known feasible solution. This is expected as solving this problem is closer to optimising only over the GBT MILP where an optimal solution may be further from the PCA subspace. Table 4.5 compares the B&B algorithm to Gurobi 7.5.2 with 24 hours time limit. The Gurobi heuristic solutions generally outperform the B&B algorithm. Nevertheless, the B&B algorithm derives better lower bounds. For $\lambda = 1000$, the B&B algorithm succeeds in proving global optimality whereas Gurobi terminates with a 7% gap.

4.3.4 Observations

The Sections 4.3.2 and 4.3.3 heuristic results show that, for the tested instances, SA performs well and that PSO can find good solutions for larger λ values. SA and PSO benefit from only

involving continuous variables $\boldsymbol{x}$ whose entire box domain $[\boldsymbol{L}, \boldsymbol{U}]$ is feasible. Hence, SA and PSO are able to make many (valid) function evaluations. However, both SA and PSO are non-deterministic methods. Mixed-integer-based methods are a deterministic approach. While SA and PSO function evaluations are always feasible, mixed-integer methods can encounter infeasibility due to the formulation, since arbitrary assignments on modelling variables $\boldsymbol{y}$ and $\boldsymbol{z}$ are not necessarily feasible, these infeasibilities can hinder the search. Using mixed-integer solvers for a fixed length of time may return heuristic solutions. Applying 1 hour timeouts for a mixed-integer solve on the entire instance shows that the solvers can perform relatively well for smaller λ values, e.g. $\lambda \in \{0, 10, 100\}$ in the chemical catalysis instance. Instances where λ is smaller are less influenced by the convex part of the objective function therefore these instances are more similar to solving the GBT MILP instance. Hence, MILP techniques may be more effective in finding heuristic solutions. For larger λ values, using a mixed-integer solver as a heuristic performs less well. Increased dominance from the convex part of the objective may hinder the solver when it applies mixed-integer techniques, which are more applicable to the GBT part of the problem. For our Section 4.1.4 iterative convex MINLP-based heuristics, we see that our selection strategies generally outperform random selection.Both of the proposed selection strategies perform comparably. In the case of training-aware selection, each successive iteration leverages how the GBT function is trained, i.e. consecutive iterations consider a sequence of GBTs that aim to rectify the errors introduced by the previous trees. The best improvement selection strategy performs relatively well when considering a larger λ. Since the convex MINLP heuristic considers larger instances, with respect to number of binary variables and constraints, successive iterations generally see an increasing trend in solve time.

The Sections 4.3.2 and 4.3.3 GBT lower bounding results show that, for large-scale GBT instances, selecting an appropriate partition subset size in the decomposition approach results has a better time-to-lower bound performance than 1 hour black-box MILP solvers. Both problem instances show that, for larger subset sizes, the running time exponentially increases, while the lower bound improvement rate exponentially decreases. This is an expected result for GBT instances with deep trees as deeper tree induce more infeasible combinations of branches. For shallower GBT instances, individual trees to may interact less with each other, hence the

decomposition strategy may derive a poorer bound than a black-box MILP solver. For small subset sizes, the partition-based lower bounding has decreasing running time because of the overhead from many sequential subproblems.

The Sections 4.3.2 and 4.3.3 B&B results also show common features. Comparing the BB-$*$-b-$*$ results for $b \in \{\text{Weight}, \text{Random}\}$ in Figures 4.9 to 4.12 and 4.19 to 4.22 assess the pseudocost effect. The Equation (4.2) initialisation outperforms random ordering (for matching lookahead values), showing that the pseudocosts select branches that aid GBT lower bounding. This pseudocost effect is more pronounced with a lookahead value of 100 since multiple branches are selected between branch-and-bound iterations. For $\lambda = 1000$, a lookahead list size $l = 100$ closes more gap than $l = 1$ (comparing BB-100-$*$ to BB-1-$*$), as the B&B algorithm accepts more branches for strong branching. The difference between $l = 100$ and $l = 1$ implies that increased strong branching improves the GBT lower bound earlier and more often. For $\lambda = 1$, using a larger strong branching lookahead size does not have a noticable effect. However, this last finding does not depreciate strong branching. Since the GBT part dominates the convex aspect for small λ values, tighter GBT lower bounds might be essential for taking full advantage of strong branching. Testing the B&B algorithm and Gurobi 7.5.2 with a 24 hour run time shows that the branch-and-bound algorithm tends to result in superior lower bounds and closes a larger proportion of the optimality gap whereas Gurobi 7.5.2 produces better heuristic solutions. Closing any outstanding gap proves difficult as the domains of the remaining unexplored nodes are highly discretised by the GBTs.

Chapter 5

Using Satisfiability Modulo Theories Derived Unsatisfiable Cores in Mathematical Optimisation

Both logic and optimisation has been long recognised as important for automated decision-making (Jain and Grossmann, 2001; Hooker and Ottoson, 2003; Maravelias and Sung, 2009; Trespalacios and Grossmann, 2014). Early work on disjunctive programming is motivated by (i) the practical need to naturally model logical conditions such as dichotomies and implications and (ii) the theoretical insights gained from novel structural characterisations (Balas, 1979). Contributions highlighting the importance of both logic and optimisation have diverse applications, e.g. spatial layout (Sawaya and Grossmann, 2005), modelling contracts in supply chain optimisation (Park et al., 2006; Rodriguez and Vecchietti, 2009), and manufacturing systems (Fattahi et al., 2014).

Problems that involve both logic and mathematical constraints are solved by enumerating the logical assignments in a branch-and-bound (B&B) framework (Land and Doig, 1960). B&B implementations often assume a mathematical programming formulation, consider continuous relaxations, and use continuous mathematical optimisation methods to derive lower bounds. Continuous relaxations have the effect of relaxing logical structure, hence lower bounds may be weaker and detecting infeasible logical assignments may require many branching steps. An

alternative to a continuous relaxation strategy is a logical relaxation strategy as considered by satisfiability modulo theories (SMT).

SMT modelling differs from mathematical programming modelling by supporting Boolean variables and connectives. SMT approaches constraint feasibility by relaxing to a propositional satisfiability logical relaxation, e.g. mathematical inequalities are replaced by fresh Boolean variables, and reasons feasibility subject to a logical relaxation assignment. SMT solvers derive unsatisfiable cores, a subset of constraints that explain infeasibility of a model. Unsatisfiable cores are similar to irreducible infeasible subsystems (IISs) (Chakravarti, 1994) in mathematical programming. However, as unsatisfiable cores operate over SMT constraints, an equivalent IIS may be larger in size, e.g. when an SMT constraint requires many mathematical programming constraints for an equivalent representation.

This chapter investigates how SMT can be utilised for optimisation by studying two-dimensional bin packing. We propose three algorithms for solving two-dimensional bin packing that break symmetries. In particular, one of these algorithms is a B&B approach that uses unsatisfiable cores to guide the branching and break symmetries arising from infeasibilities. Numerical results show that our B&B approach solves an additional 25% of the tested instances within a 1 hour timelimit when compared with commercial solvers CPLEX 12.7 and Gurobi 6.0.3.

Section 5.1 develops a branch-and-bound algorithm that solves the two-dimensional bin packing problem. Section 5.2 suggests how SMT unsatisfiable cores may be utilised as a tool to aid debugging branch-and-bound algorithms.

5.1 Solving Two-Dimensional Bin Packing with Satisfiability Modulo Theories

Two-dimensional bin packing (2BP) is the problem (Chung et al., 1982):

> Given n_C rectangular items. What is the minimum number of rectangular bins with

Table 5.1: Model symbols for the two-dimensional bin packing (2BP) problem.

Name	Description
Sets	
$i, j \in [n_C]\}$	Items
Parameters	
W, H	Width and height of the bins respectively
W_i, H_i	Width and height of item i respectively
Variables	
Y_b	Activity of bin b, Boolean
Y_{ib}	Item i assigned to bin b, Boolean
y_b	Activity of bin b, Binary
y_{ib}	Item i assigned to bin b, Binary
$x_{i,1}, x_{i,2}$	Lower left coordinate of item i
z_0	Largest active bin index
z_i	Bin containing item i
y^l_{ij}	Item i left of item j
y^b_{ij}	Item i below item j
y^p_{ij}	Item i in lower index bin than item j

width W and height H needed to pack all n_C items without overlapping or rotating the items?

This problem has typology class **2BP|O|F** (Lodi et al., 1999), i.e. oriented and free cutting. Being a generalisation of the $\mathcal{NP}$-hard (one-dimensional) bin packing problem, 2BP is also $\mathcal{NP}$-hard. See Table 5.1 for descriptions of the sets, parameters and variables mentioned in the formulations.

5.1.1 Models

We formulate two models for 2BP, a logical and a mixed-integer linear programming (MILP) model.

Logical Model

In 2BP a core constraint is: *two different items in the same bin should not overlap.* A logical formulation naturally captures this if-then relationship as the following model shows.

$$\min \sum_{b=1}^{n_C} y_b \tag{5.1a}$$

$$\text{s.t.} \bigvee_{b=1}^{n_C} \left(Y_{ib} \wedge \bigwedge_{b' \neq b} \neg Y_{ib'} \right) \qquad \forall i \in [n_C] \tag{5.1b}$$

$$(Y_{ib} \wedge Y_{jb}) \rightarrow \begin{cases} (x_{i,1} + W_i \leq x_{j,1}) \vee (x_{j,1} + W_j \leq x_{i,1}) \\ \vee (x_{i,2} + H_i \leq x_{j,2}) \vee (x_{j,2} + H_j \leq x_{i,2}) \end{cases} \qquad \forall b \in [n_C], i, j \in [n_C], i < j \tag{5.1c}$$

$$Y_{ib} \rightarrow Y_b \qquad \forall b \in [n_C], i \in [n_C] \tag{5.1d}$$

$$Y_b \rightarrow (y_b = 1) \qquad \forall b \in [n_C] \tag{5.1e}$$

$$\neg Y_b \rightarrow (y_b = 0) \qquad \forall b \in [n_C] \tag{5.1f}$$

$$0 \leq x_{i,1} \leq W - W_i, 0 \leq x_{i,2} \leq H - H_i \qquad \forall i \in [n_C] \tag{5.1g}$$

$$Y_b, Y_{ib} \in \{\text{True}, \text{False}\}, y_b \in \{0, 1\} \qquad \forall b \in [n_C], i \in [n_C]. \tag{5.1h}$$

In Problem (5.1), parameters W and H are the width and height of the bins, respectively, and W_i and H_i are the width and height of item $i \in [n_C]$, respectively. Each bin is modelled as a coordinate system where the lower left corner of the bin is $(0, 0)$. Continuous variables $(x_{i,1}, x_{i,2})$ model the position of the lower left corner of item $i \in [n_C]$. Boolean assignment $Y_{ib} = \text{True}$, $i, b \in [n_C]$, represents item i being assigned to bin b. Boolean assignment $Y_b = \text{True}$, $b \in [n_C]$, represents that bin b is active. Binary variables y_b are 0-1 equivalents of Y_b which form the Problem (5.1) objective. Equation (5.1a) minimises the number of active bins. Equation (5.1b) fixes each item into a single bin. Equation (5.1c) ensures that any two items in the same bin do not overlap. Specifically, the constraint states that if items i and j are assigned to bin b, then item i must be placed either to the [left, right, below, or above] of item j. Equation (5.1d) states that a bin must be active if it contains items. Equations (5.1e) and (5.1f) transform Boolean variables to binary variables to form the Equation (5.1a) objective.

MILP Model

The MILP formulation (Pisinger and Sigurd, 2007) is:

$$\min z_0 \tag{5.2a}$$

$$\text{s.t. } y^l_{ij} + y^l_{ji} + y^b_{ij} + y^b_{ji} + y^p_{ij} + y^p_{ji} \geq 1 \qquad \forall i, j \in [n_C], i < j \tag{5.2b}$$

$$x_{i,1} - x_{j,1} + W y^l_{ij} \leq W - W_i \qquad \forall i, j \in [n_C] \tag{5.2c}$$

$$x_{i,2} - x_{j,2} + H y^b_{ij} \leq H - H_i \qquad \forall i, j \in [n_C] \tag{5.2d}$$

$$z_i - z_j + n_C y^p_{ij} \leq n_C - 1 \qquad \forall i, j \in [n_C] \tag{5.2e}$$

$$1 \leq z_i \leq z_0 \qquad \forall i \in [n_C] \tag{5.2f}$$

$$z_i \leq i \qquad \forall i \in [n_C] \tag{5.2g}$$

$$0 \leq x_{i,1} \leq W - W_i, 0 \leq x_{i,2} \leq H - H_i \qquad \forall i \in [n_C] \tag{5.2h}$$

$$y^l_{ij}, y^b_{ij}, y^p_{ij} \in \{0, 1\} \qquad \forall i, j \in [n_C] \tag{5.2i}$$

$$z_i, z_0 \in \mathbb{Z} \qquad \forall i \in [n_C] \tag{5.2j}$$

Similar to Problem (5.1), Problem (5.2) parameters W, H are the width and height of the bins, and W_i and H_i, are the width and height of item $i \in [n_C]$. Binary assignment $y^l_{ij} = 1$ ($y^l_{ji} = 1$) represents that item i is left (right) of item j. Binary assignment $y^b_{ij} = 1$ ($y^b_{ji} = 1$) represents that item i is below (above) item j. Binary assignment $y^p_{ij} = 1$ ($y^p_{ji} = 1$) represents that the bin index to which item i is assigned is strictly less (greater) than the bin index to which item j is assigned. Integer variable z_i, $i \in [n_C]$ identifies which bin contains item i and z_0 counts the number of active bins. Equation (5.2a) minimises the number of active bins. Equation (5.2b) states that items are in different bins or that they do not overlap. Equations (5.2c) and (5.2d) are big-M constraints that characterise non-overlapping items. Equation (5.2e) characterises items being placed in different bins. Equation (5.2f) assigns the maximal active bin to z_0 (the objective ensures that the constraint is active at optimality). Equation (5.2g) is a symmetry breaking constraint. The Equations (5.2c) to (5.2e) big-M constraints can result in a loose continuous relaxation and therefore yield looser lower bounds. A natural lower bound on the

optimal number of bins is the Equation (5.3) lower bound L_0 (Martello and Vigo, 1998):

$$L_0 = \left\lceil \frac{\sum_{i=1}^{n_C} W_i \cdot H_i}{W \cdot H} \right\rceil, \tag{5.3}$$

i.e. the total area given by the active bins needs to be at least the total area of the items. Equation (5.3) is known as the continuous lower bound, we refer to it as L_0 to prevent confusion with the continuous relaxation bound. Example 5.1 shows that the continuous relaxation bound of Problem (5.2) can be less than L_0.

Example 5.1. *Consider a 2BP instance with 5 items all having width and height equal to 1 and all bins having width and height equal to 2. The Equation* (5.3) *bound for this instance is* $L_0 = \left\lceil \frac{5}{4} \right\rceil = 2$. *For this instance,* L_0 *is tight. For the continuous relaxation of Problem (5.2), consider the assignments* $y^p_{ij} = 0$, $y^l_{ij} = \frac{1}{4}$, $y^b_{ij} = \frac{1}{4}$, *for all* $i, j \in \{1, \ldots, 5\}$ *and* $z_0 = z_i = 1$, *for all* $i \in \{1, \ldots, 5\}$. *The only remaining unassigned variables are* $x_{i,1}, x_{i,2}$, *for all* $i \in \{1, \ldots, 5\}$. *Substituting existing variable assignments and parameter values into Equation* (5.2c) *gives:*

$$x_{i,1} \leq x_{j,1} + 0.5.$$

Assigning $x_{i,1} = 0$, *for all* $i \in \{1, \ldots, 5\}$ *satisfies this constraint. Similarly, by symmetry,* $x_{i,2} = x_{j,2} = 0$ *for all* $i \in \{1, \ldots, 5\}$ *satisfies Equation* (5.2d). *This solution is valid for the continuous relaxation of Problem (5.2) and the objective is less than* L_0.

Pisinger and Sigurd (2007) propose a set covering model whose continuous relaxation is at least as tight as the continuous relaxation of Problem (5.2), however the formulation can have an exponential number of constraints and requires the set of all feasible combinations of items that pack in a single bin.

As Equations (5.2b) to (5.2e) show, capturing the notion that two items are either placed in different bins or do not overlap requires several additional binary variables and constraints. Moreover, modelling that items i and j in the same bin do not overlap requires binary variables y^l_{ij}, y^l_{ji}, y^b_{ij} and y^b_{ji}, and four constraints of the Equations (5.2c) and (5.2d) form. Constraint programming may be applied in assessing whether items can be packed in a bin. Martello

et al. (2007) and Pisinger and Sigurd (2007) utilise constraint programming in decomposition frameworks for this task. Since constraint programming-based approaches are aware of the context, e.g. items are two-dimensional, they can make better use of geometric arguments, e.g. the total area of the items must not exceed the total area of the bin. This work differs from the work of Martello et al. (2007) and Pisinger and Sigurd (2007) in that it investigates how SMT may be used to solve 2BP and primarily focusses on how infeasibilities can be leveraged for branching and eliminating symmetry.

5.1.2 Optimising with Satisfiability Modulo Theories

SMT assesses feasibility of a constraint set S. When S is feasible, an SMT solver provides a feasible solution, otherwise SMT derives an unsatisfiable core. Section 2.2.1 Algorithm 1 addresses the Equation (5.1) optimisation problem. For 2BP, Algorithm 1 removes the Equation (5.1a) optimisation objective which results in the Equations (5.1b) to (5.1h) feasibility problem, let this feasibility problem be S_0. After solving S_0 with SMT, calculate U_1, the objective function evaluated at the S_0 feasible solution. Define feasibility problem S_1 by extending S_0 with additional constraint:

$$\sum_{b=1}^{n_C} y_b < U_1,$$

i.e. bound the objective function. Repeat this process until the SMT solver proves some S_i, $i > 0$, is infeasible, and thereby conclude that the S_{i-1} solution is optimal. The Algorithm 1 iterative approach solves successive feasibility problems to derive a sequence of decreasing objective values $\{U_i\}$. The difference between feasibility problems S_0 and S_i is that S_i has additional constraints:

$$\sum_{b=1}^{n_C} y_b < U_{i'}, \quad \forall i' \in \{1, \ldots, i\}. \tag{5.4}$$

The Equation (5.1) optimisation problem may alternatively be addressed via SMT with (i) one of the Section 2.3.5 optimisation frameworks or (ii) a black-box SMT-based optimisation solver (Bjørner et al., 2015; Sebastiani and Trentin, 2015; Callia D'Iddio and Huth, 2017). The following discussion uses SMT as a feasibility solver to leverage unsatisfiable cores for cut

derivation and branching. Our methods are similar to using a logic-based Benders decomposition (Section 2.3.6) where the master problem is embedded into the algorithm.

Symmetry in Two-Dimensional Bin Packing

We develop methods where an SMT solver assesses two decision problems. The first is the two-dimensional orthogonal packing problem (Baker et al., 1980):

$$\mathrm{OPP}(\mathcal{I}', W, H) \tag{5.5}$$

that questions whether a single bin of width W and height H packs all items in $\mathcal{I}' \subseteq [n_C]$. The constraints of feasibility problem OPP are the consequent of Equation (5.1c) and Equation (5.1g). As shorthand, we use $\mathrm{OPP}^{(\mathcal{I})}$ since all bins are equivalent. The second decision problem is the 2BP decision problem:

$$\mathrm{D2BP}([n_C], k, W, H) \tag{5.6}$$

that questions whether k or fewer bins each of width W and height H can pack all items in $[n_C]$. To form the D2BP feasibility model, we remove the Equation (5.1a) objective from the Equation (5.1) optimisation model and set:

$$Y_b = \mathrm{False}, \quad \forall b \in \{k+1, \ldots, n_C\}. \tag{5.7}$$

Propagating the Equation (5.7) assignments reduces D2BP to variables and constraints that only involve bins $b \in \{1, \ldots, k\}$. As shorthand, we use $\mathrm{D2BP}^{(\mathrm{k})}$ since our algorithms are only concerned with the number of available bins.

Two-dimensional bin packing exhibits symmetry, e.g. permuting bin indices immediately results in an identical packing. For an optimal 2BP solution with objective k, the same solution occurs $\binom{n_C}{k}k!$ times. Such symmetries hinder optimality proofs (Margot, 2010; Liberti, 2012). One approach to break this symmetry is by adding additional constraints, e.g. Equation (5.2g). But additional constraints only handle symmetries at a global level and further symmetries

arise at a local level when considering OPP. We study symmetry breaking using SMT in 2BP. Symmetry breaking is often applied in an instance specific manner (Margot, 2003; Ostrowski et al., 2015; Kouyialis and Misener, 2017; Vo-Thanh et al., 2018). Alternatively, symmetry may be approached with automated handling strategies (Margot, 2002; Kaibel et al., 2011; Ostrowski et al., 2011; Kouyialis et al., 2019; Dias and Liberti, 2019).

Descending Strategy Consider an Algorithm 1 implementation that iteratively bounds the objective with Equation (5.4) constraints. Assume that, in the current iteration, the algorithm checks for $(k-1)$ bins or fewer. Constraint (5.4) is problematic because it symmetrically allows any subset of $(k-1)$ active bins, i.e. the SMT solver checks D2BP$^{(k-1)}$ once for each subset of $(k-1)$ bins. We break the Equation (5.4) symmetry by adding a constraint to deactivate bins, i.e. Equation (5.7) sets $\neg Y_b$. Iteratively introducing Equation (5.7) is the *descending* algorithm. Since bin deactivation becomes part of the algorithm, we remove variables y_b (and associated constraints) from the Equation (5.1) formulation. This algorithm aids satisfiability searches because the SMT conflict graph from any previous iteration is always valid in later iterations.

An alternative method is equivalent to binary search (Callia D'Iddio and Huth, 2017). Initialise the algorithm by setting an lower and upper bound on the problem (1 and n_C). At each iteration, activate a number of bins that is halfway between the bounds. Update the lower/upper bound depending on whether the halfway problem is unsatisfiable/satisfiable. Terminate when the bounds are equal. We do not use the binary search algorithm because it assesses unsatisfiability, a costly operation for D2BP and therefore heuristically poor for this particular application.

Ascending Strategy While the descending algorithm progressively builds a conflict graph, it can struggle to efficiently prove optimality as symmetry occurs when addressing infeasibilities. For example, if OPP$^{(\mathcal{I}')}$ is unsatisfiable for some subset of items $\mathcal{I}'$, then having items $\mathcal{I}'$ in any bin for a D2BP$^{(k)}$ instance is a symmetry in unsatisfiability checks. The lower bounding *ascending algorithm* reduces symmetry in optimality proofs. This algorithm naturally extends to the branch-and-bound algorithm described in Section 5.1.3.

At a high level, the ascending and descending algorithms are opposites. The descending algorithm initially activates all bins and iteratively deactivates bins until the first unsatisfiable result (optimality proof). The ascending algorithm initially deactivates all but one bin and iteratively activates one additional bin until the first satisfiable (optimal) result, i.e. it assesses D2BP$^{(k)}$ for $k = 1, \ldots, n_C$ (in order) and terminates at the first satisfiable k.

Each unsatisfiable ascending algorithm iteration proves that we need at least one more bin. Also, each unsatisfiable iteration generates an unsatisfiable core. In the 2BP case, any Equation (5.1c) constraints in the unsatisfiable core are a conflicting subset of items. Assuming that the k^{th} iteration has k active bins and the items corresponding to Equation (5.1c) constraints are $\{i_1, \ldots, i_t\}$, the unsatisfiable core has the interpretation:

> items $\{i_1, \ldots, i_t\}$ cannot be packed into k bins.

In iteration $(k + 1)$, the ascending algorithm derives cuts relating to this set of t items and thereby prevents symmetric unsatisfiability assessments. The symmetric property is: any set of items that cannot be packed in the same bin cannot be packed in any bin. Between ascending algorithm iterations, we run intermediate OPP checks on unsatisfiable core subsets and add:

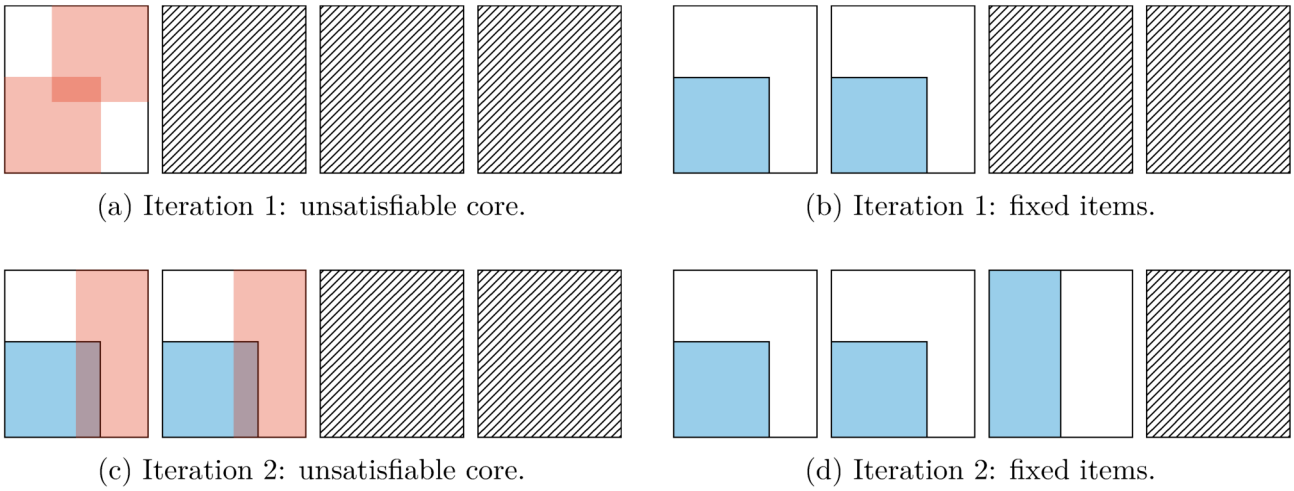
$$\bigvee_{i \in \mathcal{I}'} \neg Y_{ib}, \quad \forall b \in [n_C], \tag{5.8}$$

for any unsatisfiable result of OPP$^{(\mathcal{I}')}$. The MILP equivalent of Equation (5.8) is:

$$\sum_{\substack{i,j \in \mathcal{I}' \\ i \neq j}} p_{ij} + p_{ji} \geq 1. \tag{5.9}$$

Equation (5.8) and (5.9) have the same feasible space, but the Equation (5.9) cut can lose its logical meaning in an MILP solving strategy with relaxed fractional values. Equation (5.8) will not combine the $\mathcal{I}'$ items because Equation (5.8) is a propositional clause that is not relaxed in the SAT subsolve of an SMT solver.

When deriving the Equation (5.8) cuts, deciding OPP is expensive for each unsatisfiable core subset since there are exponentially many such subsets. But the many OPP checks may be



(a) Iteration 1: unsatisfiable core.

(b) Iteration 1: fixed items.

(c) Iteration 2: unsatisfiable core.

(d) Iteration 2: fixed items.

Figure 5.1: How the algorithms fix initial items on iterations 1 and 2. Running the algorithm with one bin may return an unsatisfiable core with two items only (a), this means that these two items must be in their own bins as shown by (b). If later unsatisfiable cores only contain one unfixed item, e.g. (c), then the unfixed item is also placed in a separate bin as shown by (d).

unnecessary since any unsatisfiable set of items dominates any superset for OPP. We leverage this dominance property by checking all subsets in ascending size order and filtering any dominated supersets. Filtering all subsets of a given size terminates the OPP checks and the ascending algorithm continues by assessing the next iteration (additional bin). Effectively, we're building a dictionary of previously checked subsets to prevent repeat checks.

The Equation (5.8) cuts aid SMT unsatisfiability proofs by reducing the number of symmetric OPP checks required to assess D2BP. But, if k active bins are sub-optimal, the SMT solver will still have to derive an unsatisfiable core. Building this new core may contain redundant checks that the ascending algorithm has not yet investigated. We aim to (partially) break this symmetry by fixing items and thereby construct a partial optimal solution as the algorithm progresses, as Figure 5.1 shows. The unsatisfiable core associated with $\text{D2BP}^{(k)}$ implies that we require at least $(k+1)$ bins to pack the items. If the unsatisfiable core in first iteration (one bin) contains only two items, Figure 5.1a, then we pack these items separately, Figure 5.1b. On any later iteration, if the corresponding unsatisfiable core reduces to one item after filtering fixed items, then we know that this single unfixed item can be placed in the next bin, Figures 5.1c and 5.1d show this for iteration 2. We stop fixing items if, after filtering, the corresponding unsatisfiable core has more than one item.

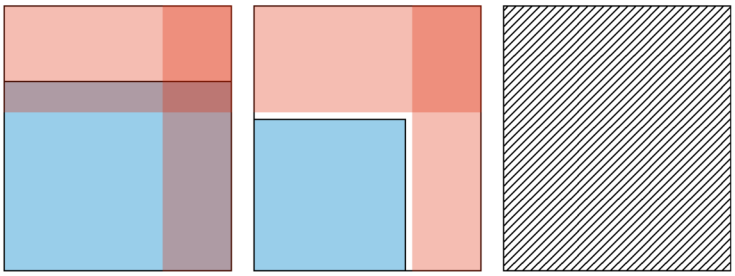


Figure 5.2: A scenario where we cannot definitively choose an item to fix in the next bin. The blue items are assigned to their respective bins (the position is not fixed within the bin) and the red (translucent) items are unfixed. Here one of the two red items must be placed in its own bin but we do not know which choice definitely leads to an optimal solution.

The description above concludes the ascending algorithm definition. this paragraph motivates the branch-and-bound extension described in the next section. The Figure 5.1 process simply derives a set of items that must be placed separately in any feasible solution. When we stop fixing items, the corresponding unsatisfiable core, after filtering, contains more than one item. Here the interpretation is: one of the unfixed items must be placed in its own bin, see e.g. Figure 5.2. As Figure 5.2 shows, to continue fixing items requires assessing alternatives, motivating a branch-and-bound strategy.

5.1.3 Satisfiability Modulo Theories Based Branch-and-Bound for Two-Dimensional Bin Packing

The branch-and-bound algorithm extends the ascending algorithm of Section 5.1.2 by branching on alternative choices from an unsatisfiable core of unfixed items. Figure 5.3 provides an overview of the structure of the branch-and-bound tree.

Taking the root node as a special case where we pop an item from the first unsatisfiable core and fix it in the first bin, each node branches on the elements of the unfixed items in its corresponding unsatisfiable core. The branch-and-bound algorithm matches the ascending algorithm up to a chain of unary branches from the root node of the tree. If the branch-and-bound algorithm requires a non-trivial decision, e.g. Figure 5.2, it forms a set of branches, each of which corresponds to an unfixed item in the current unsatisfiable core. These branches form nodes at some depth d and fix their item into bin $(d + 1)$. An SMT assessment at a given node

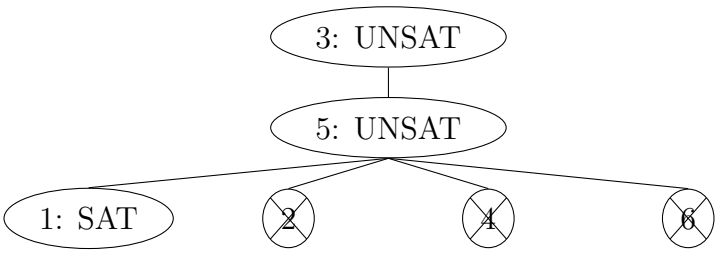


Figure 5.3: An example branching tree for an instance with optimal objective 3. Each node contains an item i and a node's depth, d, corresponds to fixing item i in bin $d+1$. Crossed nodes are pruned. In this instance the first iteration fixed items 3 and 5 in bins 1 and 2. The second iteration derives unsatisfiable core of unfixed items $\{1, 2, 4, 6\}$ (the alternative choices for bin 3) and fixes item 1 in bin 3. The third iteration finds fixing items 3, 5 and 1 separately is feasible. The remaining nodes are pruned since a feasible solution equal to their depth has already been found. There are no further branches after pruning, therefore the bottom left branch gives an optimal solution.

addresses a slightly stricter version of D2BP since there is an item fixed in each bin, i.e. assessing whether we satisfy 2BP in exactly $(d+1)$ bins with the corresponding items fixes. Since the depth of a node corresponds to how many bins are active, the branch-and-bound algorithm does not search beyond the depth of any feasible node that it has found. The branch-and-bound algorithm terminates by exhausting each alternative path.

Since the branch-and-bound algorithm extends the ascending algorithm, it still derives Equation (5.8) cuts in a local setting. Assuming that $\text{OPP}^{(\mathcal{I}')}$ is unsatisfiable with $j \in \mathcal{I}'$ and j fixed in bin b, the associated local cut is:

$$\bigvee_{\substack{i \in \mathcal{I}' \\ i \neq j}} \neg Y_{ib}. \tag{5.10}$$

Furthermore, any local cuts, i.e. cuts depending on a fixed item, are promoted to global cuts by including their corresponding fixed item when the branch-and-bound algorithm investigates sibling or ancestor branches. So all Equation (5.8) cuts can become global regardless of where in the tree they are derived.

Fixing items in the branch-and-bound tree reduces the number of symmetric checks on a given root to leaf path. Adding/promoting Equation (5.8) cuts aids unsatisfiability proofs across the entire unexplored tree. But assessing alternative branches of a given unsatisfiable core retains the symmetry of bin permutations, hence sibling branches contain identical solutions. The main problem is that, if we have explored a particular branch having fixed item i_1, then in a sibling

branch, having the freedom to select any bin for i_1 includes paths explored by the first branch due to bin permutations. Lemma 4 derives *push back* cuts that remove this symmetry by forcing explored branch items to be pushed back into earlier bins.

Lemma 4. *Assume that the branch-and-bound algorithm is assessing alternatives at some node of depth $b-1$, i.e. each of the first b bins have one fixed item, and that this node is unsatisfiable. Let this node have unsatisfiable core of unfixed items $\mathcal{I}' = \{i_1, \ldots, i_m\}$, $m > 1$, i.e. we have to assess each of these items being placed in bin $(b+1)$. When branching on element i_k, $k > 1$, if we add the cuts*

$$\bigvee_{b'=1}^{b} Y_{ib'}, \quad \forall i \in \mathcal{I}(k) = \{i_{k'} | k' < k\}, \tag{5.11}$$

then among all branches the best objective will match that of assessing these alternative branches without adding the Equation (5.11) *cuts.*

Proof. Define OPT_k, $k \in \mathcal{K} = \{1, \ldots, m\}$ as the optimal objective of branch i_k with its corresponding Equation (5.11) cuts. Assume, for a contradiction, that there exists item $i_{k'}$, $1 < k' \leq m$ such that without adding its Equation (5.11) cuts, gives optimal objective OPT' that satisfies:

$$OPT' < \min_{k \in \mathcal{K}} \{OPT_k\}. \tag{5.12}$$

Let f_1 map items to bins for the OPT' solution. We define the set S (items that violate Equation (5.11)) as:

$$S = \{k \,|\, f_1(i_k) \geq b+1,\, i_k \in \mathcal{I}(k')\}.$$

Clearly S is non-empty, otherwise $OPT' = OPT_{k'} \geq \min_{k \in \mathcal{K}} \{OPT_k\}$. Let $k_l = \min S$. We permute the bins such that item k_l is in bin $b+1$ with corresponding item to bin map:

$$f_2(i) = \begin{cases} b+1, & \text{if } f_1(i) = f_1(i_{k_l}) \\ f_1(i_{k_l}), & \text{if } f_1(i) = b+1 \\ f_1(i), & \text{otherwise.} \end{cases}$$

But f_2 is a feasible solution for branch i_{k_l} with its associated Equation (5.11) cuts as $k_l = \min S$

(note that branch i_1 does not add any cuts). Since f_2 has objective OPT':

$$\min_{k \in \mathcal{K}} \{OPT_k\} \leq OPT_{k_l} \leq OPT',$$

contradicting the Equation (5.12) assumption. □

Repeatedly applying Lemma 4 in the branch-and-bound algorithm adds a level of independence between alternative search paths. Equation (5.11) constraints aid unsatisfiability proofs when branching on later elements of $\mathcal{I}'$, since there are a larger number of pushed back items. Furthermore, we automatically remove pushed back items from new branch sets even though they are unfixed, thus reducing the number of branching decisions. Lemma 5 proves that, when pushing back item i, we also can push back all items i' that are identical to i.

Lemma 5. *Assume that the branch-and-bound algorithm is assessing alternatives at some node of depth* $(b-1)$*, i.e. each of the first* b *bins have one fixed item, and that this node is unsatisfiable. Let this node have unsatisfiable core of unfixed items* $\mathcal{I}' = \{i_1, \ldots, i_m\}$*,* $m > 1$*, i.e. we have to assess each of these items being placed in bin* $(b+1)$*. Then when we push back item* $i \in \mathcal{I}'$ *(according to Equation* (5.11)*) we can also push back all items* $i' \in \mathcal{I}$ *that are identical to* i*.*

Proof. Let item i' be identical to i such that i' is not fixed or pushed back (the lemma holds trivially for these cases). With item i pushed back according to its Equation (5.11) constraints, pick any feasible solution with i' placed in bin $b' \geq b+1$ and let f_1 be the associated item to bin map. Then the following map:

$$f_2(j) = \begin{cases} b+1, & \text{if } f_1(j) = b' \\ b', & \text{if } f_1(j) = b+1 \\ f_1(j), & \text{otherwise} \end{cases}$$

permutes bins $(b+1)$ and b'. But f_2 is a feasible solution for the item i branch if we swap identical items i and i'. Since branch i contains any feasible solution with item i' in bin $b' \geq b+1$, we can also push back i' as well as i. □

Finally, since we only fix a single item per bin, we eliminate mirror and rotational symmetries of fixed items by limiting their center to the lower left quadrant of their bins with the constraints:

$$x_{i,1} \leq \frac{W - W_i}{2}, \quad x_{i,2} \leq \frac{H - H_i}{2}. \tag{5.13}$$

Equation (5.13) cuts involve the continuous variables, so they correspond to the arithmetic theory solver aspect of an SMT solver. Since SAT solver checks relax the arithmetic theories, these cuts may be less effective in SMT.

Numerical Results

We solve the MILP and SMT models using CPLEX 12.7 and Gurobi 6.0.3, and Z3 4.5.1 (De Moura and Bjørner, 2008), respectively. The MILP models are in Pyomo (Hart et al., 2011, 2012), and the Z3 implementation is in Python. As Z3 is a feasibility checker, it does not solve Problem (5.1) as a stand-alone package. Tools that support SMT-based black-box optimisation include νZ (Bjørner and Phan, 2014), ManyOpt (Callia D'Iddio and Huth, 2017) and MathSAT (Cimatti et al., 2013). All test cases were run on a HP EliteDesk 800 G1 TWR with 16GB RAM and an Intel® Core™ i7-4770 @ 3.40Ghz running Ubuntu 16.04.1 LTS. The test set contains 500 instances grouped into 10 classes. Each class has 10 instances with 20, 40, 60, 80 and 100 items. These instances were originally generated by Berkey and Wang (1987). Martello and Vigo (1998) and Lodi et al. (1999) describe the differences between classes.

Figure 5.4 is a performance profile comparing time-to-convergence (Dolan and Moré, 2002). We compare the descending, ascending and branch-and-bound algorithm to CPLEX and Gurobi. We also add a *can solve* line for the Pisinger and Sigurd (2007) column generation and constaint programming results. We use SMT to assess OPP, Pisinger and Sigurd (2007) use a specialised algorithm for the underlying OPP decision problem, the 'P&S (2007) solved' *can solve* line in Figure 5.4 gives an indication of the kind of performance improvement achieved by using more bespoke methods. We compare against the mixed-integer solvers in the subsequent analysis. The branch-and-bound algorithm outperforms the alternative SMT-based methods

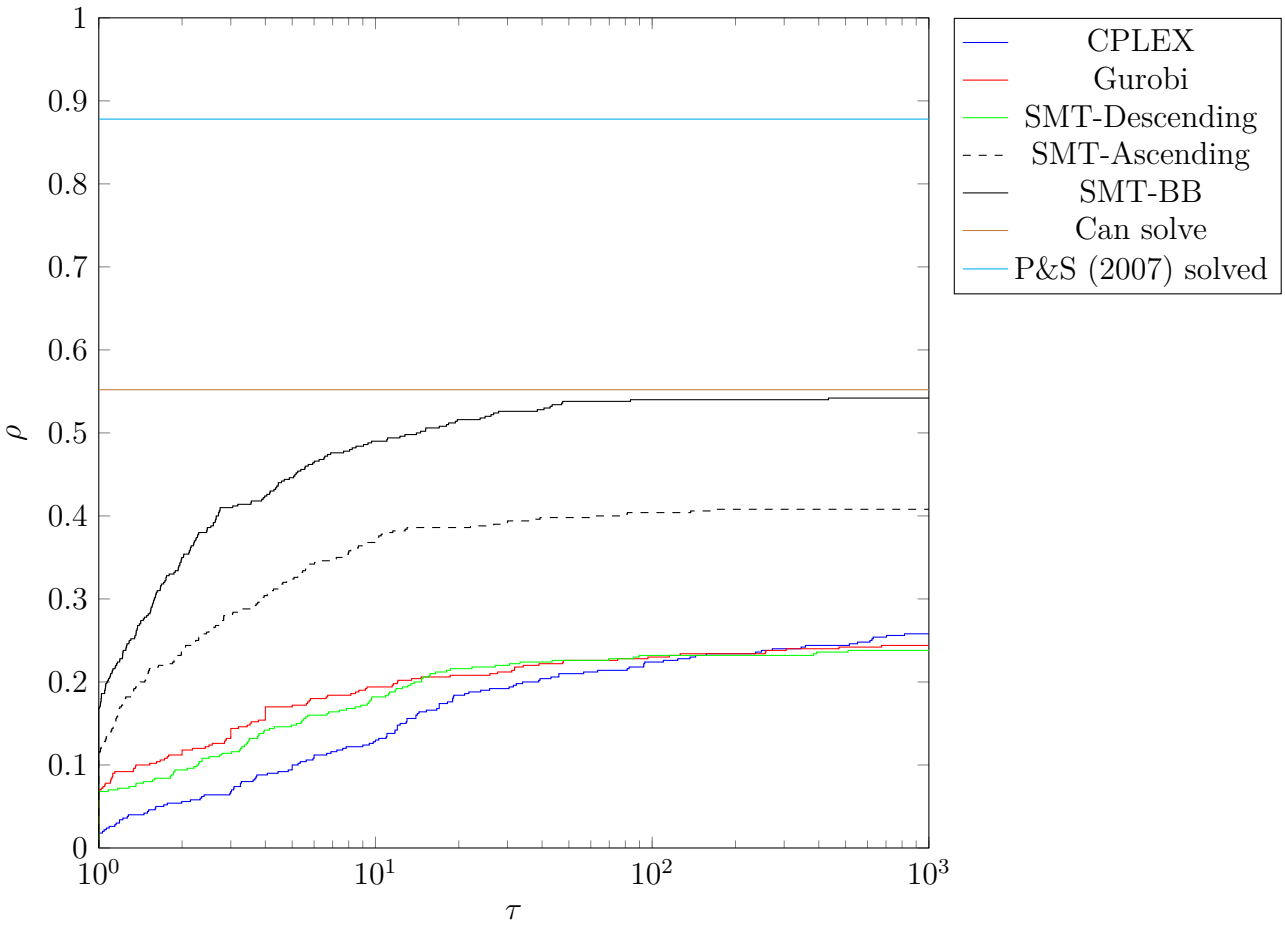


Figure 5.4: Performance profile for 500 bin packing instances. Among CPLEX, Gurobi and the SMT-based algorithms, SMT-BB solves almost all tractable problems and solves the largest proportion of instances first. Each solver has a timelimit of one hour. All solvers are limited to one thread. P&S (2007) is the number of problems that Pisinger and Sigurd (2007) solve to optimality.

Table 5.2: The number of instances (Berkey and Wang, 1987; Martello and Vigo, 1998) solved to optimality by each solver. Each class has 50 instances. Bold formatting indicates which algorithm solves the largest number of instances.

Class	CPLEX	Gurobi	SMT-Descending	SMT-Ascending	SMT-BB
1	10	7	11	22	**36**
2	10	10	**11**	**11**	**11**
3	11	12	17	33	**45**
4	11	11	**12**	11	11
5	13	13	13	32	**43**
6	11	13	**14**	**14**	**14**
7	7	4	10	11	**15**
8	5	3	10	10	**16**
9	47	**49**	1	36	**49**
10	12	8	21	26	**33**

and mixed-integer solvers. The descending algorithm and the mixed-integer solvers perform similarly. The performance difference between the ascending and branch-and-bound algorithms, where the branch-and-bound algorithm solves twice as many instances in the hour, quantifies the effect of 2BP symmetry and how it is better managed by our branch-and-bound algorithm. Furthermore, the performance difference between the ascending and branch-and-bound algorithm shows the effectiveness of the push back constraints and symmetry breaking Equation (5.8) cuts as the branch-and-bound algorithm is able to solve an additional 10% of the instances. Overall, the methods can solve about half of the instances within the hour. For the branch-and-bound algorithm, this limitation is due to larger instances requiring deeper searches within the tree. Deeper searches require larger unsatisfiable cores when proving unsatisfiability and, for the initial part of the search, there are fewer Equation (5.8) cuts present to aid such proofs. However, assuming that there has been sufficient exploration, i.e. we have generated push back constraints and symmetry breaking Equation (5.8) cuts, SMT BB appears to speed up in its later assessments.

Table 5.2 displays the number of instances optimally solved in each class. The branch-and-bound algorithm generally outperforms the other solvers, but there is a fair amount of differences among classes. Generally, the classes where the branch-and-bound algorithm solves more instances contain larger items, i.e. problems with easy-to-generate unsatisfiability proofs. For classes 2, 4 and 6, where instances contain many small items, all methods exhibit inferior performance.

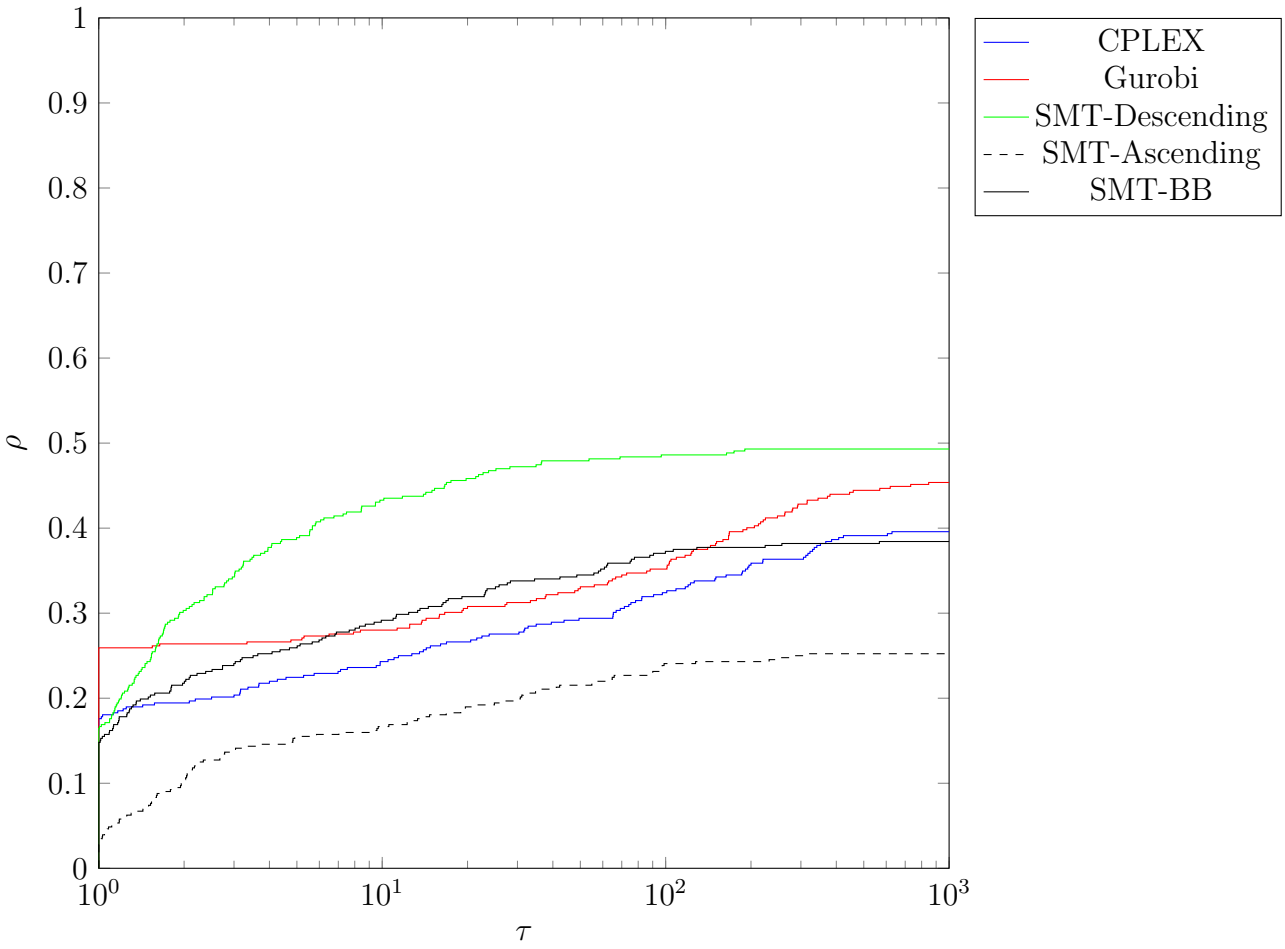


Figure 5.5: Performance profile for 500 bin packing instances comparing heuristic times to optimality. Each solver has a timelimit of one hour. All solvers are limited to one thread.

While the optimal objective is smaller for these instances, for SMT the unsatisfiable cores are relatively large and therefore take longer to generate. Our algorithms use SMT as a black box for unsatisfiable core generation, so the branch-and-bound algorithm may not have sufficient time to explore the tree. The relative quality of the branch-and-bound algorithm shows the usefulness of an unsatisfiable core in algorithm design.

Figure 5.5 shows a performance profile comparing heuristic performance. The descending algorithm is a relatively good heuristic. The descending algorithm progressively reduces the number of available bins, to reduce symmetry, while building a globally applicable conflict graph, that helps in finding heuristic solutions. The ascending algorithm performs less well as its first 'heuristic' solution is an optimal solution, i.e. it has to prove optimality. The branch-and-bound algorithm's heuristic performance is also quite good, this suggests that the tree depth should not be too far from the optimal depth, i.e. the branch-and-bound algorithm does not have to

search too far.

Discussion

Our SMT-based methods utilise SMT as a black-box for solving the 2BP decision problem and the orthogonal packing problem. Since the SMT solver is not necessarily aware of 2BP context, it may perform more enumeration than necessary. Here we discuss approaches address 2BP and how they differ from our methodology.

Martello and Vigo (1998) develop a B&B method for solving 2BP. The Martello and Vigo (1998) approach branches over the item to bin assignment, applies heuristics at each node, leverages 2BP lower bounds to assist in assessing infeasibility of the orthogonal packing problem, and applies a reduction step at the root node that fixes items in the first bin. Our B&B approach branches over unsatifiable cores of the 2BP decision problem, propagates orthogonal packing infeasibilities across the tree, and generates push back constraints that reduce symmetries arising from the algorithm itself. Both the Martello and Vigo (1998) and our B&B algorithm have elements that could improve each other. In particular, the Martello and Vigo (1998) orthogonal packing infeasibility assessments could assist the SMT solver since they account for 2BP problem structure, and applying heuristics at each node of our B&B tree may reduce its overall size. Furthermore, our infeasibility propagation may be applied to the Martello and Vigo (1998) algorithm, especially when their algorithm invokes a sub-B&B instance to assess the orthogonal packing problem.

Lodi et al. (1999) develop a deterministic heuristic and a tabu search metaheuristic (Glover and Laguna, 1998) that are applicable to the 2BP problem that we study. The Lodi et al. (1999) metaheuristic approach can generate solutions quickly, systematically attempts to lower the number of active bins, and performs diversification steps to inspect a variety of item to bin assignments. Among our tested approaches, the descending algorithm has the best performance when considering heuristic solutions. Our descending algorithm has the benefit of being a deterministic global optimisation approach. However, the descending algorithm may struggle to find good solutions as problems scale, since the SMT solver may encounter

symmetric infeasibilities that impede the SMT solver from exploring a diverse range of item to bin assignments.

5.2 Unsatisfiable Core for Cut Generation and Model Explainers

SMT solvers can derive an *unsatisfiable core* of a constraint set, i.e. a constraint subset that is unsatisfiable. This section discusses cut generation and model explainers as two applications of unsatisfiable cores.

Cut Generation

Logic-based Benders decomposition (LBBD) (Hooker and Ottoson, 2003) is a framework for solving problems with master-subproblem structure, e.g. a high level assignment and subproblem assessments resulting from the assignments. In LBBD, the subproblem generates cutting planes to add to the master problem. These cutting planes can be *infeasibility cuts* which exclude infeasible solutions or *lower bounding cuts* which enforce an objective bound.

In each major iteration, a subproblem can generate a cut incorporating all of its corresponding assignments, but the resulting cut is often weak because fewer assignments may give an equivalent cut, i.e. a smaller unsatisfiable core. Hooker (2007) develops an approach generating stronger cuts by repeating local subproblem assessments and removing subsets that do not change the underlying result. Hooker (2007) thereby derives a minimal unsatisfiable core, an unsatisfiable core where removing any constraint makes it satisfiable. SMT does not necessarily calculate minimal unsatisfiable cores, but the reported cores may be smaller than the entire assignment set. Smaller cores imply fewer of the expensive local iterations needed to reduce to a minimal unsatisfiable core.

Model Explainers

Practical applications of optimisation models and algorithms involve using abstract modelling software. Consider a scenario where a mathematical model is known to be correct and an instance is known to be feasible. If a solver or algorithm states infeasibility, then an implementation needs to be corrected. Here SMT is a useful tool to aid development.

A feasible but incorrect model may be over- or under-constrained. Over-constrained models may report an sub-optimal objective or infeasible constraints. Reasoning the incorrectness of a model implementation may be difficult because of various properties, e.g. the instance size or incorrect application of transformation techniques. Analyzing a concrete instance can be a time consuming task since it may have a large number of constraints. But an incorrect model still contains a constraint subset explaining what is wrong, i.e. an unsatisfiable core.

When an incorrect model is over-constrained, it is sufficient to limit our discussion to the infeasible case since, if the model reports a sub-optimal objective, we are questioning why a better objective is infeasible. The unsatisfiable core corresponding to the infeasible model may have many constraints, but we can filter, i.e. eliminate, correct constraints such as variable bounds. After filtering, the constraint subset may be much smaller than the entire instance and analysis becomes easier. If we assume further that we know a feasible solution, we can add constraints fixing variables to this solution. Fixing variables may produce an unsatisfiable core in terms of the feasible solution that we fed to the model. Fixing a solution is more useful when the incorrect model reports the wrong optimal objective (here we would need to know a feasible solution that is better than the reported optimal objective).

Another, more likely case for elusive bugs is when we have an algorithm that adds and removes constraints, i.e. an issue may only occur for larger instances. Compulsory computational considerations may be a reason for having such a case, e.g. a given node of a branch-and-bound algorithm can add and remove constraints to an existing instance, or build a fresh instance. However, we may prefer the former when considering the computational cost of building a fresh instance. Figure 5.6 provides a concrete example in the context of the branch-and-bound

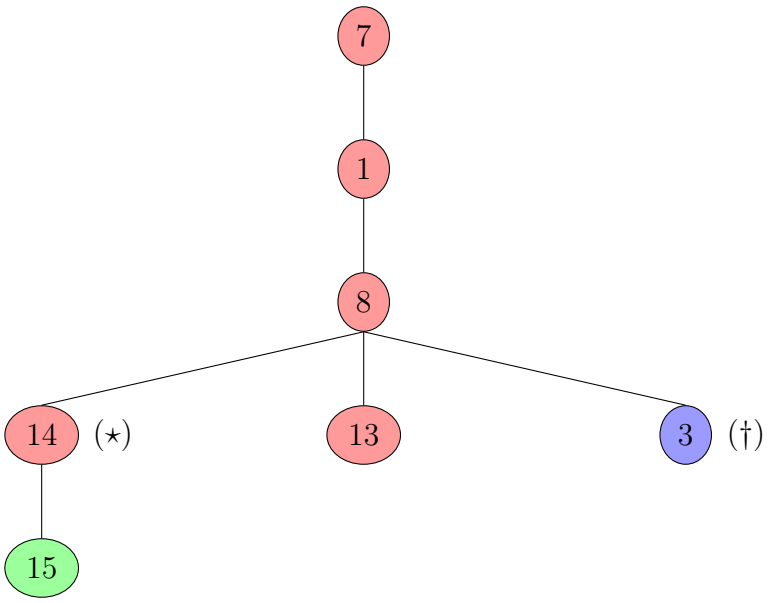


Figure 5.6: An example of a Section 5.1.3 branch-and-bound tree where an incorrect implementation requires correction. Red nodes are unsatisfiable, green nodes are satisfiable and blue nodes are reported unsatisfiable nodes that we know a feasible solution for, i.e. an incorrect conclusion. The error assumed here is that we generate a local cut that item 15 is not in bin 4 at (⋆), however the incorrect implementation fails to remove/promote the local cut before switching to branch (†) giving the wrong conclusion.

algorithm of Section 5.1.3. The incorrect implementation assumed here is that local cuts are not removed when switching to branches where they are no longer valid. As Figure 5.6 shows, we get a reported optimal objective of 5 when we should get 4. After filtering constraints at (†) in Figure 5.6, e.g. variable bounds and inactive bins, an unsatisifable core consists of only 7 constraints among which are constraints generated by the algorithm. In particular, we have the constraint *item 15 is not in bin 4*, the significance of this constraint is that it corresponds to a local cut generated at (⋆) therefore it is no longer valid telling us that there is an issue with handling of local cuts. The fact that we can limit our view to just a few constraints makes this error easier to find. Given just the result 'infeasible', finding such an error may require multiple re-solves with minor source code changes or log file parses, both of these tasks are time consuming (more so if the only failing instances are large).

Chapter 6

Branching and Infeasibility in Convex Generalized Disjunctive Programming

The convex generalized disjunctive programming (GDP) framework (Raman and Grossmann, 1994) is an alternative modelling framework to mixed-integer convex programming. Convex GDP applications include packing problems, layout problems and scheduling (Sawaya and Grossmann, 2005; Sawaya, 2006; Castro and Grossmann, 2012). Convex GDP also appears when relaxing a (nonconvex) GDP instance. GDP applications include heat exchanger network synthesis and reactor network design (Mizutani et al., 2003; Ruiz and Grossmann, 2013).

Convex GDPs can be reformulated to a convex MINLP (Trespalacios and Grossmann, 2014, 2015b). Hence, branch-and-bound (B&B) implementations for convex MINLP can solve convex GDPs. In the case of convex GDP reformulations, a convex MINLP B&B solver generally branches on binary variables, i.e. considering the two subproblems that fix $y = 0$ and $y = 1$, respectively. However, enforcing that every feasible solution must choose a single disjunct in each disjunction, as convex GDP does, motivates a branching approach that partitions disjunctions. Beaumont (1990) and Lee and Grossmann (2000) design B&B algorithms that select a disjunction for branching and introduce a new child for each disjunct.

This chapter investigates branching on disjunctions in convex GDP. We study (i) how to choose which disjunction to branch on, (ii) properties that make for good partitions and (iii) how to

construct branches. Branching in convex GDP may also encounter infeasible subproblems, we analyse how we may leverage such an infeasibility to prevent a similar infeasibility appearing elsewhere in the search.

Chapter organisation Section 6.1 gives an overview of the B&B algorithm. Section 6.2 develops disjunction selection and branch construction strategies. Section 6.3 analyses how infeasibility may be leveraged in convex GDP. Section 6.4 formulates the constrained layout problem. Section 6.5 performs numerical tests on constrained layout instances.

6.1 Overview

This chapter presents B&B Algorithm 9 for solving convex GDPs. Algorithm 9 uses a hull reformulation (Grossmann and Lee, 2003) to derive lower bounds. Similar to Beaumont (1990) and Lee and Grossmann (2000), our B&B algorithm, on lines 21 and 22, branches on a disjunction from the convex GDP instance. Our approach differs in that we allow the branching strategy to partition the disjunction into disjoint smaller disjunctions opposed to creating a single child for each disjunct.

Each node in the B&B tree considers a hull relaxation (Grossmann and Lee, 2003) of the local convex GDP instance. Hull relaxations form the convex hull of individual disjunctions using the perspective function (Stubbs and Mehrotra, 1999). Hull relaxations introduce many additional variables and can encounter numerical difficulties (Grossmann and Lee, 2003; Sawaya and Grossmann, 2007; Furman et al., 2020). Using Big-M relaxations (Nemhauser and Wolsey, 1988; Trespalacios and Grossmann, 2014) involves less variables and are more numerically stable, however big-M relaxations generally derive weaker optimality bounds and, therefore, may result in more enumeration. One remedy for a weak relaxation is to generate cutting planes, e.g. Vecchietti et al. (2003) or Trespalacios and Grossmann (2016a), at a node level.

Algorithm 9

$UB \leftarrow \infty$
S^{root}: Initial relaxation feasible convex GDP instance
Select $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{\nu}}) \in \arg\min_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\nu}) \in R^H(S^{\text{root}})} \boldsymbol{c}^\top \boldsymbol{x}$
if $\hat{\boldsymbol{y}}$ is integer feasible **then**
    return $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$
else
    $Q \leftarrow \{(S^{\text{root}}, (\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{\nu}}))\}$
end if
while $Q \neq \emptyset$ **do**
    Select $(\hat{S}, (\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{\nu}})) \in Q$
    $k \leftarrow$SelectDisjunction$(\hat{S},\ (\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{\nu}}))$ ▷ Section 6.2.2
    $(S_1, \ldots, S_m) \leftarrow$PartitionDisjunction$(\hat{S},\ k,\ (\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{\nu}}))$ ▷ Section 6.2.3
    for $S' \in \{S_1, \ldots, S_m\}$ **do**
        if $R^H(S') \neq \emptyset$ **then**
            Select $(\boldsymbol{x'}, \boldsymbol{y'}, \boldsymbol{\nu'}) \in \arg\min_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\nu}) \in R^H(S')} \boldsymbol{c}^\top \boldsymbol{x}$
            if $\boldsymbol{c}^\top \boldsymbol{x'} < UB$ **then**
                if $\bar{\boldsymbol{y}}$ is integer feasible **then**
                $(\boldsymbol{x^*}, \boldsymbol{y^*}, UB) \leftarrow (\boldsymbol{x'}, \boldsymbol{y'}, \boldsymbol{c}^\top \boldsymbol{x'})$
                $Q \leftarrow \{(S, (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\nu})) \in Q \mid \boldsymbol{c}^\top \boldsymbol{x} < UB\}$
                else
                $Q \leftarrow Q \cup \{(S', (\boldsymbol{x'}, \boldsymbol{y'}, \boldsymbol{\nu'}))\}$
                end if
            end if
        else
            PropagateInfeasibility$(Q,\ S')$ ▷ Section 6.3
        end if
    end for
    $Q \leftarrow Q \setminus \left\{\left(\hat{S}, (\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{\nu}})\right)\right\}$
end while
if $UB < \infty$ **then**
    return $(\boldsymbol{x^*}, \boldsymbol{y^*})$
else
    return *infeasible*
end if

Notation We use S to denote B&B nodes. S^{root} denotes the root node of the B&B tree. We also treat S as the feasible domain associated with the B&B node, e.g. $(\boldsymbol{x}, \boldsymbol{Y}) \in S$. $R^H(S)$ denotes the hull relaxation of S. As for B&B nodes, $R^H(S)$ is also treated as a set of feasible points, e.g. $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\nu}) \in R^H(S)$. $S = \emptyset$ and $R^H(S) = \emptyset$ denote that B&B node S and its hull relaxation are infeasible, respectively. $[n]$ denotes the set $\{1, \ldots, n\}$, for $n \in \mathbb{Z}_{\geq 1}$. Table 4 summarises the symbols that this chapter uses.

6.2 Branching in Convex Generalized Disjunctive Programming

Algorithm 9 applies branching to B&B node S on lines 15 and 16. Note that, between lines 14 and 15, an extension to Algorithm 9 could manipulate S further, e.g. by applying basic steps or generating cutting planes (Vecchietti et al., 2003; Ruiz and Grossmann, 2012; Trespalacios and Grossmann, 2016a). Branching may still be applied after such manipulations.

This section describes branching strategies that branch on disjunctions. Definition 6.1 non-trivial convex GDPs ensure that disjunctions are present.

Definition 6.1. *A convex GDP instance S is non-trivial if it has at least one disjunction, i.e. $K \neq \emptyset$.*

Child Node Construction Algorithm 9 applies the same solution strategy to all nodes of the search tree. Hence, all constructed subproblems should also be convex GDPs. The convex GDP Equation (2.15d), and similarly the convex MINLP Equation (2.17c), selection constraints allow for a branching strategy that splits a disjunction. Definition 6.2 constructs a subproblem of convex GDP S by reducing a disjunction $k \in K$.

Definition 6.2. *Let S be a non-trivial convex GDP instance, $k \in K$, $D'_k \subset D_k$, $D'_k \neq \emptyset$. Convex GDP instance* $\text{child}(S, k, D'_k)$ *is the subproblem constructed from S by reducing disjunction k to disjuncts indexed by D'_k. Specifically, if $|D'_k| > 1$,* $\text{child}(S, k, D'_k)$ *replaces the index k instances of Equations* (2.15c) *and* (2.15d) *in S with*

$$\bigvee_{i \in D'_k} \begin{bmatrix} Y_{ik} \\ r_{ikj}(\boldsymbol{x}) \leq 0, \quad j \in [p_{ik}] \end{bmatrix}, \tag{6.1}$$

$$\underline{\bigvee}_{i \in D'_k} Y_{ik}, \tag{6.2}$$

otherwise, if $|D'_k| = 1$, $\text{child}(S, k, D'_k)$ *replaces the index k instances of Equations* (2.15c)

and (2.15d) *in* S *with*

$$r_{ikj}(\boldsymbol{x}) \leq 0, \quad i \in D'_k,\ j \in [p_{ik}], \tag{6.3}$$

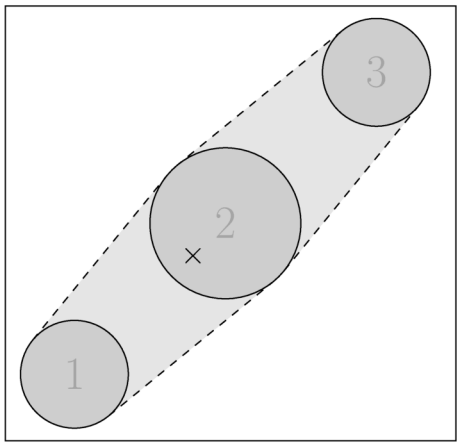
where $[p_{ik}] = \{1, \ldots, p_{ik}\}$. *Furthermore,* $Y_{ik} = \mathit{False}$ *is propagated for all* $i \in D_k \setminus D'_k$ *in the instance* S *Equation* (2.15e) *propositional constraints. If* $|D'_k| = 1$, *i.e.* $D'_k = \{i\}$ *for some* $i \in D_k$, *then* $\text{child}(S, k, D'_k)$ *also propagates* $Y_{ik} = \mathit{True}$.

Let S be a non-trivial convex GDP instance and $P = \{D_k^1, \ldots, D_k^m\}$, $m > 1$ be a partition of index set D_k, for some disjunction $k \in K$. Then P constructs subproblems $\{S_i\}_{i=1}^m$ where $S_i = \text{child}(S, k, D_k^i)$. Hence, subproblems $S_1, \ldots, S_m$: (i) cover all feasible solutions of S, (ii) are disjoint, and (iii) all belong to the convex GDP problem class, i.e. partitioning a disjunction's disjunct indices forms a branching strategy for convex GDP. This partition-based branch construction subsumes: (a) branching on a Boolean variable in convex GDP (or binary variables in a convex MINLP reformulation), e.g. consider the partition $P = \{\{i\}, D_k \setminus \{i\}\}$ for disjunction k, and (b) the Beaumont (1990) and Lee and Grossmann (2000) wide branching strategy, e.g. consider the partition $\{\{i\} \mid i \in D_k\}$ for disjunction k.

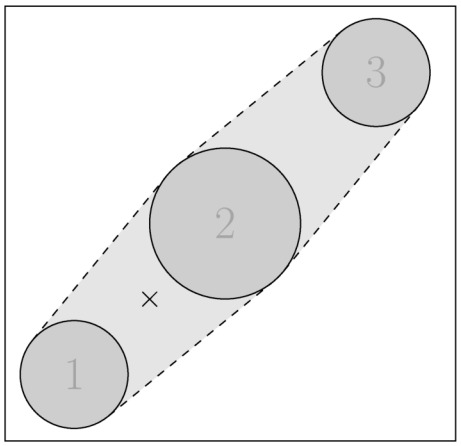
The remainder of this section discusses fractionality (Section 6.2.1), disjunction selection (Section 6.2.2), branch separation (Section 6.2.3), and branch construction (Section 6.2.4).

6.2.1 Fractionality

Branching in convex GDP requires a fractional disjunction. Solving a convex GDP hull relaxation includes assignments $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$. Definition 2.5 assesses fractionality using the binary variables only. However, the convex GDP objective is only dependent on the $\hat{\boldsymbol{x}}$ assignment. If $\hat{\boldsymbol{x}}$ is not contained within a disjunct then we can guarantee that branching can eliminate (infeasible) assignment $\hat{\boldsymbol{x}}$ from descendant relaxations. Furthermore, the presence of relaxation regions that include $\hat{\boldsymbol{x}}$ prevent the local lower bound from improving. Figure 6.1 shows two cases that adhere to Definition 2.5. Relaxation solutions of the Figure 6.1b form can necessarily be eliminated with branching. Definition 6.3 introduces *qualitative fractionality* which considers fractionality



(a) A relaxation solution that can be fractional but not qualitatively fractional, i.e. the relaxation solution is contained in disjunct 2. The relaxation solution can be fractional since it could be defined as a convex combination of solutions in disjunct 1 and 3.



(b) A qualitatively fractional relaxation solution and therefore also a fractional relaxation solution (Proposition 1).

Figure 6.1: Example of the relationship between a Definition 2.5 *fractional* relaxation solution and Definition 6.3 *qualitatively fractional* relaxation solution. The disjunction is the choice of being in region 1, 2 or 3. The cross is a relaxation solution. The entire shaded region is the convex hull of the disjunction.

of $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$ from the perspective of continuous variables $\boldsymbol{x}$. Intuitively, Definition 6.3 corresponds to relaxation solutions of the Figure 6.1b form.

Definition 6.3. *Let S be a non-trivial convex GDP instance and $\hat{\boldsymbol{x}} \in \text{proj}_{\boldsymbol{x}} R^H(S)$. Disjunction k is qualitatively fractional under $\hat{\boldsymbol{x}}$ if for all $i \in D_k$, there exists $j \in [p_{ik}]$ such that $r_{ikj}(\hat{\boldsymbol{x}}) > 0$.*

Figure 6.1b suggests that a qualitatively fractional relaxation solution must be fractional, Proposition 1 proves this result.

Proposition 1. *Let S be a non-trivial convex GDP instance and $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \text{proj}_{(\boldsymbol{x},\boldsymbol{y})} R^H(S)$. If disjunction k is qualitatively fractional under $\hat{\boldsymbol{x}}$ then it is fractional under $\hat{\boldsymbol{y}}$.*

Proof. We prove the contrapositive. Assume that disjunction k is not fractional under $\hat{\boldsymbol{y}}$. Then for all $i \in D_k$, $\hat{y}_{ik} \notin (0, 1)$. Since the relaxation solution satisfies Equations (2.17c) and (2.24), we have that $\hat{y}_{i'k} = 1$ for some $i' \in D_k$ and $\hat{y}_{ik} = 0$, for all $i \in D_k \setminus \{i'\}$. But this means that disjunction k selects disjunct i' in the relaxation solution and therefore $r_{i'kj}(\hat{\boldsymbol{x}}) \leq 0$, for all $j \in [p_{ik}]$, i.e. disjunction k is not qualitatively fractional. □

Figure 6.1a suggests that, under relaxation solution $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$, if disjunction k is fractional but not qualitatively fractional then it may be converted to an equivalent solution that assigns disjunction k, i.e. select the disjunct that contains $\hat{\boldsymbol{x}}$. Proposition 2 shows this case for convex GDPs without propositional constraints.

Proposition 2. *Let S be a non-trivial convex GDP instance without any Equation* (2.17d) *propositional constraints, $R^H(S)$ its hull relaxation, and $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \text{proj}_{(\boldsymbol{x},\boldsymbol{y})}\, R^H(S)$. If disjunction $\bar{k} \in K$ is not qualitatively fractional under $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$. Then there exists a $(\hat{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \in \text{proj}_{(\boldsymbol{x},\boldsymbol{y})}\, R^H(S)$ where*

$$\bar{y}_{ik} = \begin{cases} 1, & i = \bar{i},\ k = \bar{k} \\ 0, & i \neq \bar{i},\ k = \bar{k} \\ \hat{y}_{ik}, & \textit{otherwise} \end{cases} \tag{6.4}$$

for some $\bar{i} \in D_{\bar{k}}$.

Proof. Since S does not have any Equation (2.17d) propositional constraints, the Equation (2.15a) objective, Equation (2.15b) global constraints, and all disjunctions $k' \in K \backslash \{\bar{k}\}$ are independent of the Boolean variables $Y_{i\bar{k}}$, $i \in D_{\bar{k}}$. Therefore, disjunction $\bar{k}$ only affects the other disjunctions and constraints through continuous variables $\boldsymbol{x}$. Hence, it suffices to show that some Equation (6.4) assignment does not exclude $\hat{\boldsymbol{x}}$ from relaxed disjunction $\bar{k}$.

Since disjunction $\bar{k}$ is not qualitatively fractional under $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$ there exists a $i' \in D_{\bar{k}}$ such that $r_{i'\bar{k}j}(\hat{\boldsymbol{x}}) \leq 0$ for all $j \in [p_{i'\bar{k}}]$. Claim $\bar{i} = i'$. Let $\hat{\boldsymbol{\nu}}$ be the corresponding assignment on disaggregated variables $\boldsymbol{\nu}$ for the $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$ solution. Consider the assignment

$$\bar{\boldsymbol{\nu}}_{\boldsymbol{ik}} = \begin{cases} \hat{\boldsymbol{x}}, & i = \bar{i},\ k = \bar{k} \\ \boldsymbol{0}, & i \neq \bar{i},\ k = \bar{k} \\ \hat{\boldsymbol{\nu}}_{\boldsymbol{ik}}, & \text{otherwise.} \end{cases}$$

By selection of i', $(\bar{\boldsymbol{y}}, \bar{\boldsymbol{\nu}})$ satisfies Equation (2.22a). All Equations (2.22a) and (2.22c) constraints under $(\bar{\boldsymbol{y}}, \bar{\boldsymbol{\nu}})$ for $i \in D_{\bar{k}} \setminus \{i'\}$ reduce to trivial inequalities, i.e. $\boldsymbol{0} \leq \boldsymbol{0}$. Finally, Equation (2.22b) reduces to $\boldsymbol{x} = \hat{\boldsymbol{x}}$ and the result follows. □

Proposition 2 shows that a non-qualitatively fractional disjunction may be converted to an integer feasible disjunction without affecting the convex GDP relaxation objective. In the presence of Equation (2.17d) propositional constraints, Proposition 2 does not necessarily hold. However, if a disjunction is fractional but not qualitatively fractional under $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$ and Equation (6.4) does not break any of the Equation (2.17d) relaxed propositional constraints, then the alternative, less fractional solution constructed in the Proposition 2 proof is still applicable.

6.2.2 Disjunction Selection

Branching variable selection in convex MINLP, i.e. selecting which variable to branch on can effect the performance of a B&B solver (Morrison et al., 2016). Branching variable selection strategies include using formulas, setting priorities based on problem specific knowledge, tracking and updating scores, and solving auxiliary problems (Gupta and Ravindran, 1985; Achterberg et al., 2005; Klabjan et al., 2001; Gilpin and Sandholm, 2011; Fischetti and Monaci, 2012, 2013). Indeed, after using a reformulation, these branching variable selection also become applicable to convex GDP. However, branching on disjunctions motivates disjunction selection, i.e. selecting which disjunction to branch on. Rado (1963) defines the infeasibility of $\boldsymbol{a}^\top \boldsymbol{x} \leq b$ as $\boldsymbol{a}^\top \boldsymbol{x} - b$ and scores linear disjunctions by their disjunct that has the smallest infeasibility. The selection strategy chooses the disjunction with the largest infeasibility. Beaumont (1990) proposes a disjunction selection strategy for linear disjunctive programs that considers shadow prices.

If all disjunctions are integer feasible then we have a feasible solution of the convex GDP. Any disjunction that is fractional under relaxation solution $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$, i.e. Definition 2.5, is a candidate for branching. We denote these candidates with $K^F(\hat{\boldsymbol{y}}) \subseteq K$, i.e.

$$K^F(\hat{\boldsymbol{y}}) = \{k \in K \,|\, \text{disjunction } k \text{ is fractional under } \hat{\boldsymbol{y}}\}.$$

Table 6.1 summarises the disjunction selection strategies that this section proposes.

Grossmann and Lee (2003) propose selecting the fractional disjunction with the largest binary assignment. Definition 6.4 refers to this approach as the least fractional selection strategy, since

Table 6.1: Disjunction selection strategies. $K^F(\hat{\boldsymbol{y}})$ is the set of disjunctions that are fractional under $\hat{\boldsymbol{y}}$.

Name [Definition]	Formula	Description
Least fractional (Grossmann and Lee, 2003) [Definition 6.4]	$\arg\max_{k \in K^F(\hat{\boldsymbol{y}})} \max_{i \in D_k} \hat{y}_{ik}$	Scores fractional disjunctions by their largest binary variable. Chooses disjunction with the largest score, i.e. closer to being assigned.
Most fractional [Definition 6.5]	$\arg\min_{k \in K^F(\hat{\boldsymbol{y}})} \max_{i \in D_k} \hat{y}_{ik}$	Scores fractional disjunctions by their largest binary variable. Chooses disjunction with the smallest score, i.e. further from being assigned.
Most non-zero [Definition 6.6]	$\arg\max_{k \in K^F(\hat{\boldsymbol{y}})} \frac{\{\hat{y}_{ik} \mid y_{ik} > 0, i \in D_k\}}{\lvert D_k \rvert}$	Scores fractional disjunctions by the proportion of non-zero binary variables. Chooses disjunction with the largest score, i.e. largest number of fractional selections.
Centre-shifted most fractional [Definition 6.7]	$\arg\min_{k \in K^F(\hat{\boldsymbol{y}})} \left\lVert \hat{\boldsymbol{y}}_{\boldsymbol{k}} - \frac{1}{\lvert D_k \rvert}\mathbf{1} \right\rVert$	Scores fractional disjunctions by their distance to vector that equally assigns all disjunct selecting variables. Chooses disjunction with the smallest score, i.e. most undecided disjunction.

it considers a disjunction that is closest to being integer feasible.

Definition 6.4 (Grossmann and Lee (2003))**.** *Let S be a non-trivial convex GDP and $\hat{\boldsymbol{y}} \in \operatorname{proj}_{\boldsymbol{y}} R^H(S)$. Assume that $\hat{\boldsymbol{y}}$ is fractional for S. Then disjunction $k \in K^F(\hat{\boldsymbol{y}})$ is a least fractional disjunction if*

$$k \in \arg\max_{k' \in K^F(\hat{\boldsymbol{y}})} \max_{i \in D_{k'}} \hat{y}_{ik'}.$$

Similarly, Definition 6.5 defines a most fractional disjunction, which, like Definition 6.4, scores each disjunction by its largest binary assignment. However, now the strategy selects a disjunction with the smallest of these scores. Hence, Definition 6.5 aims to quantify which disjunction is most undecided in the relaxation solution.

Definition 6.5. *Let S be a non-trivial convex GDP and $\hat{\boldsymbol{y}} \in \operatorname{proj}_{\boldsymbol{y}} R^H(S)$. Assume that $\hat{\boldsymbol{y}}$ is fractional for S. Then disjunction $k \in K^F(\hat{\boldsymbol{y}})$ is a most fractional disjunction if*

$$k \in \arg\min_{k' \in K^F(\hat{\boldsymbol{y}})} \max_{i \in D_{k'}} \hat{y}_{ik'}.$$

While the Definition 6.5 most fractional selection strategy tries to selected an undecided disjunction, it discriminates between disjunctions by only considering the largest binary assignment in each disjunction. For example, consider two length 4 disjunctions D_1 and D_2 where the relaxation assignment on their binary variables are $(0.55, 0.45, 0, 0)$ and $(0.6, 0.2, 0.1, 0.1)$, respectively. Among these two disjunctions, most fractional selection will select D_1. However, D_2 has a larger number of partially selected disjuncts. Definition 6.6 provides an alternative selection strategy that accounts for the assignments on all disjuncts.

Definition 6.6. *Let S be a non-trivial convex GDP and $\hat{\boldsymbol{y}} \in \text{proj}_{\boldsymbol{y}} R^H(S)$. Assume that $\hat{\boldsymbol{y}}$ is fractional for S. Then disjunction $k \in K^F(\hat{\boldsymbol{y}})$ is a most non-zero disjunction if*

$$k \in \underset{k' \in K^F(\hat{\boldsymbol{y}})}{\arg\max} \frac{|\{\hat{y}_{ik'} \mid \hat{y}_{ik'} > 0, i \in D_{k'}\}|}{|D_{k'}|}.$$

The Definition 6.6 most non-zero selection strategy naively quantifies fractionality as the proportion of binary variables that are non-zero in the relaxation solution. However, this strategy does not account for the magnitude of the binary variables in the relaxation solution. For example, binary assignments $(0.9, 0.03, 0.03, 0.04)$ seem closer to integer feasibility than binary assignments $(0.4, 0.3, 0.15, 0.15)$ in a size 4 disjunction. For disjunction $k \in K$, the most undecided assignment we can have on its binary variables, i.e. the furthest from integer feasibility, is $\boldsymbol{y}^\dagger = \frac{1}{|D_k|}\mathbf{1}$, i.e. all disjunct binary variables are equally assigned in the relaxation solution. Definition 6.7 scores disjunctions by considering the distance of their binary assignment from $\boldsymbol{y}^\dagger$.

Definition 6.7. *Let S be a non-trivial convex GDP and $\hat{\boldsymbol{y}} \in \text{proj}_{\boldsymbol{y}} R^H(S)$. Assume that $\hat{\boldsymbol{y}}$ is fractional for S. Then disjunction $k \in K^F(\hat{\boldsymbol{y}})$ is a centre-shifted most fractional disjunction if*

$$k \in \underset{k' \in K^F(\hat{\boldsymbol{y}})}{\arg\min} \left\| \hat{\boldsymbol{y}}_{\boldsymbol{k'}} - \frac{1}{|D_{k'}|}\mathbf{1} \right\|$$

where $\|\cdot\|$ is any norm.

6.2.3 Separation by Branching

After selecting disjunction $k \in K^F(\hat{\boldsymbol{y}})$, Algorithm 9 constructs subproblems at B&B node S by considering a partition of D_k. Any non-trivial partition of D_k forms a set of branches by constructing subproblems according to Definition 6.2. Disjointedness of the subproblem feasible regions follows since the branches partition the set of valid Boolean assignments, i.e the set

$$\left\{ \left(Y_{1k}, \ldots, Y_{|D_k|k}\right)^\top \in \{\text{True}, \text{False}\}^{|D_k|} \,\middle|\, \bigvee_{i \in D_k} Y_{ik} \right\}. \tag{6.5}$$

If $\hat{y}_{ik} = 0$ for some $i \in D_k$, then an arbitrary partition may be redundant, e.g. $P = \{\{i\}, D_k \setminus \{i\}\}$ includes $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$ in the subproblem hull relaxation associated with the second subset. Constructing a partition P' of D_k such that two distinct subsets in P' respectively contain indices $i_1, i_2 \in D_k$ and $\hat{y}_{i_1,k}, \hat{y}_{i_2,k} > 0$ avoids this redundancy. Grossmann and Lee (2003) identify that partitioning based on fractionality of $\hat{\boldsymbol{y}}$ alone can hinder improving optimality bounds, since the convex GDP objective only depends on $\hat{\boldsymbol{x}}$. Example 6.1 describes such a scenario.

Example 6.1. *Consider Figure 6.1b. Let the hull relaxation solution associated with this disjunction be* $(\hat{\boldsymbol{x}}, (\hat{y}_1, \hat{y}_2, \hat{y}_3))$ *with all* $\hat{y}_i > 0$. *Partition* $P = \{\{1, 2\}, \{3\}\}$ *clearly excludes* $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$, *however the subproblem induced by subset* $\{1, 2\}$ *includes a relaxation solution* $(\hat{\boldsymbol{x}}, (\bar{y}_1, \bar{y}_2, 0))$ *for some* $\bar{y}_1, \bar{y}_2 > 0$, *i.e. the lower bound found by considering all three disjuncts does not improve after dividing the disjunction. Furthermore, since the Figure 6.1b disjunction is qualitatively fractional under the* $\hat{\boldsymbol{x}}$, *it is more preferable for a partition to exclude* $\hat{\boldsymbol{x}}$ *from the relaxations of the subproblems it creates.*

Definition 6.8 *separating partitions* avoid the search redundancy that Example 6.1 describes, i.e. they construct branches whose feasible regions do not include the current relaxation assignment to continuous variables $\boldsymbol{x}$.

Definition 6.8. *Let* S *be a non-trivial convex GDP instance and* $R^H(S)$ *be its hull relaxation. Let* $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \text{proj}_{(\boldsymbol{x},\boldsymbol{y})} R^H(S)$ *such that* $\hat{\boldsymbol{y}}$ *is fractional for* S. *Let* P *be a partition of* D_k *for some*

disjunction $k \in K^F(\hat{\boldsymbol{y}})$*. Partition* P *is a separating partition induced by disjunction* k *and* $\hat{\boldsymbol{x}}$ *if*

$$\forall D \in P : \hat{\boldsymbol{x}} \notin \operatorname{proj}_{\boldsymbol{x}} \left(R^H(\operatorname{child}(S, k, D)) \right).$$

Proposition 3 shows that branching on separating partitions at B&B node S ensures non-decreasing local bounds.

Proposition 3. *Let* S *be a non-trivial convex GDP instance and* $R^H(S)$ *be its hull relaxation. Let* $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \operatorname{proj}_{(\boldsymbol{x},\boldsymbol{y})} \arg\min_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\nu}) \in R^H(S)} \boldsymbol{c}^\top \boldsymbol{x}$*. Assume that* $\hat{\boldsymbol{y}}$ *is fractional for* S *and let* $k \in K^F(\hat{\boldsymbol{y}})$*. If* P *is a separating partition induced by disjunction* k *and* $\hat{\boldsymbol{x}}$ *then* $\min_{D \in P} \min_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\nu}) \in R^H(\operatorname{child}(S,k,D))} \boldsymbol{c}^\top \boldsymbol{x} \geq \boldsymbol{c}^T \hat{\boldsymbol{x}}$*, where we take* $\min_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\nu}) \in \emptyset} \boldsymbol{c}^\top \boldsymbol{x} = \infty$*.*

Proof. For all $D \in P$, $\operatorname{proj}_{\boldsymbol{x}} R^H(\operatorname{child}(S, k, D)) \subseteq \operatorname{proj}_{\boldsymbol{x}} R^H(S)$, hence if the result does not hold then we have a contradiction to the definition of $\hat{\boldsymbol{x}}$. □

Applying Proposition 3 requires the existence of a separating partition for $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$ B&B node S. Proposition 4 shows that qualitative fractionality of disjunction $k \in K$ under $\hat{\boldsymbol{x}}$ is a sufficient condition for existence of a separating partition induced by $k \in K$ and $\hat{\boldsymbol{x}}$.

Proposition 4. *Let* S *be a non-trivial convex GDP instance,* $R^H(S)$ *its hull relaxation and* $\hat{\boldsymbol{x}} \in \operatorname{proj}_{\boldsymbol{x}} R^H(S)$*. If disjunction* $k \in K$ *is qualitatively fractional under* $\hat{\boldsymbol{x}}$ *then there exists a separating partition induced by disjunction* k *and* $\hat{\boldsymbol{x}}$*.*

Proof. The partition $\{\{i\}\}_{i \in D_k}$ is a separating partition induced by disjunction k and $\hat{\boldsymbol{x}}$. □

If disjunction k admits a separating partition for $\hat{\boldsymbol{x}}$ then we may have many candidates. While separation ensures progress with respect to lower bound improvement, successive branching on partitions with large cardinalities may have a multiplicative effect on the number of nodes explored. Definition 6.9 defines minimal separating partitions, a property that results in the smallest local increase in nodes when branching.

Definition 6.9. *Let S be a non-trivial GDP instance and $R^H(S)$ be its hull relaxation. Let $\hat{\boldsymbol{x}} \in \text{proj}_{\boldsymbol{x}} R^H(S)$. Assume that a separating partition exists and let P be a separating partition induced by disjunction $k \in K$ and $\hat{\boldsymbol{x}}$. If P has minimal cardinality among all separating partitions P' induced by disjunction k and $\hat{\boldsymbol{x}}$ then P is a minimal separating partition induced by disjunction k and $\hat{\boldsymbol{x}}$.*

Deriving a Definition 6.9 minimal separating partition P may require checking many partitions. Pairwise irreducibility Definition 6.10 supports the construction separating partitions without checking all possible partitions, Section 6.2.4 explores this idea.

Definition 6.10. *Let S be a non-trivial GDP instance and $R^H(S)$ be its hull relaxation. Let $\hat{\boldsymbol{x}} \in \text{proj}_{\boldsymbol{x}} R^H(S)$. Let P be a separating partition induced by disjunction $k \in K$ and $\hat{\boldsymbol{x}}$. Separating partition P is pairwise irreducible if*

$$\forall D', D'' \in P, D' \neq D'' : \hat{\boldsymbol{x}} \in \text{proj}_{\boldsymbol{x}} R^H(\text{child}(S, k, D' \cup D'')) \,. \tag{6.6}$$

Proposition 5 relates Definitions 6.9 and 6.10 by showing that all minimal separating partitions are pairwise irreducible.

Proposition 5. *Let S be a non-trivial convex GDP instance and $R^H(S)$ be its hull relaxation. Let $\hat{\boldsymbol{x}} \in \text{proj}_{\boldsymbol{x}} R^H(S)$. Let P be a separating partition induced by disjunction $k \in K$ and $\hat{\boldsymbol{x}}$. If P is a minimal separating partition then P is pairwise irreducible.*

Proof. We prove the contrapositive. Assume that P is not pairwise irreducible. Hence, there exists $D', D'' \in P,\ D' \neq D''$ such that $\hat{\boldsymbol{x}} \notin \text{proj}_{\boldsymbol{x}} \left(R^H(\text{child}(S, k, D' \cup D'')) \right)$. Then $P' = \{D' \cup D''\} \cup P \setminus \{D', D''\}$ is a separating partition and $|P'| < |P|$, i.e. P is not minimal. □

A pairwise irreducible separating partition may not be minimal, e.g. we may be able to distribute a subset entirely across the other subsets in the partition. However, pairwise irreducibility provides a form of local minimality with respect to set union.

Table 6.2: Branch construction strategies.

Name	Definition	Description
Greedy	Algorithm 10	Greedily builds branches iteratively by adding next disjunct to first branch that maintains separation. Builds an additional branch if none is found.
Semi-Balanced	Definition 6.11	Similar strategy to *Greedy* except the branches are tested in ascending order of number of disjuncts currently accepted.
Wide (Beaumont, 1990)	Definition 6.12	Constructs a branch for each disjunct.
Semi-wide	Definition 6.13	Combines disjuncts that are not selected into a single branch, uses the *Wide* strategy for remaining disjuncts.

6.2.4 Constructing Branches

Given hull relaxation solution $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$ of non-trivial convex GDP instance S with $\hat{\boldsymbol{y}}$ fractional for S, the B&B algorithm selects fractional disjunction $k \in K$ for branching. Section 6.2.3 discusses desirable properties of child node constructing partition P of D_k to ensure progress in the B&B search. This section discusses how we may construct partition P. We propose two greedy algorithms that constructs partitions satisfying Definition 6.10 pairwise irreducibility when disjunction k is qualitatively fractional under $\hat{\boldsymbol{x}}$. We also propose a naive construction approach that may create less nodes that the Beaumont (1990) strategy. Table 6.2 summarises the branch construction strategies.

Separation oracles. The greedy algorithm iteratively tests the following condition derived from Equation (6.6):

$$\hat{\boldsymbol{x}} \in \operatorname{proj}_{\boldsymbol{x}} R^H(\text{child}(S, k, D')), \tag{6.7}$$

where S is the parent (non-trivial convex GDP) instance, k is the selected disjunction and $D' \subset D_k$, $D' \neq \emptyset$ is the subset of disjuncts under consideration for the current child. We refer to Equation (6.7) as the *branch separation problem*. When assessing the branch separation problem, we have three cases: (i) $R^H(\text{child}(S, k', D'))$ is infeasible, (ii) $R^H(\text{child}(S, k, D'))$ is feasible and $\hat{\boldsymbol{x}} \in R^H(\text{child}(S, k, D'))$, or (iii) $R^H(\text{child}(S, k, D'))$ is feasible and $\hat{\boldsymbol{x}} \notin R^H(\text{child}(S, k, D'))$. We propose two approaches for distinguishing between these cases. The first approach solves at most two feasibility problems whereas the second approach solves a single optimisation problem.

Algorithm 10 Greedy disjunction partitioning

S ▷ Non-trivial convex GDP instance
$(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \text{proj}_{(\boldsymbol{x},\boldsymbol{y})}\, R^H(S)$ ▷ Hull relaxation solution to separate
$k \in K$ ▷ Fractional disjunction (under $\hat{\boldsymbol{y}}$) chosen for branching
$D^j = \emptyset, j = 1, \ldots, |D_{k'}|$
for $i \in \{1, \ldots, |D_{k'}|\}$ **do**
    $j \leftarrow \min\left(\left\{j' \,\middle|\, \hat{\boldsymbol{x}} \notin \text{proj}_{\boldsymbol{x}}\, R^H(\text{child}(S, k, D^{j'} \cup \{i\})), D^{j'} \neq \emptyset\right\} \cup \{j' \mid D^{j'} = \emptyset\}\right)$
    $D^j \leftarrow D^j \cup \{i\}$
end for
return $\{D^j \mid D^j \neq \emptyset\}$

The first approach begins by assessing feasibility of $R^H(\text{child}(S, k, D'))$ if this constraint set is infeasible then we have case (i). If $R^H(\text{child}(S, k, D'))$ is feasible then we assess feasibility of $\hat{\boldsymbol{x}} \in R^H(\text{child}(S, k, D'))$, e.g. feasibility of the constraint system $[\boldsymbol{x} = \hat{\boldsymbol{x}} \wedge R^H(\text{child}(S, k, D'))]$, feasibility and infeasibility of this second problem correspond to cases (ii) and (iii), respectively. The second approach repurposes the cutting plane separation Problem (2.27) (Stubbs and Mehrotra, 1999; Vecchietti et al., 2003). Let S^{sep} denote Problem (2.27) with $R^H(\text{child}(S, k, D'))$ in place of $R'(S)$ in Equation (2.27b). Cases (i), (ii), and (iii) correspond to S^{sep}: being infeasible, and having optimal objective equalling zero, and having positive optimal objective, respectively. In the second approach, case (iii) may be identified prior to an optimality proof if the underlying solution algorithm has a feasible solution and positive objective bound. Since we are separating against disjunction k, it suffices to consider a constraint set of only disjunction k, i.e. the branch separation problem becomes:

$$\hat{\boldsymbol{x}} \in \text{proj}_{\boldsymbol{x}}\, R^H(S_{k',D'}), \tag{6.8}$$

where $S_{k,D'}$ denotes the convex GDP constraints given by Equations (2.15f) and (2.15g) and restricting Equations (2.15c), (2.15d) and (2.15h) to disjunction k and disjuncts D'. Considering the smaller $S_{k,D'}$ constraint set may construct a significantly smaller hull reformulation. Furthermore, when the individual disjuncts are all known to be feasible, case (i) need not be addressed.

Greedy Algorithm. The greedy algorithm described below is parameterised by a separation oracle that assesses Equation (6.7), Equation (6.8) or some intermediate form, i.e. at the

very least the separation oracle has to consider the disjunction given by k. In practice, a mathmatical constraint feasibility checker or a mathematical optimiser can implement this oracle. Algorithm 10 lists a greedy algorithm for constructing a partition of disjunction k disjunct index set D_k in convex GDP instance S. Algorithm 10 builds a partition of D_k by maintaining a list of disjoint subsets (initially all empty). Each disjunct iterates through the disjoint subsets and the separation oracle tests if the subset combined with the new disjunct is a separating subset, i.e. Equation (6.7). The test disjunct is added to the first subset that remains separating under the test disjunct's inclusion. If no such subset is found, the test disjunct forms a new singleton subset. The set of all resulting nonempty subsets forms the partition of the current problem. Algorithm 10 does not have any special handling of infeasibility of a disjunct, if this is required then a single pass checking a feasibility of the individual disjuncts can be applied. Proposition 7 shows that applying Algorithm 10 over a disjunction that is qualitatively fractional under the relaxation solution derives pairwise irreducible partitions. We first prove Proposition 6 which is used in the Proposition 7 proof.

Proposition 6. *Let S be a non-trivial convex GDP instance and $R^H(S)$ be its hull relaxation. Let $\hat{\boldsymbol{x}} \in \operatorname{proj}_{\boldsymbol{x}} R^H(S)$ and disjunction $k \in K$ be qualitatively fractional under $\hat{\boldsymbol{x}}$. Let $D \subset D_k$, $|D| > 1$. If $\hat{\boldsymbol{x}} \notin \operatorname{proj}_{\boldsymbol{x}} R^H(\text{child}(S, k, D))$ then $\forall D' \subseteq D, D' \neq \emptyset : \hat{\boldsymbol{x}} \notin \operatorname{proj}_{\boldsymbol{x}} R^H(\text{child}(S, k, D'))$.*

Proof. We prove the contrapositive. Assume that $D' \subseteq D$, $D' \neq \emptyset$ and $\hat{\boldsymbol{x}} \in \operatorname{proj}_{\boldsymbol{x}} R^H(\text{child}(S, k', D'))$. Hence, there exists $(\hat{\boldsymbol{y}}, \hat{\boldsymbol{\nu}})$ such that $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{\nu}}) \in R^H(\text{child}(S, k', D'))$. Assigning $(\bar{\boldsymbol{y}}, \bar{\boldsymbol{\nu}})$ to

$$\bar{y}_{ik'} = \begin{cases} 0, & i \notin D', k' = k \\ \hat{y}_{ik'}, & \text{otherwise,} \end{cases}, \qquad \bar{\boldsymbol{\nu}}_{\boldsymbol{ik'}} = \begin{cases} \mathbf{0}, & i \notin D', k' = k \\ \hat{\boldsymbol{\nu}}_{\boldsymbol{ik'}}, & \text{otherwise,} \end{cases}$$

satisfies $(\hat{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{\nu}}) \in R^H(\text{child}(S, k, D))$, i.e. $\hat{\boldsymbol{x}} \in \operatorname{proj}_{\boldsymbol{x}} R^H(\text{child}(S, k, D))$. □

Proposition 7. *Let S be a non-trivial convex GDP instance and $R^H(S)$ be its hull relaxation. Let $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in \operatorname{proj}_{(\boldsymbol{x},\boldsymbol{y})} \arg\min_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\nu}) \in R^H(S)} \boldsymbol{c}^\top \boldsymbol{x}$. If disjunction $k \in K$ is qualitatively fractional under $\hat{\boldsymbol{x}}$ then Algorithm 10 parameterised by S, $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$ and k returns a pairwise irreducible separating partition induced by k and $\hat{\boldsymbol{x}}$.*

Proof. Let P denote the set that Algorithm 10 returns. Clearly, P must be a partition of D_k. By construction, every $D \in P$ such that $|D| > 1$ satisfies $\hat{\boldsymbol{x}} \notin \text{proj}_{\boldsymbol{x}} R^H(\text{child}(S, k, D))$. For the remaining (singleton) subsets, i.e. $D \in P$ such that $|D| = 1$, $\hat{\boldsymbol{x}} \notin \text{proj}_{\boldsymbol{x}} R^H(\text{child}(S, k, D))$ follows from qualitative fractionality of k under $\hat{\boldsymbol{x}}$. Hence, P is a separating partition.

We now show pairwise irreducibility. Assume, for a contradiction, that P is not pairwise irreducible. Hence, there exists D^{j_1}, and D^{j_2}, $j_1 \neq j_2$ such that $\hat{\boldsymbol{x}} \notin \text{proj}_{\boldsymbol{x}} R^H(\text{child}(S, k, D^{j_1} \cup D^{j_2}))$. Without loss of generality, we assume that $j_1 < j_2$. Choose the lexicographically minimal such (j_1, j_2). Let $i_2^{\min} = \min D^{j_2}$ and $D_*^{j_1} = \{i \in D^{j_1} \mid i < i_2^{\min}\}$. We have that $D_*^{j_1} \neq \emptyset$, since $j_1 < j_2$, and D^{j_2}, in Algorithm 10, is initialised by $i_2^{\min}$, hence there must be $i \in D^{j_1}$ satisfying $i < i_2^{\min}$. By Proposition 6, $\hat{\boldsymbol{x}} \notin R^H(\text{child}(S, k, D_*^{j_1} \cup \{i_2^{\min}\}))$. By choice of j_1 and j_2, Algorithm 10 must also encounter this test and therefore $i_2^{\min} \in D^{j_1}$. But this contradicts P being a partition of D_k. □

When disjunction k is not qualitatively fractional under $\hat{\boldsymbol{x}}$, Algorithm 10 returns a partition P of D_k such that (i) there exists $P' \subseteq P$ containing only singleton sets and $\forall D \in P' : \hat{\boldsymbol{x}} \in \text{proj}_{\boldsymbol{x}} R^H(\text{child}(S, k, D))$ and (ii) $\forall D \in P \setminus P' : \hat{\boldsymbol{x}} \notin \text{proj}_{\boldsymbol{x}} R^H(\text{child}(S, k, D))$.

Greedy Algorithm 10 places disjuncts into the first subset that retains separability. This approach could lead earlier subsets being more populated than the later subsets. Hence, the child nodes associated with larger subsets in the partition may construct larger hull relaxations. To encourage a partition that contains subsets that closer to equal in size, we propose adapting Algorithm 10 on line 6 to test against subsets $\{D^j \mid D^j \neq \emptyset\}$ in ascending order of their size. Making this change does not affect pairwise irreducibility of the resulting partition. We call this alternative the Definition 6.11 *semi-balanced greedy* branch construction strategy.

Definition 6.11. *The semi-balanced greedy branch construction strategy refers to the algorithm given by adapting line 6 of Algorithm 10 to consider the nonempty subsets in increasing size order.*

An alternative to a greedy approach is to branch entirely on the selected disjunction k (Beaumont, 1990), i.e. $P = \{\{i\} \mid i \in D_k\}$, we call this approach the Definition 6.12 *wide* branch construction

strategy.

Definition 6.12 (Beaumont (1990))**.** *Let $k \in K^F(\hat{\boldsymbol{y}})$ be the disjunction selected for branching. The wide branch construction strategy creates partition $P = \{\{i\} \mid i \in D_k\}$ of disjunction k to form subproblems.*

A wide branch construction strategy may construct many child nodes. The Definition 6.13 *semi-wide* branching strategy differs from the wide branching strategy by collecting disjuncts that are not selected, i.e. $\hat{y}_{ik} = 0$ for non-selected disjunct (i, k), into the same subset. As a heuristic, the semi-wide branching strategy groups disjuncts that do not improve the relaxation objective (otherwise they would be partially selected) and therefore the subset containing the non-selected disjuncts may increase the local lower bound in the branch that they form. Note that the semi-wide strategy does not necessarily form a separating partition.

Definition 6.13. *Let $k \in K^F(\hat{\boldsymbol{y}})$ be the disjunction selected for branching. The semi-wide branch construction strategy creates partition $P = \{\{i\} \mid y_{ik} > 0, i \in D_k\} \cup \{\{i \in D_k \mid y_{ik} = 0\}\}$ of disjunction k to form subproblems.*

6.3 Leveraging Infeasibility in Convex Generalized Disjunctive Programming

Any constraint set that admits no solutions is an infeasible system of constraints. Such a constraint set may also be referred to as a conflicting constraint set. Conflict-driven clause-learning (CDCL) is a topic that is widely applied in propositional satifiability (Marques-Silva and Sakallah, 1996; Zhang et al., 2001; Marques-Silva et al., 2009). When CDCL solver encounters a local infeasibility, it derives additional propositional constraints that prevent the DPLL search algorithm (Davis and Putnam, 1960; Davis et al., 1962) from encountering the same infeasibility again. Achterberg (2007a) adapts these propositional satisfiability-based techniques for mixed-integer programming.

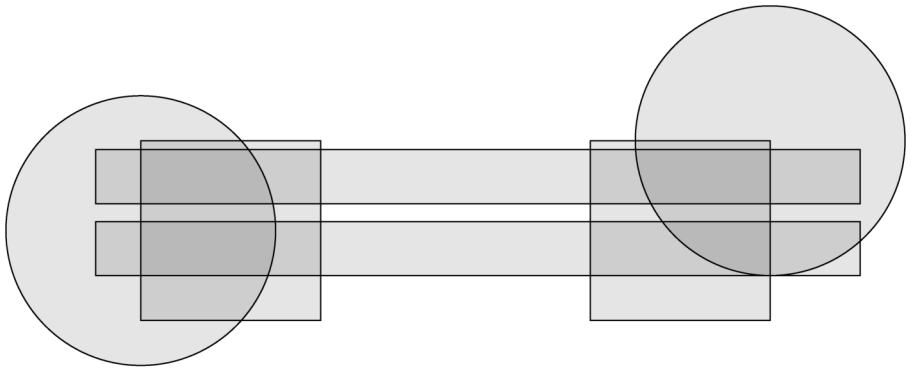


Figure 6.2: Convex generalized disjunctive programming feasible region that supports Example 6.2. The instance contains 3 disjunctions each having 2 disjuncts. The disjunctions are identified by matching shapes. The darkest shaded regions, i.e. where a circle, square and oblong simultaneously overlap, are the integer feasible regions.

B&B algorithms generally encounter infeasibility when a local continuous relaxation becomes infeasible. The same infeasibility can occur in multiple B&B nodes, Example 6.2 describes such a case.

Example 6.2. *Consider the convex GDP feasible region given by Figure 6.2. There are two distinct partial assignments that cause a conflict: (1) the left circle and right square or (2) the right circle and the left square. Hence, selection of an oblong region is independent of the conflict causing selections. If a B&B algorithm branches on the oblong regions first, conflicts (1) and (2) will both occur twice in the B&B search tree. Assume that the B&B algorithm selects the top oblong region first and in the following search encounters conflict (1). With analysis, we may derive the condition 'the left circle and right square cannot occur together' and strengthen the B&B algorithm. Now when the B&B search considers the bottom oblong path and fixes either the left circle or the right square, the conflict condition immediately allows the B&B algorithm to locally discard the right square or the left circle, respectively.*

Throughout this section, we assume that the constraint set given by Equations (2.15b), (2.15f) and (2.15g) is feasible. A hull relaxation does not weaken these constraints. Hence, infeasibility between these constraints would be detected at the root node of the B&B tree, i.e. an immediate conclusion that the convex GDP is infeasible. Furthermore, we use *subcombination* Definition 6.14 to generate subproblems of convex GDP S, isolate infeasible combinations of disjuncts, and to simplify our notation. A subcombination selects a subset of disjunctions and for each of these

disjunctions considers a subset of disjuncts.

Definition 6.14. *Let S be a non-trivial convex GDP instance without any propositional constraints. Let $\mathcal{D} = \{(k, D'_k) \mid k \in K'\}$ where $K' \subseteq K$ and $D'_k \subseteq D_k$. If $D \neq \emptyset$ for all $(k, D) \in \mathcal{D}$ then we call $\mathcal{D}$ a subcombination of S. The following convex GDP constraint set is the subproblem of S induced by subcombination $\mathcal{D}$.*

$$
\begin{aligned}
& g_j(\boldsymbol{x}) \leq 0, && j \in [p] \\
& r_{ikj}(\boldsymbol{x}) \leq 0, && (k, D'_k) \in \mathcal{D}, |D'_k| = 1, i \in D'_k, j \in [p_{ik}] \\
& \bigvee_{i \in D'_k} \begin{bmatrix} Y_{ik} \\ r_{ikj}(\boldsymbol{x}) \leq 0, \quad j \in [p_{ik}] \end{bmatrix}, && D'_k \in \mathcal{D}, |D'_k| > 1, \\
& \veebar_{i \in D'_k} Y_{ik}, && (k, D'_k) \in \mathcal{D}, |D'_k| > 1 \\
& \boldsymbol{x}^{\boldsymbol{L}} \leq \boldsymbol{x} \leq \boldsymbol{x}^{\boldsymbol{U}}, \\
& \boldsymbol{x} \in \mathbb{R}^n \\
& Y_{ik} \in \{\mathit{True}, \mathit{False}\}, && (k, D'_k) \in \mathcal{D}, |D'_k| > 1, i \in D'_k.
\end{aligned}
$$

We use $S_{\mathcal{D}}$ to denote the subproblem of S induced by subcombination $\mathcal{D}$.

Subcombinations treat disjuncts as atomic units. We will use this property to generate conflict clauses (no-good cuts) over the Boolean (binary) variables attached to disjuncts (Silva and Sakallah, 1996; Balas and Jeroslow, 1972; Stallman and Sussman, 1977; Achterberg, 2007a).

6.3.1 Conflicts in Convex Generalized Disjunctive Programming

This section discusses how (partial) assignments that cause a convex GDP to become infeasible may be analysed and propagated as a conflict clause. We note that both branching and applying basic steps can cause a convex GDP to become infeasible. In the case of basic steps, an infeasible combination of disjuncts may be detected without the entire convex GDP instance becoming infeasible (Sawaya and Grossmann, 2012; Ruiz and Grossmann, 2012). Algorithm 9 does not

consider the application of basic steps, hence conflicts only occur under branching and are detected when the problem becomes infeasible.

Proposition 8 states how conflicting combinations of disjuncts can be extracted from a conflicting set of disjunctions and global constraints, i.e. an infeasible convex GDP.

Proposition 8. *Let S be a non-trivial convex GDP instance without any propositional constraints. Let $\mathcal{D}$ be a subcombination of S. Let* $\text{zip}(k, D) = \{(k, \{i\}) | i \in D\}$. *If $S_{\mathcal{D}} = \emptyset$ then for all* $I \in \prod_{(k,D)\in\mathcal{D}} \text{zip}(k, D)$, $S_I = \emptyset$.

Proof. We prove the contrapositive. Assume that there exists $\hat{\boldsymbol{x}} \in S_{I'}$ for some $I' \in \prod_{(k,D)\in\mathcal{D}} \text{zip}(k, D)$ ($S_{I'}$ does not have Boolean variables by construction). Then we construct the assignment $\hat{\boldsymbol{Y}}$ in $S_{\mathcal{D}}$ as:

$$\hat{Y}_{ik} = \begin{cases} \text{True}, & (k, \{i\}) \in I' \\ \text{False}, & \text{otherwise.} \end{cases}$$

We have that $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{Y}}) \in S_{\mathcal{D}}$. □

The conflicting combinations of disjuncts that Proposition 8 derives are converted into conflict clauses using Proposition 9. We first state Definition 6.15 which introduces the *disjunct selecting* property for subcombinations. Disjunct selecting subcombinations are those where each disjunction in the subcombination is reduced to a single disjunct, i.e. the problem induced by a disjunct selecting subcombination is continuous.

Definition 6.15. *Let S be a convex GDP instance without any propositional constraints. Subcombination $\mathcal{D}$ of S is disjunct selecting if for all $(k, D) \in \mathcal{D} : |D| = 1$.*

Proposition 9 derives conflict clauses from infeasible disjunct selecting subcombinations of S. We note that conflict clauses of the Proposition 9 type are well-known (Silva and Sakallah, 1996; Achterberg, 2007a) and can be derived with automated tools such as SMT (Bjørner and De Moura, 2011; de Moura and Jovanović, 2013), we provide a proof here in the convex GDP context for completeness.

Proposition 9 (Silva and Sakallah (1996); Achterberg (2007a))**.** *Let S be a non-trivial convex GDP instance without any propositional constraints. Let $\mathcal{D}$ be a disjunct selecting subcombination of S. If $S_{\mathcal{D}}$ is infeasible, then*

$$\bigvee_{(k,\{i\})\in\mathcal{D}} \neg Y_{ik} \tag{6.9}$$

is valid for S.

Proof. We prove the contrapositive. If Equation (6.9) is not valid then it excludes some feasible solution $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{Y}}) \in S$. Hence, $g_j(\hat{\boldsymbol{x}}) \leq 0$, $j \in [p]$ and $\hat{\boldsymbol{x}} \in [\boldsymbol{x}^{\boldsymbol{L}}, \boldsymbol{x}^{\boldsymbol{U}}]$. Since $\hat{\boldsymbol{Y}}$ satisfies the negation of Equation (6.9), i.e. $\hat{Y}_{ik} = \text{True}$, for all $(k, \{i\}) \in \mathcal{D}$, $r_{ikj}(\hat{\boldsymbol{x}}) \leq 0$, for all $(k, \{i\}) \in \mathcal{D}, j \in [p_{ik}]$. Hence, $\hat{\boldsymbol{x}} \in S_{\mathcal{D}}$. □

Proposition 9 shows that the problem S constraint set may be extended by Equation (6.9) without affecting feasibility. Let S^{root} denote the root node constraint set in the B&B tree. Every B&B node is associated with a subcombination of S^{root}, i.e. for each B&B node S there exists a subcombination $\mathcal{D}$ of S^{root} such that $S = S^{\text{root}}_{\mathcal{D}}$. If $R^H(S^{\text{root}}_{\mathcal{D}}) = \emptyset$, i.e. the local relaxation is infeasible, then Propositions 8 and 9 motivate propagating this infeasibility. However, applying Propositions 8 and 9 under subcombination $\mathcal{D}$ only covers descendant nodes of $S^{\text{root}}_{\mathcal{D}}$. Since $S^{\text{root}}_{\mathcal{D}}$ and all descendant nodes are discarded from the search, this particular propagation has no effect. If, however, we can find $\mathcal{D}' \subset \mathcal{D}$, $\mathcal{D}' \neq \emptyset$ such that $S^{\text{root}}_{\mathcal{D}'} = \emptyset$, then applying Propositions 8 and 9 under subcombination $\mathcal{D}'$ may affect unexplored nodes. Section 6.3.2 analyses how an irreducible infeasible subsystem (IIS) deriving solver may be leveraged for deriving such a $\mathcal{D}'$.

6.3.2 Hull Relaxation Irreducible Infeasible Subsystems in Convex Generalized Disjunctive Programming

B&B Algorithm 9 encounters infeasibility when solving hull relaxations at a given B&B node. Convex MINLP solvers may derive irreducible infeasible subsystems (IISs) (Chakravarti, 1994). An IIS is a subset of constraints from a mathematical program such that (i) the constraint set is infeasible and (ii) removal of any constraint causes the system to become feasible.

IISs applications include motion planning, classification, and treatment planning (Lagriffoul and Andres, 2016; Chinneck, 2019). An infeasible problem instance may have multiple IISs. Permuting the constraint set of an infeasible instance can result in the derivation of alternative IISs, e.g. using a greedy approach as suggested by Hooker (2007). The B&B algorithm only performs infeasibility analysis on a single IIS in a given infeasible node. If an additional IIS exists within the current node, it may be handled later in the search if it causes a further infeasibility. Example 6.3 motivates how we utilise an IIS from an infeasible hull relaxation result.

Example 6.3. *Consider the Figure 6.2 non-trivial convex GDP constraint system. Assume that the current B&B node follows the root-to-node path: [*`select left circle`*,* `select top oblong`*,* `select right square`*] and let the subcombination of S^{root} associated with this node be $\mathcal{D}$. The hull relaxation is infeasible at this node. Clearly, any infeasible hull relaxation subset of constraints that are associated with the top oblong can be removed without affecting infeasibility. Hence, no IIS will include any constraints associated with the top oblong. We can conclude that this IIS is also associated with some $\mathcal{D}' \subset \mathcal{D}$.*

Definition 6.16 characterises *well-formed* non-trivial convex GDPs, these are convex GDPs whose global constraints are feasible and each individual disjunct is feasible. If the global constraints are infeasible in convex GDP problem S, then $R^H(S)$ is infeasible. We can check feasibility of the individual disjuncts by assessing feasibility of the each disjunct separately. Hence, local infeasibilities in well-formed convex GDPs without propositional constraints always involve constraints from at least one disjunct.

Definition 6.16. *Let S be a non-trivial convex GDP instance. Let $\bar{S}$ be S without any propositional constraints. Let S^* denote S without its global constraints. If for all $k \in K$, $i \in D_k$, $\bar{S}^*_{(k,\{i\})} \neq \emptyset$ and $\bar{S}_\emptyset \neq \emptyset$ then we call S well-formed.*

Proposition 11 states necessary and sufficient conditions for a subcombination to remain infeasible with respect to a given infeasible well-formed convex GDP instance without propositional constraints S and an associated IIS associated with its $R^H(S)$ constraint set. We first prove

Proposition 10 which shows that the hull relaxation of a single disjunction is infeasible if and only if its disjuncts are all infeasible. Proposition 10 is required for the Proposition 11 proof.

Proposition 10. *Let S be a non-trivial convex GDP instance without any propositional constraints and $\mathcal{D} = \{(k, D'_k)\}$ be a subcombination for S for some $k \in K$, and $D'_k \subseteq D_k$. $R^H(S^*_{\mathcal{D}}) = \emptyset$ if and only if for all $i \in D'_k$, $S^*_{\{(k,\{i\})\}} = \emptyset$.*

Proof. For both cases, we show the contrapositive. Let $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{\nu}}) \in R^H(S^*_{\mathcal{D}})$ then by Equation (2.17c) there exists $i' \in D'_k$ with $\hat{y}_{i'k} > 0$. This implies that $\hat{\boldsymbol{\nu}}_{\boldsymbol{i'k}}/\hat{y}_{i'k} \in S^*_{\{(k,\{i'\})\}}$, i.e. the if direction. For the only-if direction, let $i' \in D'_k$ be such that there exists $\hat{\boldsymbol{x}} \in S^*_{\{(k,\{i\})\}}$, then $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{\nu}}) \in R^H(S^*_{\mathcal{D}})$ where

$$\hat{y}_{ik} = \begin{cases} 1, & i = i' \\ 0, & \text{otherwise,} \end{cases} \qquad \hat{\boldsymbol{\nu}}_{\boldsymbol{ik}} = \begin{cases} \hat{\boldsymbol{x}}, & i = i' \\ \mathbf{0}, & \text{otherwise.} \end{cases}$$

□

For Propositions 11 and 12 let $\text{disagg}(R^H(S), k)$ denote the disjunction $k \in K$ instance of Equation (2.22b) in the $R^H(S)$ constraint set.

Proposition 11. *Let S be a non-trivial well-formed convex GDP instance without any propositional constraints. Let $R^H(S)$ be infeasible and $\mathcal{I}$ be an IIS of the $R^H(S)$ constraint set. Let $\mathcal{R} = \{k \in K \mid \text{disagg}(R^H(S), k) \in \mathcal{I}\}$ and $\mathcal{D} = \{(k, D_k) \mid k \in \mathcal{R}\}$. Let $\mathcal{D}'$ be a subcombination of S. Then $\mathcal{I}$ is contained in the $R^H(S_{\mathcal{D}'})$ constraint set if and only if $\mathcal{D} \subseteq \mathcal{D}'$.*

Proof. We begin with the if case. Assume $\mathcal{D} \subseteq \mathcal{D}'$. We have that all constraints in $R^H(S_{\mathcal{D}})$ are contained in $R^H(S_{\mathcal{D}'})$, since the latter extends the former. Hence, showing that $\mathcal{I}$ is contained in the $R^H(S_{\mathcal{D}})$ constraint set suffices for the result to hold. Let $\mathcal{R}' \subseteq K \setminus \mathcal{R}$ be such that $k' \in \mathcal{R}'$ if $\mathcal{I}$ contains at least one disjunction k' constraint associated with Equations (2.17c), (2.22a), (2.22c), or (2.24). Assume that $\mathcal{I}$ is not contained in the $R^H(S_{\mathcal{D}})$ constraint set. Hence, $\mathcal{R}' \neq \emptyset$. We partition $\mathcal{I}$ into (i) $\mathcal{I}_g$ which contains all global or variable bounding constraints from the $R^H(S)$ constraint set that are also contained in $\mathcal{I}$ and (ii) $\mathcal{I}_k$, for each

$k \in \mathcal{R} \cup \mathcal{R}'$ which contain all constraints associated with disjunction k that are also contained in $\mathcal{I}$. We will now consider the following cases: (1) $|\mathcal{R}'| > 1$, (2) $|\mathcal{R}'| = 1$ and $\mathcal{I}_g \cup \bigcup_{k\in\mathcal{R}} \mathcal{I}_k \neq \emptyset$ and (3) $|\mathcal{R}'| = 1$ and $\mathcal{I}_g \cup \bigcup_{k\in\mathcal{R}} \mathcal{I}_k = \emptyset$. Assume case (1) and consider the sets $\mathcal{I}_k$, for $k \in \mathcal{R}'$. Since $\text{disagg}(R^H(S), k) \notin \mathcal{I}_k$ for all $k \in \mathcal{R}'$, no two constraints in $\mathcal{I}_{k_1}$ and $\mathcal{I}_{k_2}$, for $k_1, k_2 \in \mathcal{R}', k_1 \neq k_2$ have any variables in common, i.e. the constraints in these sets cannot affect feasibility/infeasibility of each other. Similarly, the constraints in $\mathcal{I}_k$ for all $k \in \mathcal{R}'$ cannot affect feasibility/infeasibility of the system given by $\mathcal{I}_g \cup \bigcup_{k\in\mathcal{R}} \mathcal{I}_k$. We conclude that $\mathcal{I} \setminus \mathcal{I}_{k'}$ remains infeasible for some $k' \in \mathcal{R}'$, i.e. a contradiction to irreducibility of $\mathcal{I}$. A similar argument to case (1) shows that case (2) also contradicts irreducibility of $\mathcal{I}$. We now assume case (3). By Proposition 10, this case contradicts S being well-formed. Since cases (1) to (3) all lead to a contradiction, we conclude that $\mathcal{R}' = \emptyset$ which contradicts our initial assumption. Hence, we have that $\mathcal{I}$ is contained in the $R^H(S_\mathcal{D})$ constraint set.

We show the only-if case by proving the contrapositive. Assume that $\mathcal{D} \not\subseteq \mathcal{D}'$. Hence, $(k', D_{k'}) \in \mathcal{D}$ and $(k', D_{k'}) \notin \mathcal{D}'$ for some $k' \in \mathcal{R}$. Then the $R^H(S_{\mathcal{D}'})$ constraint set does not include $\text{disagg}(R^H(S), k')$, whereas $\text{disagg}(R^H(S), k') \in \mathcal{I}$. We conclude that $\mathcal{I}$ is not contained in the $R^H(S_{\mathcal{D}'})$ constraint set. □

Proposition 12 derives a subcombination that maintains infeasibility for well-fromed convex GDP S, where S is a node of the B&B Algorithm 9 search tree. In particular, Proposition 12 shows that presence of a disjunction's Equation (2.22b) disaggregation sum constraint in the IIS is sufficient for identifying whether the disjunction participates in the infeasibility.

Proposition 12. *Let S^{root} be a non-trivial well-formed convex GDP instance without any propositional constraints and $\mathcal{D}'$ a subcombination of S^{root}. Assume that S^{root} is the root node instance of B&B Algorithm 9 and $\mathcal{D}'$ the subcombination associated with some B&B node. If $R^H(S^{root}_{\mathcal{D}'}) = \emptyset$ then $S^{root}_{\mathcal{D}} = \emptyset$, where $\mathcal{I}$ is an IIS contained in the $R^H(S^{root}_{\mathcal{D}'})$ constraint set, $\mathcal{R}_1 = \{k \in K \,|\, (k, D'_k) \in \mathcal{D}', |D'_k| = 1, r_{ikj} \in \mathcal{I}, \text{for some } j \in [p_{ik}]\}$, $\mathcal{R}_2 = \{k \in K \,|\, \text{disagg}(R^H(S), k) \in \mathcal{I}, (k, D'_k) \in \mathcal{D}', |D'_k| > 1\}$ and $\mathcal{D} = \{(k, D'_k) \in \mathcal{D}' \,|\, k \in \mathcal{R}_1 \cup \mathcal{R}_2\}$.*

Proof. We consider the cases: (1) $\mathcal{R}_1 = \emptyset$, (2) $\mathcal{R}_1 \neq \emptyset$ and $\mathcal{R}_2 = \emptyset$, and (3) $\mathcal{R}_1 \neq \emptyset$ and $\mathcal{R}_2 \neq \emptyset$.

We begin with case (1). Assume that (non-relaxed) $S^{\text{root}}_{\mathcal{D}} \neq \emptyset$. Hence, there exists $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{Y}}) \in S^{\text{root}}_{\mathcal{D}}$. Let

$$\hat{y}_{ik} = \begin{cases} 1, & \hat{Y}_{ik} = \text{True} \\ 0, & \text{otherwise,} \end{cases} \qquad \hat{\boldsymbol{\nu}}_{\boldsymbol{ik}} = \begin{cases} \hat{\boldsymbol{x}}, & \hat{Y}_{ik} = \text{True} \\ \mathbf{0}, & \text{otherwise.} \end{cases}$$

Clearly, $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{\nu}}) \in R^H(S^{root}_{\mathcal{D}})$. By Proposition 11, $\mathcal{I}$ is contained in the $R^H(S^{\text{root}}_{\mathcal{D}})$ constraint set. Since, $R^H(S^{\text{root}}_{\mathcal{D}})$ is feasible, $\mathcal{I}$ must be feasible. We have a contradiction to $\mathcal{I}$ being an IIS. We now show case (2). We have that if $\mathcal{I}$ has constraints associated with some disjunction k such that $(k, D'_k) \in \mathcal{D}'$, $|D'_k| > 1$ other than disagg$(R^H(S^{\text{root}}_{\mathcal{D}'}), k)$, then $\mathcal{I}$ is not irreducible. Hence, by construction $S^{\text{root}}_{\mathcal{D}}$ is a trivial convex GDP and therefore $\mathcal{I}$ must be a subset of the $S^{\text{root}}_{\mathcal{D}}$ constraint set, i.e. $S^{\text{root}}_{\mathcal{D}} = \emptyset$. We now show case (3). We have that the disjunctions associated with $\mathcal{R}_1$ construct additional global constraints in $S^{\text{root}}_{\mathcal{D}}$. Consider the global constraints and the additional constraints given by $\mathcal{R}_1$ in $S^{\text{root}}_{\mathcal{D}}$. This constraint set is feasible or infeasible. If this constraint set is infeasible then the result clearly follows. If this constraint set is feasible then $S^{\text{root}}_{\mathcal{D}}$ is well-formed and a similar argument to case (1) holds. □

Combining Propositions 8, 9 and 12 derives conflict clauses from hull reformulations of convex GDPs. We include an Equation (6.9) conflict clause into a hull reformulation by converting it to a *no-good* cut (Balas and Jeroslow, 1972; Stallman and Sussman, 1977; Achterberg, 2007a):

$$\sum_{(k,\{i\})\in\mathcal{D}} y_{ik} \leq |\mathcal{D}| - 1, \tag{6.10}$$

where $\mathcal{D}$ is a disjunct selecting subcombination as defined by Proposition 9.

Propositions 8, 9 and 12 assume a convex GDP instance that does not have any propositional constraints. While the initial instance, i.e. S^{root}, may not contain propositional constraints, applying Proposition 9 introduces propositional constraints to any applicable descendant node. We note that the Proposition 12 result still holds if the hull relaxation IIS does not contain any propositional constraints or any Equation (6.10) no-good cuts.

6.4 Constrained Layout Problem

This section formulates the constrained layout problem for our Section 6.5 numerical results. The constrained layout problem (CLay) is (Sawaya, 2006):

> Let n_c circles have centres $(c_{1,j}, c_{2,j})$ and radius r_j, $j \in [n_c]$. Given n_r rectangular items and costs $F_{i_1,i_2} \geq 0$ associated with each pair $i_1, i_2 \in [n_r]$, $i_1 < i_2$. Find a placement for the rectangles such that each rectangle is contained inside a circle, no two distinct rectangles overlap and the sum of F_{i_1,i_2}-weighted L_1-distances between the centres of rectangles i_1, i_2 is minimised.

6.4.1 Convex Generalized Disjunctive Programming Formulation

There are two types of disjunctions in CLay. The first type enforces that a distinct pair of rectangles do not overlap, or equivalently that they are place above/below one another or left/right of each other. The second type enforce that a particular rectangle is contained within the boundary of some circle $j \in [n_c]$. Table 6.3 list the CLay model parameters and variables.

To aid readability, we state the convex GDP constraint associated with the first type of disjunction below separately (split on multiple lines):

$$\begin{bmatrix} Y^r_{i_1,i_2,1} \\ x_{1,i_1} + \frac{W_{i_1}}{2} \leq x_{1,i_2} - \frac{W_{i_2}}{2} \end{bmatrix} \vee \begin{bmatrix} Y^r_{i_1,i_2,2} \\ x_{1,i_2} + \frac{W_{i_2}}{2} \leq x_{1,i_1} - \frac{W_{i_1}}{2} \end{bmatrix} \\ \vee \begin{bmatrix} Y^r_{i_1,i_2,3} \\ x_{2,i_1} + \frac{H_{i_1}}{2} \leq x_{2,i_2} - \frac{H_{i_2}}{2} \end{bmatrix} \vee \begin{bmatrix} Y^r_{i_1,i_2,4} \\ x_{2,i_2} + \frac{H_{i_2}}{2} \leq x_{2,i_1} - \frac{H_{i_1}}{2} \end{bmatrix} \tag{6.11}$$

A CLay convex GDP formulation is (Sawaya, 2006):

$$\min \quad \sum_{\substack{i_1,i_2 \in [n_r] \\ i_1 < i_2}} F_{i_1,i_2} \left(d_1^{i_1,i_2} + d_2^{i_1,i_2} \right) \tag{6.12a}$$

$$\text{subject to} \quad d_{1,i_1,i_2} \geq x_{1,i_1} - x_{1,i_2}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2 \tag{6.12b}$$

Table 6.3: Model symbols for the constrained layout (CLay) problem.

Name	Description
Sets	
$i \in [n_r]\}$	Rectangle indices
$j \in [n_c]\}$	Circle indices
Parameters	
F_{i_1,i_2}	Objective cost on L_1-distance between the centres of rectangle i_1 and i_2
W_i, H_i	Width and height of rectangle i respectively
$c_{1,j}$, $c_{2,j}$	Coordinates of circle j centre
r_j	Radius of circle j
Variables	
$x_{1,i}$, $x_{2,i}$	Coordinates of rectange i centre
d_{1,i_1,i_2}, d_{2,i_1,i_2}	Auxiliary variables for representing L_1-distance
$Y^r_{i_1,i_2,j}$	Disjunct selection variable enforcing that rectangle i_1 is $\{\text{left of}, \text{right of}, \text{below}, \text{above}\}$ i_2 for $j \in \{1,2,3,4\}$, respectively
$Y^c_{i,j}$	Disjunct selection variable assigning rectangle i to circle j

$$d_{1,i_1,i_2} \geq x_{1,i_2} - x_{1,i_1}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2 \qquad (6.12c)$$

$$d_{2,i_1,i_2} \geq x_{2,i_1} - x_{2,i_2}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2 \qquad (6.12d)$$

$$d_{2,i_1,i_2} \geq x_{2,i_2} - x_{2,i_1}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2 \qquad (6.12e)$$

Equation (6.11), $\qquad \forall i_1, i_2 \in [n_r], i_1 < i_2 \qquad (6.12f)$

$$\underset{j \in \{1,2,3,4\}}{\veebar} Y^r_{i_1,i_2,j}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2 \qquad (6.12g)$$

$$\bigvee_{j \in [n_c]} \begin{bmatrix} Y^c_{i,j} \\ \left(x_{1,i} - \frac{W_i}{2} - c_{1,j}\right)^2 + \left(x_{2,i} + \frac{H_i}{2} - c_{2,j}\right)^2 \leq r_j^2 \\ \left(x_{1,i} - \frac{W_i}{2} - c_{1,j}\right)^2 + \left(x_{2,i} - \frac{H_i}{2} - c_{2,j}\right)^2 \leq r_j^2 \\ \left(x_{1,i} + \frac{W_i}{2} - c_{1,j}\right)^2 + \left(x_{2,i} + \frac{H_i}{2} - c_{2,j}\right)^2 \leq r_j^2 \\ \left(x_{1,i} + \frac{W_i}{2} + c_{1,j}\right)^2 + \left(x_{2,i} - \frac{H_i}{2} - c_{2,j}\right)^2 \leq r_j^2 \end{bmatrix}, \qquad \forall i \in [n_r] \qquad (6.12h)$$

$$\underset{j \in [n_c]}{\veebar} Y^c_{i,j}, \qquad \forall i \in [n_r] \qquad (6.12i)$$

$$x^L_{1,i} \leq x_{1,i} \leq x^U_{1,i}, \quad x^L_{2,i} \leq x_{2,i} \leq x^U_{2,i}, \qquad \forall i \in [n_r] \qquad (6.12j)$$

$$d_{1,i_1,i_2}, d_{2,i_1,i_2} \in \mathbb{R}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2 \qquad (6.12k)$$

$$x_{1,i}, x_{2,i} \in \mathbb{R}, \qquad \forall i \in [n_r] \qquad (6.12l)$$

$$Y^r_{i_1,i_2,1}, Y^r_{i_1,i_2,2}, Y^r_{i_1,i_2,3}, Y^r_{i_1,i_2,4} \in \{\text{True}, \text{False}\}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2 \qquad (6.12m)$$

$$Y^c_{i,j} \in \{\text{True}, \text{False}\}, \qquad \forall i \in [n_r], j \in [n_c]. \quad \text{(6.12n)}$$

Parameters W_i, H_i are the width and height of rectangle $i \in [n_r]$, respectively. Variables $(x_{1,i}, x_{2,i})$ are the coordinates of the centre of rectangle $i \in [n_r]$. Equation (6.12a) minimises the total sum of weighted L_1-distances between the centres of rectangles, auxiliary variables $d_{1,i_1,i_2}, d_{2,i_1,i_2}$ are used to model the L_1-distance. Equations (6.12b) to (6.12e) model the L_1-distances (note that correctness follows from the fact that the objective minimises). Equations (6.12f) and (6.12g) ensure that no distinct pairs of rectangles overlap. Equations (6.12h) and (6.12i) ensure that each rectangle is contained in a circle. Equations (6.12j) to (6.12n) are variable bounds and domains.

Basic Stepped Model

Algorithm 9 branches on the disjunctions and propagates infeasibility proofs. Hence, Algorithm 9 only affects the objective bound through these two operations. In CLay, the objective only depends on the Equation (6.12l) auxiliary variables. These variables only interact with the objective. The branching and infeasibility propagation interacts indirectly with the objective.

A basic step between Equations (6.12b) to (6.12f) for a particular $i_1, i_2 \in [n_r], i_1 < i_2$ constructs:

$$\begin{bmatrix} Y^r_{i_1,i_2,1} \\ x_{1,i_1} + \frac{W_{i_1}}{2} \leq x_{1,i_2} - \frac{W_{i_2}}{2} \\ d_{1,i_1,i_2} \geq x_{1,i_1} - x_{1,i_2} \\ d_{1,i_1,i_2} \geq x_{1,i_2} - x_{1,i_1} \\ d_{2,i_1,i_2} \geq x_{2,i_1} - x_{2,i_2} \\ d_{2,i_1,i_2} \geq x_{2,i_2} - x_{2,i_1} \end{bmatrix} \vee \begin{bmatrix} Y^r_{i_1,i_2,2} \\ x_{1,i_2} + \frac{W_{i_2}}{2} \leq x_{1,i_1} - \frac{W_{i_1}}{2} \\ d_{1,i_1,i_2} \geq x_{1,i_1} - x_{1,i_2} \\ d_{1,i_1,i_2} \geq x_{1,i_2} - x_{1,i_1} \\ d_{2,i_1,i_2} \geq x_{2,i_1} - x_{2,i_2} \\ d_{2,i_1,i_2} \geq x_{2,i_2} - x_{2,i_1} \end{bmatrix} \tag{6.13}$$
$$\vee \begin{bmatrix} Y^r_{i_1,i_2,3} \\ x_{2,i_1} + \frac{H_{i_1}}{2} \leq x_{2,i_2} - \frac{H_{i_2}}{2} \\ d_{1,i_1,i_2} \geq x_{1,i_1} - x_{1,i_2} \\ d_{1,i_1,i_2} \geq x_{1,i_2} - x_{1,i_1} \\ d_{2,i_1,i_2} \geq x_{2,i_1} - x_{2,i_2} \\ d_{2,i_1,i_2} \geq x_{2,i_2} - x_{2,i_1} \end{bmatrix} \vee \begin{bmatrix} Y^r_{i_1,i_2,4} \\ x_{2,i_2} + \frac{H_{i_2}}{2} \leq x_{2,i_1} - \frac{H_{i_1}}{2} \\ d_{1,i_1,i_2} \geq x_{1,i_1} - x_{1,i_2} \\ d_{1,i_1,i_2} \geq x_{1,i_2} - x_{1,i_1} \\ d_{2,i_1,i_2} \geq x_{2,i_1} - x_{2,i_2} \\ d_{2,i_1,i_2} \geq x_{2,i_2} - x_{2,i_1} \end{bmatrix}$$

The Equation (6.13) disjunction interacts directly with the objective via the Equation (6.12l) auxiliary variables. Furthermore, Equation (6.13) does not grow with respect to the number of disjuncts. Hence, replacing Equations (6.12b) to (6.12f) with Equation (6.13) in Problem (6.12) does not cause the relaxation to grow too much (the corresponding hull relaxation does grow due to additional constraints and disagregated variables). We consider the equivalent CLay problem:

$$\begin{aligned} \min \quad & \sum_{\substack{i_1,i_2 \in [n_r] \\ i_1 < i_2}} F_{i_1,i_2} \left(d_1^{i_1,i_2} + d_2^{i_1,i_2} \right) \\ \text{subject to} \quad & \textit{Equations } (6.12g) \textit{to } (6.12n) \textit{and } (6.13) \end{aligned} \tag{6.14}$$

for an instance that contains disjunctions that share variables with the objective.

6.4.2 Second-order Cone-Based Representation for the Hull Reformulation of the Constrained Layout Problem

B&B Algorithm 9 utilises hull relaxations. A hull relaxation of Problem (6.12) relaxes mathematical constraints in the disjunctions with the perspective transformation, i.e. the mathematical constraints in Equations (6.11) and (6.12h). When the pre-transformed constraints are nonlinear, a direct application of the perspective transformation results in functions that can result in divisions-by-zero when evaluated. The Equation (6.12h) disjunctions contain nonlinear convex constraints. In particular, these constraints are squared Euclidean norms. Hull relaxations of these constraints are SOCP-representable (Ben-Tal and Nemirovski, 2001).

Below we derive the SOCP-representation for the first nonlinear constraint within the Equation (6.12h) disjunction. This SOCP-representability is well known (Ben-Tal and Nemirovski, 2001; Günlük and Linderoth, 2012), we show it here for completeness. Appendix A provides complete reformulations for Problem (6.12). The consider the constraint taken from the Equation (6.12h) disjunction:

$$\left(x_{1,i} - \frac{W_i}{2} - c_{1,j}\right)^2 + \left(x_{2,i} + \frac{H_i}{2} - c_{2,j}\right)^2 \leq r_j^2$$

for some $i \in [n_r]$, $j \in [n_c]$. The hull relaxation of Equation (6.12h) introduces variables $\nu_{1,i,j}, \nu_{2,i,j}$ that as disaggregated variables for $x_{1,i}$ and $x_{2,j}$, repectively, and $y^c_{i,j} \in [0,1]$ in place of $Y^c_{i,j}$. The hull relaxation applies the perspective transformation to Section 6.4.2 deriving (Grossmann and Lee, 2003):

$$y^c_{i,j}\left[\left(\frac{\nu_{1,i,j}}{y^c_{i,j}} - \frac{W_i}{2} - c_{1,j}\right)^2 + \left(\frac{\nu_{2,i,j}}{y^c_{i,j}} + \frac{H_i}{2} - c_{2,j}\right)^2\right] \leq r_j^2 y^c_{i,j} \tag{6.15}$$

For the case $y^c_{i,j} > 0$, we have that:

$$\begin{aligned}
\textit{Equation (6.15)} &\iff \frac{1}{y^c_{i,j}}\left[\left(\nu_{1,i,j} - \frac{W_i y^c_{i,j}}{2} - c_{1,j}y^c_{i,j}\right)^2 + \left(\nu_{2,i,j} + \frac{H_i y^c_{i,j}}{2} - c_{2,j}y^c_{i,j}\right)^2\right] \leq r_j^2 y^c_{i,j} \\
&\iff \left(\nu_{1,i,j} - \frac{W_i y^c_{i,j}}{2} - c_{1,j}y^c_{i,j}\right)^2 + \left(\nu_{2,i,j} + \frac{H_i y^c_{i,j}}{2} - c_{2,j}y^c_{i,j}\right)^2 \leq r_j^2 (y^c_{i,j})^2 \\
&\iff \left\| \left(\nu_{1,i,j} - \frac{W_i y^c_{i,j}}{2} - c_{1,j}y^c_{i,j}, \nu_{2,i,j} + \frac{H_i y^c_{i,j}}{2} - c_{2,j}y^c_{i,j}\right)^\top \right\|_2 \leq r_j y^c_{i,j} \qquad (6.16)
\end{aligned}$$

i.e. we have an SOCP constraint. Evaluating Equation (6.16) at $y_{i,j} = 0$ gives $0 \leq 0$, since $\nu_{1,i,j} = \nu_{2,i,j} = 0$ by Equation (2.22c). Hence, the Equations (2.22c) and (6.16) constraint set is equivalent to the Equations (2.22c) and (6.15) constraint set (under the Equation (2.23) definition of the perspective transformation). Furthermore, Equation (6.16) does not encounter a division-by-zero at $y_{i,j} = 0$.

6.5 Numerical Results

This section compares the effect of applying the Section 6.2 disjunction selection and greedy algorithm, and the Section 6.3 infeasibility propagation strategies to the constrained layout (CLay) Problems (6.12) and (6.14).

6.5.1 System and Solver Specifications

Experiments are run on an Ubuntu 18.04 HP EliteDesk 800 G3 TWR with 16GB RAM and an Intel Core i7-7700K@4.20GHz CPU. Implementations are in Python 3.6.10 using Pyomo 5.6.8 (Hart et al., 2011, 2017) for GDP modelling. We use Gurobi 8.1.1 for: (i) solving CLay hull relaxation in SOCP-form and (ii) deriving irreducible infeasible subsystems in infeasible relaxation instances. Alternative solvers capable of solving the CLay SOCP hull relaxation include CPLEX 12.9 and MOSEK 9.2.

For disjunction selection, our B&B Algorithm 9 implementation breaks ties by selecting the

first disjunction, e.g. under a Definition 6.4 least fractional selection strategy, if disjunction 2 and disjunction 3 both satisfy Definition 6.4 then disjunction 2 is chosen. For Algorithm 10, our B&B implementation assesses separation using Equation (6.8), i.e. we assess separation by only considering the selected disjunction. Finally, for infeasibility propagation, i.e. line 25 of Algorithm 9, our implementation employs laziness. For example, if an Equation (6.9) conflict clause applies to an unexplored node it is added to a local stack of conflict clauses. The conflict clauses in the stack are applied when the node is explored.

6.5.2 Constrained Layout Problem Results

We test our Algorithm 9 implementation with the CLay instances found on MINLPLib2 (Bussieck et al., 2003). The instances are named 'CLay0m0n' where m is the number of circles and n is the number of rectangles. Figures 6.3 to 6.6 parameterise the B&B algorithm with pairs of the Table 6.1 disjunction selection and Table 6.2 branch construction strategies. Since the hull relaxation contains SOCP constraints, Gurobi 8.1.1 uses an interior point algorithm. We derive bounds on all variables involved in disjunctions and filter any disjuncts that are infeasible alone to ensure that the our hull relaxations satisfy constraint qualification conditions (Slater, 1959; Hijazi and Liberti, 2016). This filteration removes one rectangle-circle assignment from each of the clay03$*$ instances.

Using the Problem (6.12) Formulation

Figures 6.3 and 6.4 plot the number of Algorithm 9 B&B nodes explored when solving the CLay instances formulated using the Problem (6.12) formulation. All of these figures show a noticable variance with respect to the disjunction selection strategy, i.e. which disjunction to branch on. For most of the instances, a least fractional disjunction selection strategy explores the least amount of nodes. However, for the larger instances a most non-zero or centre-shifted most fractional strategy, e.g. CLay0205 and CLay0305, becomes competitive. The most fractional disjunction selection strategy does not appear to perform well overall. In particular, the most

fractional disjunction selection strategy does not appear to outperform a centre-shifted most fractional approach. This result is expected since both of these strategies aim to quantify when a disjunction is furthest from integrality, however only the centre-shifted most fractional selection strategy considers the role of the Equation (2.17c) constraint in the relaxation.

We assess branch construction with two variants of greedy Algorithm 9 and two variants of wide branching. Overall the wide branching strategies tend explore less nodes. For the CLay02∗instances there does not appear to a clear difference between these approaches. For the CLay03∗instances complete wide branching shows the best performance. Among the greedy algorithms, we do not see a significant difference between the two proposed approaches.

We see that infeasibility propagation has a significant effect on the number of nodes explored. CLay0304 shows that infeasibility propagation can reduce the number of explored nodes by an order of magnitude. The effect of infeasibility propagation is more modest for the remaining instances, but still generally has a noticable effect in reducing the number of explored nodes.

Using the Problem (6.14) Formulation

We consider the Problem (6.14) to analyse whether having the disjunctions share variables with the objective function affects the performance of Algorithm 9. Figures 6.5 and 6.6 plot the results of solving the CLay instances with Algorithm 9. With respect to the disjunction selection strategies, we see that centre-shifted most fractional performs well with most non-zero also having comparable performance for the larger instances. Most fractional selection performs poorly in general.

Between the branch construction strategies, we establish similar conclusions to that of Figures 6.3 and 6.4, i.e. the wide branch and semi-wide construction approaches tend to perform well. In particular, wide branching performs the best. For the greedy algorithms, we see that the semi-balanced approach does appear to perform well after infeasibility propagation, e.g. for the CLay03∗ instances.

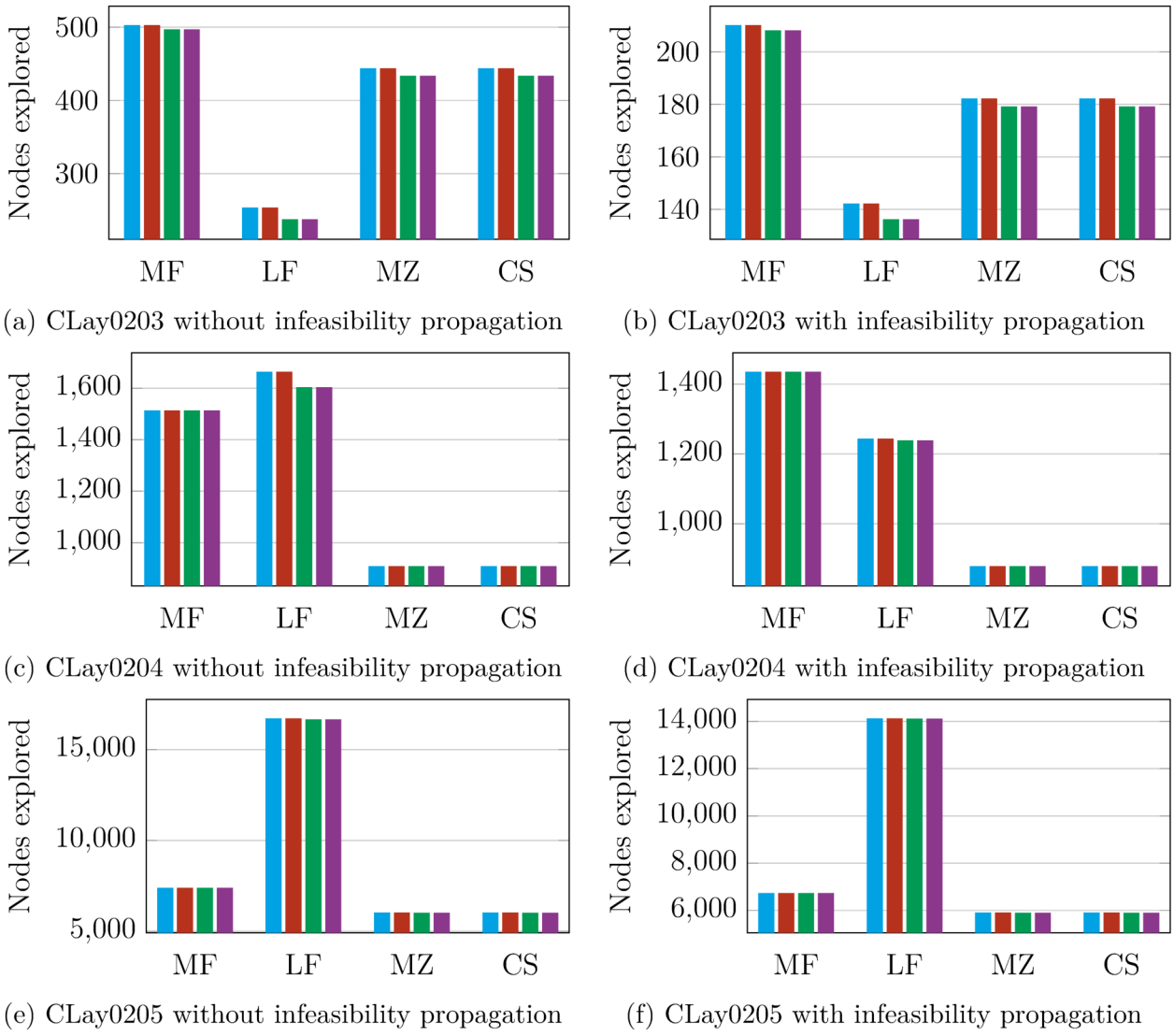
The results of infeasibility propagation in the Problem (6.14) instance generally sees a relatively

significant drop in the number of nodes explored.

Observations

Our numerical results, i.e. Figures 6.3 to 6.6, test three aspects of the B&B algorithm: (i) disjunction selection strategy, (ii) branch construction strategy, and (iii) infeasibility propagation. For the tested instances, we see that nodes explored is most influenced by the disjunction selection strategy and infeasibility propagation. For disjunction selection strategies, centre-shifted most fractional disjunction selection is the most effective strategy overall, however least fractional selection performs well for smaller Problem (6.12) instances. Since, centre-shifted most fractional disjunction selection outperforms the remaining disjunction strategies for the Problem (6.14) instances, it may be an effective in a B&B algorithm that applies basic steps at a node level. Infeasibility propagation is effective as it allows additional subproblems be tightened which can improve bounds and it can enable pruning without having to explore other nodes that contain similar infeasibilities.

Generally, the wide branch construction outperforms the other branch construction strategies in terms of nodes explored. Wide branch construction is effective as relaxation bounds may increase significantly since a disjunct is selected in each child hence pruning may happen sooner. In the greedy branch construction strategies, since a child nodes does not necessarily reduce a disjuction to single disjunct, the corresponding relaxation bounds may not increase quickly and therefore more branching may be required, i.e. additional explored nodes. Such drastic shifts relaxation bounds can happen for CLay, since the objective sums Manhattan distances between rectangle centres, i.e. if circles are far from one another then assigning rectangles to different circles can cause the relaxation bound to increase significantly, whereas if a rectangle still has a disjunctive choice between two circles it can still be placed at (integer infeasible) location between the circles in the relaxation.



(a) CLay0203 without infeasibility propagation

(b) CLay0203 with infeasibility propagation

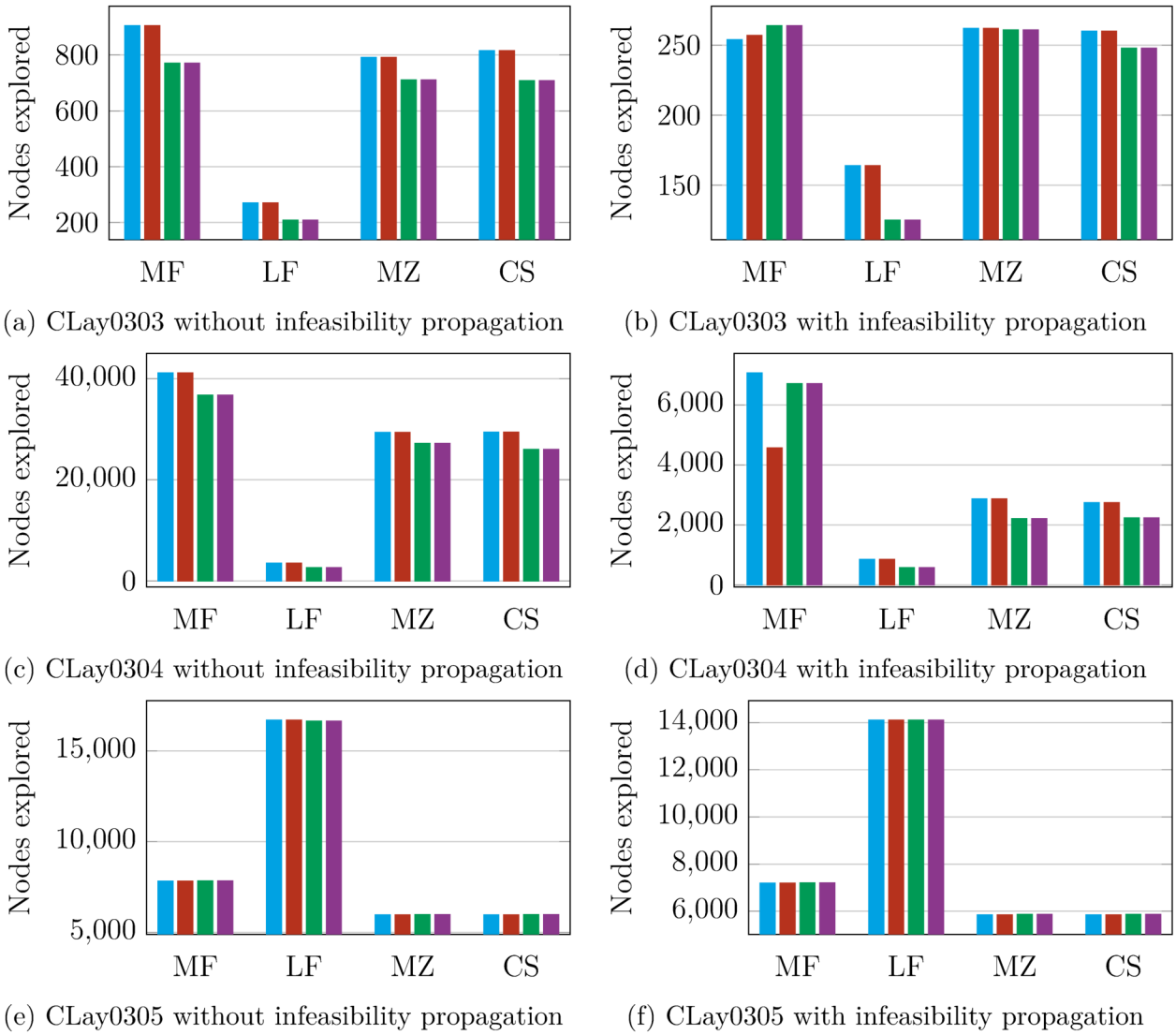
(c) CLay0204 without infeasibility propagation

(d) CLay0204 with infeasibility propagation

(e) CLay0205 without infeasibility propagation

(f) CLay0205 with infeasibility propagation

Figure 6.3: Number of nodes explored by Algorithm 9 when solving the CLay instances formulated with Problem (6.12). CLay0m0n means that the instance has m circles and n rectangles. The left and right figures in a common row solve the same instance but with and without infeasibility propagation. Each grouped set of bars in a given figure correspond to running Algorithm 9 with a Table 6.1 branching disjunction selection strategy. Bars that share the same colour a given figure correspond to running Algorithm 9 with a Table 6.2 branch construction strategy.



(a) CLay0303 without infeasibility propagation

(b) CLay0303 with infeasibility propagation

(c) CLay0304 without infeasibility propagation

(d) CLay0304 with infeasibility propagation

(e) CLay0305 without infeasibility propagation

(f) CLay0305 with infeasibility propagation

Figure 6.4: Number of nodes explored by Algorithm 9 when solving the CLay instances formulated with Problem (6.12). CLay0m0n means that the instance has m circles and n rectangles. The left and right figures in a common row solve the same instance but with and without infeasibility propagation. Each grouped set of bars in a given figure correspond to running Algorithm 9 with a Table 6.1 branching disjunction selection strategy. Bars that share the same colour a given figure correspond to running Algorithm 9 with a Table 6.2 branch construction strategy.

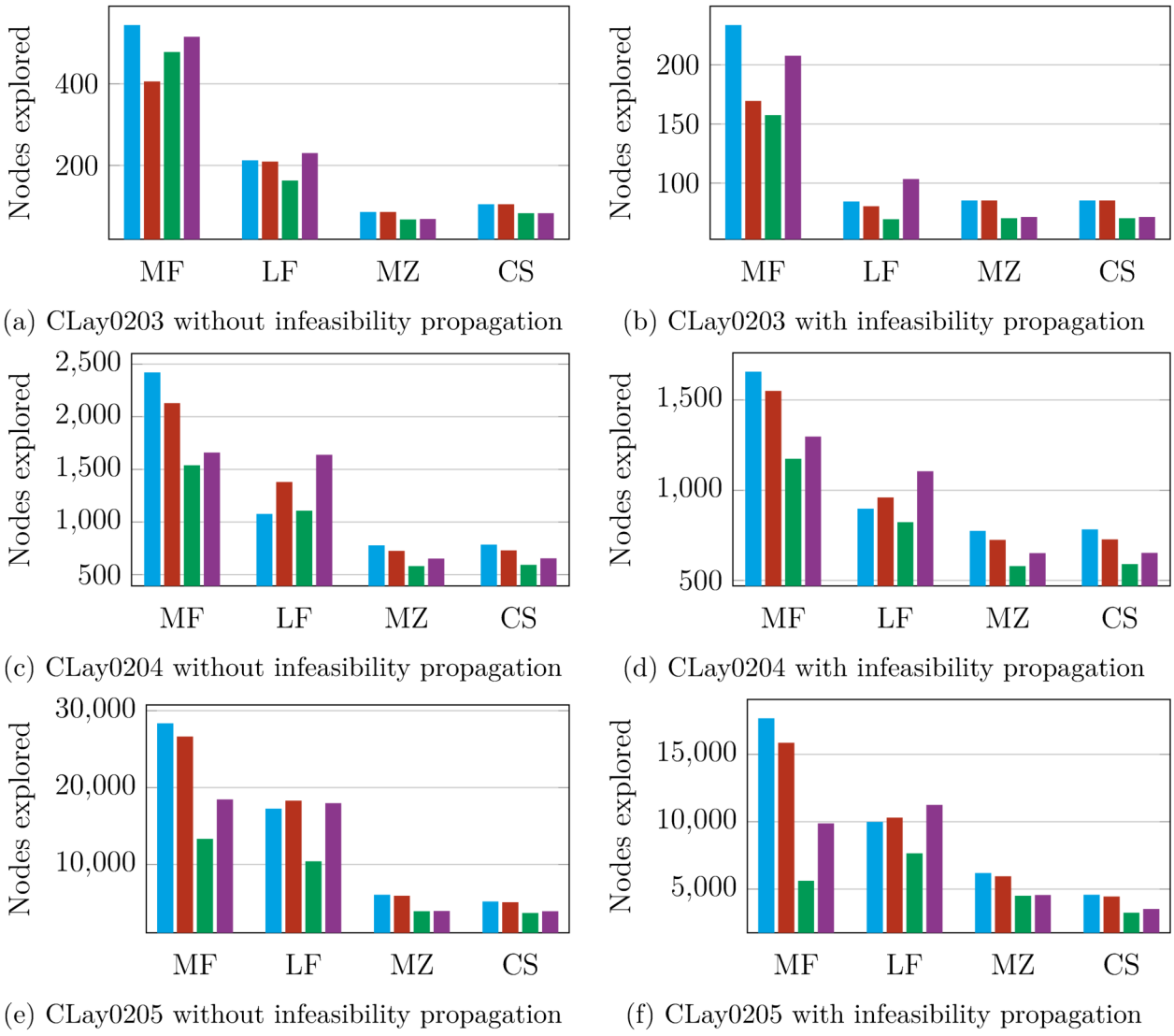
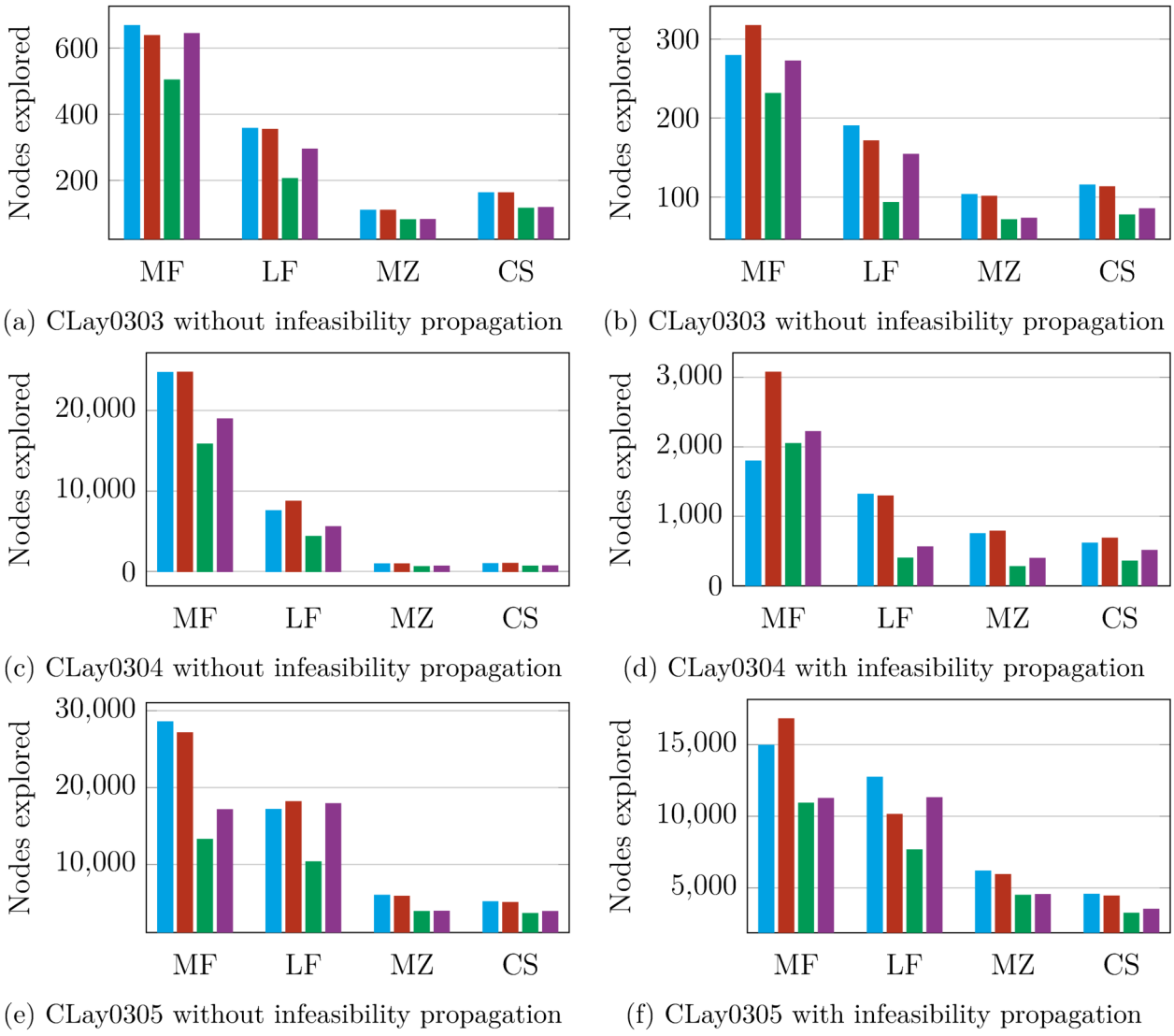


Figure 6.5: Number of nodes explored by Algorithm 9 when solving the CLay instances formulated with Problem (6.14). CLay0m0n means that the instance has m circles and n rectangles. The left and right figures in a common row solve the same instance but with and without infeasibility propagation. Each grouped set of bars in a given figure correspond to running Algorithm 9 with a Table 6.1 branching disjunction selection strategy. Bars that share the same colour a given figure correspond to running Algorithm 9 with a Table 6.2 branch construction strategy.



(a) CLay0303 without infeasibility propagation

(b) CLay0303 without infeasibility propagation

(c) CLay0304 without infeasibility propagation

(d) CLay0304 with infeasibility propagation

(e) CLay0305 without infeasibility propagation

(f) CLay0305 with infeasibility propagation

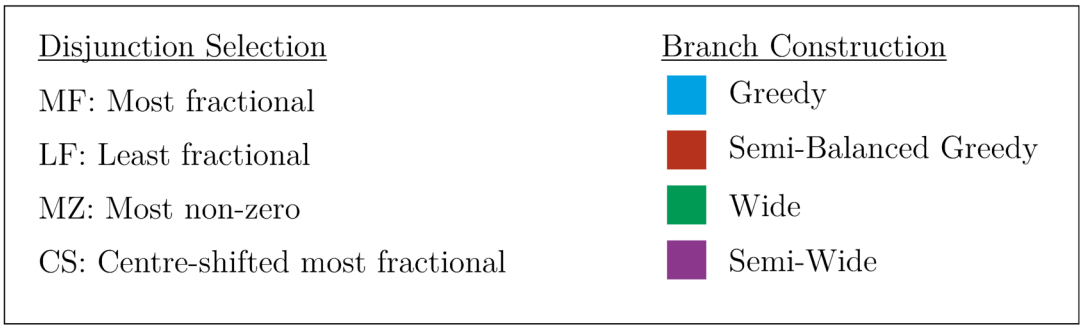


Figure 6.6: Number of nodes explored by Algorithm 9 when solving the CLay instances formulated with Problem (6.14). CLay0m0n means that the instance has m circles and n rectangles. The left and right figures in a common row solve the same instance but with and without infeasibility propagation. Each grouped set of bars in a given figure correspond to running Algorithm 9 with a Table 6.1 branching disjunction selection strategy. Bars that share the same colour a given figure correspond to running Algorithm 9 with a Table 6.2 branch construction strategy.

Chapter 7

Conclusion

7.1 Contributions

7.1.1 Mixed-Integer Convex Nonlinear Optimisation with Gradient-Boosted Trees Embedded

This chapter addresses the large-scale, industrially-relevant Chapter 3 gradient-boosted tree model by directly exploiting: (i) advanced mixed-integer programming technology with strong optimisation formulations, (ii) GBT tree structure with priority towards searching on commonly-occurring variable splits, and (iii) convex penalty terms with enabling fewer mixed-integer optimisation updates. The general form of the optimisation problem appears whenever we wish to optimise a pre-trained gradient-boosted tree with convex terms in the objective, e.g. penalties. It would have been alternatively possible to train and then optimise a smooth and continuous machine learning model, but applications with legacy code may start with a GBT. Our numerical results test against concrete mixture design and chemical catalysis, two applications where the global solution to an optimisation problem is often particularly useful. Our methods not only generate good feasible solutions to the optimisation problem, but they also converge towards proving the exact solution.

We note that functions obtained from limited, known evaluations with machine learning are approximate by default and may deviate from the ground truth, thus, resulting in false optima. The final solution error depends on the training data distribution, noise, and machine learning model complexity. The performance of our PCA-based approach is deteriorated when dealing with clustered data. In this case, regions within the PCA subspace might be far from the training observations. A remedy to this weakness is data analysis, e.g. clustering (Hastie et al., 2009). However, such an approach may introduce additional computational overhead. An alternative direction is using proximity measures (Liaw and Wiener, 2002; Mišić, 2017).

Finally, we acknowledge other approaches for decision-making with optimisation problems whose input is specified by machine learning models. Donti et al. (2017) consider end-to-end task-based learning where probabilistic models are trained to be subsequently used within stochastic programming tasks. Elmachtoub and Grigas (2017) develop a framework for training predictive models with a specific loss function so that the resulting optimisation problem has desirable convexity properties and is statistically consistent. Wilder et al. (2019) propose a two-stage approach for integrating machine learning predictions with combinatorial optimisation problem decisions. The main difference with our work is that we are more focused on the optimisation side.

7.1.2 Using Satisfiability Modulo Theories Derived Unsatisfiable Cores in Mathematical Optimisation

This chapter suggests uses for satisfiability modulo theories (SMT) derived unsatisfiable cores in an optimisation context. The application under study in this chapter is the two-dimensional bin packing problem (2BP). We design three algorithms for solving 2BP, two of which are iterative methods that, respectively, tighten or loosen a single instance, and the third algorithm is a B&B strategy. All of these algorithms partially break an inherent symmetric structure present in 2BP. The B&B algorithm utilises the SMT unsatisfiable cores to: (i) form branching decisions and (ii) generate no-good cuts. Branching on an unsatisfiable core introduces symmetries in the search, we propose additional cuts that separate these symmetries. The no-good cuts derived

directly from the unsatisfiable core break symmetries that occur in infeasibilities. Our numerical tests show that our black-box use of SMT can outperform state-of-the-art mixed-integer linear programming solvers. The chapter discusses using an SMT unsatisfiable cores for cut generation and as a model checker.

7.1.3 Branching and Infeasibility Propagation in Convex Generalized Disjunctive Programming

This chapter designs a B&B algorithm for the convex generalized disjunctive programming (GDP) framework. The contributions of the B&B algorithm include (i) selection strategies for branching on a disjunction, (ii) a greedy algorithm for branch construction, and (iii) infeasibility propagation results for generating no-good cuts. For branching disjunction selection, we propose the *most fractional*, *most non-zero* and *centre-shifted most fractional* strategy. Intuitively, all of these strategies attempt to quantify how undecided, with respect to selecting a disjunct, a relaxation solution is. We study branching by repurposing the use case of the convex GDP cutting plane generating separation problem (Stubbs and Mehrotra, 1999; Vecchietti et al., 2003) to deciding separation among branches, i.e. a branch set that removes the current infeasible-for-GDP relaxation solution. We define minimal separation as a favourable property of in a set of branches and pairwise irreducibility as a heuristic for minimal separation. We show that if a relaxation solution is qualitatively fractional then we can guarantee separation. We design greedy Algorithm 10 that can construct pair irreducible partitions given a qualitatively fractional solution. Furthermore, we suggest a naive branch construction strategy that may generate less local nodes than the Beaumont (1990) wide branching strategy. Our analysis of infeasibility in convex GDP studies leveraging hull relaxation irreducible infeasible subsystems. We show that for a well-formed convex GDP without any propositional constraints, a sufficient requirement for presence of a disjunction in a hull relaxation irreducible infeasible set is given by presence of its disaggregation sum constraint. We numerically evaluate our strategies on the constrained layout problem. For the tested instances, our results show that, with respect to nodes explored, the centre-shifted most fractional disjunction selection strategy performs well

for the larger tested instances, that infeasibility propagation in convex GDP can drastically reduced the number of nodes explored. For the branch construction strategies, wide branching performs the best.

7.2 Future Work

7.2.1 An Online Satisfiability Modulo Theories Approach for a Generalisation of Bin Packing

Firstly, a direct improvement that retains the black-box use case of SMT in the Chapter 5 B&B algorithm is embedding heuristics into the B&B tree. Simple heuristics may be necessary for larger applications, since deriving larger unsatisfiable cores is more time consuming.

This thesis focuses on utilising SMT as a black-box tool that may guide an algorithm, we now discuss development of an online SMT approach. The black-box approach leads to a duplication of effort in the proposed algorithm, since the SMT solver deriving an unsatisfiable core followed by the algorithm recovering the no-good cuts can be carried out together. Furthermore, an SMT solver would benefit from symmetry breaking internally for the decision version of 2BP when deriving unsatisfiable cores. An interesting feature of the Chapter 5 B&B algorithm is that all aspects of the algorithm except for the Equation (5.13) cuts apply to a generalisation of bin packing. This generalisation is the problem:

> Let $\mathcal{I} = [n_C]$ index a set of items and $\mathcal{C} : 2^{\mathcal{I}} \to \mathbb{B}$. Find a partition P of $\mathcal{I}$ such that $\forall S \in P : \mathcal{C}(S)$ and P has minimal cardinality.

For 2BP, $\mathcal{C} = \text{OPP}$, the orthogonal packing problem (Baker et al., 1980). This generalisation motivates two developments: (1) a generic SMT-based B&B algorithm that carries out the same symmetry breaking search strategy and no-good cuts but assumes nothing about $\mathcal{C}$, and (2) theory solvers that assess the condition $\mathcal{C}$ in question. For example, the linear rational arithmetic theory employed in Chapter 5 is more general than 2BP requires.

7.2.2 Initialisation Selection Strategy for the Chapter 4 Branch-and-Bound Parameters

The Chapter 4 decomposition-based GBT lower bounding methodology is parameterised by an initial partition. This thesis selects the initial partition by grouping, in training order, fixed sized subsets where the size is selected by conducting numerical tests offline. An alternative choice of this partition may provide better initial bounds, this motivates the research question: how may we select the initial partition for the decomposition-based GBT lower bounding approach? The trade-off that has to be accounted for is time-to-bound time vs tightness-of-bound. Time-to-bound may be heuristically guided by the subset size. Tightness-of-bound may be related to how interactive trees in a given subset are with each other, i.e. do they split on the same variables? A scoring approach similar to the pseudocost initialisation may provide a good quantification of interactivity.

7.2.3 Basic Step Selection in Convex Generalized Disjunctive Programming

This thesis proposes strategies for B&B branching in convex GDP. However, effective bounding is also an important aspect of B&B. For convex GDP algorithms, basic steps support effective bounding. Ruiz and Grossmann (2010) propose rules for selection of basic steps that tighten a GDP formulation. Papageorgiou and Trespalacios (2018) also investigate basic step selection by leveraging a Lagrangian decomposition to quantify the expected bound improvement of a basic step and develop *pseudo basic steps* that intersect disjuctions within the Lagrangian decomposition. In addition to the Ruiz and Grossmann (2010) rules, we consider two goals for basic step selection: (1) infeasibility induction and (2) bound tightening. Infeasibility induction aims to combine disjunctions and global constraints that cause several immediate infeasible disjuncts in the resulting disjunction, i.e. problem growth does not hit the worst case. Bound tightening attempts to select basic steps that a cause a direct improvement in the lower bound. The Papageorgiou and Trespalacios (2018) quantification of expected bound improvement of

a basic step is in line with the second goal. An important consideration for both of these goals is interaction, i.e. how many variables do the selected disjunctions have in common and do their constraints oppose each other? For bound tightening, we have to quantify the effect of a basic step on the objective variables, i.e. do interacting constraints affect the objective variables significantly? These considerations motivate modelling a convex GDP as a graph where each global constraint, disjunction and the objective is a unique vertex. Two vertices share an edge if and only if their corresponding modelling structures share variables. Let the distance between the objective vertex and a non-objective vertex be the length of a shortest path from the objective vertex to the non-objective vertex. An interesting research question is: can we model a convex GDP as graph to guide basic step selection?

7.2.4 Using Alternative Relaxation Strategies in the Convex Generalized Disjunctive Programming Branch-and-Bound Algorithm

Hull reformulations are generally much larger than big-M reformulations. Also, the presence of the perspective function may cause numerical difficulties in practice. Using big-M reformulations may be more resistant to these issues, however the cost is relaxation tightness (Vecchietti et al., 2003). Furthermore, since relaxation solutions can be outside of the hull reformulation feasible region, a branch-and-cut framework may be more appropriate (Ceria et al., 1998; Stubbs and Mehrotra, 1999). For a branch-and-cut approach, Algorithm 10 can also provide valid cutting planes when the branch separation problem is solved to optimality.

Bibliography

Achterberg, T. (2007a). Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20.

Achterberg, T. (2007b). *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin.

Achterberg, T. (2009). SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41.

Achterberg, T., Koch, T., and Martin, A. (2005). Branching rules revisited. *Operations Research Letters*, 33(1):42–54.

Anstreicher, K., Brixius, N., Goux, J.-P., and Linderoth, J. (2002). Solving large quadratic assignment problems on computational grids. *Mathematical Programming*, 91(3):563–588.

Anstreicher, K. M. (2003). Recent advances in the solution of quadratic assignment problems. *Mathematical Programming*, 97(1):27–42.

Apt, K. R. (1999). The essence of constraint propagation. *Theoretical Computer Science*, 221(1):179–210.

Backofen, R. and Will, S. (1999). Excluding symmetries in constraint-based search. In Jaffar, J., editor, *Principles and Practice of Constraint Programming – CP'99*, pages 73–87, Berlin, Heidelberg. Springer Berlin Heidelberg.

Baker, B. S., Coffman, Jr., E. G., and Rivest, R. L. (1980). Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855.

Balas, E. (1974). Intersection cuts from disjunctive constraints. Technical Report Management Sciences Research Report No. 330, Graduate School of Industrial Administration, Carnegie Mellon University.

Balas, E. (1975). Disjunctive programming: Cutting planes from logical conditions. In *Nonlinear Programming 2*, pages 279–312. Elsevier.

Balas, E. (1977). A note on duality in disjunctive programming. *Journal of Optimization Theory and Applications*, 21(4):523–528.

Balas, E. (1979). Disjunctive programming. *Annals of Discrete Mathematics*, 5:3–51.

Balas, E. (1985). Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. *SIAM Journal on Algebraic Discrete Methods*, 6(3):466–486.

Balas, E. (1998). Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1):3–44.

Balas, E. (2018). *Disjunctive Programming*. Springer International Publishing.

Balas, E., Ceria, S., and Cornuéjols, G. (1993). A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical Programming*, 58(1):295–324.

Balas, E., Ceria, S., and Cornuéjols, G. (1996). Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42(9):1229–1246.

Balas, E. and Jeroslow, R. (1972). Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, 23(1):61–69.

Beale, E. M. L. and Forrest, J. J. H. (1976). Global optimization using special ordered sets. *Mathematical Programming*, 10(1):52–69.

Beale, E. M. L. and Tomlin, J. A. (1970). Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables. In *Proceedings of the fifth international conference on operational research*, pages 447–454. Tavistock, London.

Beaumont, N. (1990). An algorithm for disjunctive programs. *European Journal of Operational Research*, 48(3):362–371.

Bellmore, M. and Nemhauser, G. L. (1968). The traveling salesman problem: A survey. *Operations Research*, 16(3):538–558.

Belotti, P. and Berthold, T. (2017). Three ideas for a feasibility pump for nonconvex minlp. *Optimization Letters*, 11(1):3–15.

Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., and Mahajan, A. (2013). Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131.

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

Bemporad, A. and Giorgetti, N. (2004). *SAT-Based Branch & Bound and Optimal Control of Hybrid Dynamical Systems*, pages 96–111. Springer Berlin Heidelberg.

Bemporad, A. and Giorgetti, N. (2006). Logic-based solution methods for optimal control of hybrid systems. *IEEE Transactions on Automatic Control*, 51(6):963–976.

Ben-Tal, A. and Nemirovski, A. (2001). *Lectures on Modern Convex Optimization*. SIAM.

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252.

Berkey, O. J. and Wang, Y. P. (1987). Two-dimensional finite bin-packing algorithms. *Journal of the Operational Research Society*, 38(5):423–429.

Berthold, T. (2014). *Heuristic algorithms in global MINLP solvers*. PhD thesis, Technische Universität Berlin.

Berthold, T. (2017). Improving the performance of mip and minlp solvers by integrated heuristics. In Dörner, K. F., Ljubic, I., Pflug, G., and Tragler, G., editors, *Operations Research Proceedings 2015*, pages 19–24. Springer International Publishing.

Bertsekas, D. P. (2014). *Constrained optimization and Lagrange multiplier methods*. Academic press.

Bienstock, D. (1996). Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming*, 74(2):121–140.

Biere, A., Cimatti, A., Clarke, E., and Zhu, Y. (1999). *Tools and Algorithms for the Construction and Analysis of Systems: 5th International Conference, TACAS'99*, chapter Symbolic Model Checking without BDDs, pages 193–207. Springer Berlin Heidelberg.

Biere, A., Heule, M., van Maaren, H., and Walsh, T. (2009). *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands.

Bjørner, N. and De Moura, L. (2011). Satisfiability modulo theories: Introduction and applications. *Communications of the ACM*, pages 69–77.

Bjørner, N. and Phan, A. D. (2014). νZ - maximal satisfaction with Z3. In *Proceedings of the 6th International Symposium on Symbolic Computation in Software Science*, volume 30 of *SCSS*, pages 1–9.

Bjørner, N., Phan, A.-D., and Fleckenstein, L. (2015). *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015*, chapter νZ - An Optimizing SMT Solver, pages 194–199. Springer Berlin Heidelberg.

Bollapragada, S., Ghattas, O., and Hooker, J. N. (2001). Optimal design of truss structures by logic-based branch and cut. *Operations Research*, 49(1):42–51.

Bollas, G. M., Barton, P. I., and Mitsos, A. (2009). Bilevel optimization formulation for parameter estimation in vapor-liquid(-liquid) phase equilibrium problems. *Chemical Engineering Science*, 64(8):1768–1783.

Bonami, P. (2011). Lift-and-project cuts for mixed integer convex programs. In Günlük, O. and Woeginger, G. J., editors, *Integer Programming and Combinatoral Optimization*, pages 52–64. Springer Berlin Heidelberg.

Bonami, P. and Lejeune, M. A. (2009). An exact solution approach for portfolio optimization problems under stochastic and integer constraints. *Operations Research*, 57(3):650–670.

Bonfietti, A., Lombardi, M., and Milano, M. (2015). Embedding decision trees and random forests in constraint programming. In *Integration of AI and OR Techniques in Constraint Programming*, pages 74–90.

Boukouvala, F., Misener, R., and Floudas, C. A. (2016). Global optimization advances in mixed-integer nonlinear programming, MINLP, and constrained derivative-free optimization, CDFO. *European Journal of Operational Research*, 252(3):701–727.

Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth, Inc.

Brucker, P., Drexl, A., MÃűhring, R., Neumann, K., and Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41.

Bussieck, M. R., Drud, A. S., and Meeraus, A. (2003). MINLPLib—A Collection of Test Models for Mixed-Integer Nonlinear Programming. *INFORMS Journal on Computing*, 15(1):114–119.

Caballero, J. A. and Grossmann, I. E. (2001). Generalized disjunctive programming model for the optimal synthesis of thermally linked distillation columns. *Industrial & Engineering Chemistry Research*, 40(10):2260–2274.

Callia D'Iddio, A. and Huth, M. (2017). Manyopt: An extensible tool for mixed, non-linear optimization through SMT solving. *CoRR*, abs/1702.01332.

Carbonneau, R. A., Caporossi, G., and Hansen, P. (2011). Globally optimal clusterwise regression by mixed logical-quadratic programming. *European Journal of Operational Research*, 212(1):213–222.

Carbonneau, R. A., Caporossi, G., and Hansen, P. (2012). Extensions to the repetitive branch and bound algorithm for globally optimal clusterwise regression. *Computers & Operations Research*, 39(11):2748–2762.

Carvajal, R., Ahmed, S., Nemhauser, G., Furman, K., Goel, V., and Shao, Y. (2014). Using diversification, communication and parallelism to solve mixed-integer linear programs. *Operations Research Letters*, 42(2):186–189.

Castro, P. M. (2015). Tightening piecewise McCormick relaxations for bilinear problems. *Computers & Chemical Engineering*, 72:300–311.

Castro, P. M. and Grossmann, I. E. (2012). Generalized disjunctive programming as a systematic modeling framework to derive scheduling formulations. *Industrial & Engineering Chemistry Research*, 51(16):5781–5792.

Castro, P. M. and Grossmann, I. E. (2014). Optimality-based bound contraction with multiparametric disaggregation for the global optimization of mixed-integer bilinear problems. *Journal of Global Optimization*, 59(2):277–306.

Castro, P. M. and Marques, I. (2015). Operating room scheduling with generalized disjunctive programming. *Computers & Operations Research*, 64:262–273.

Ceria, S., Cordier, C., Marchand, H., and Wolsey, L. A. (1998). Cutting planes for integer programs with general integer variables. *Mathematical Programming*, 81(2):201–214.

Chakravarti, N. (1994). Some results concerning post-infeasibility analysis. *European Journal of Operational Research*, 73(1):139–143.

Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794.

Chinneck, J. W. (2019). The maximum feasible subset problem (maxFS) and applications. *INFOR: Information Systems and Operational Research*, 0(0):1–21.

Chou, J.-S., Chiu, C.-K., Farfoura, M., and Al-Taharwa, I. (2011). Optimizing the prediction accuracy of concrete compressive strength based on a comparison of data-mining techniques. *Journal of Computing in Civil Engineering*, 25(3):242–253.

Chung, F. R. K., Garey, M. R., and Johnson, D. S. (1982). On packing two-dimensional bins. *SIAM Journal on Algebraic Discrete Methods*, 3(1):66–76.

Cimatti, A., Griggio, A., Schaafsma, B. J., and Sebastiani, R. (2013). *Tools and Algorithms for the Construction and Analysis of Systems: 19th International Conference, TACAS 2013*, chapter The MathSAT5 SMT Solver, pages 93–107. Springer Berlin Heidelberg.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*, pages 151–158, New York, New York, USA. ACM Press.

Cornuéjols, G., Liberti, L., and Nannicini, G. (2011). Improved strategies for branching on general disjunctions. *Mathematical Programming*, 130(2):225–247.

D'Ambrosio, C., Frangioni, A., Liberti, L., and Lodi, A. (2012). A storm of feasibility pumps for nonconvex MINLP. *Mathematical Programming*, 136:375–402.

Dantzig, G. B. (1998). *Linear Programming and Extensions*. Princeton University Press.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397.

Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215.

De Moura, L. and Bjørner, N. (2008). *Z3: An Efficient SMT Solver*, pages 337–340. Springer.

de Moura, L. and Jovanović, D. (2013). A model-constructing satisfiability calculus. In Giacobazzi, R., Berdine, J., and Mastroeni, I., editors, *Verification, Model Checking, and Abstract Interpretation*, pages 1–12. Springer Berlin Heidelberg.

de Moura, L. and Passmore, G. O. (2013). Computation in real closed infinitesimal and transcendental extensions of the rationals. In Bonacina, M. P., editor, *Automated Deduction - CADE-24*, volume 7898 of *Lecture Notes in Computer Science*, pages 178–192. Springer Berlin Heidelberg.

DeRousseau, M., Kasprzyk, J., and Srubar, W. (2018). Computational design optimization of concrete mixtures: A review. *Cement and Concrete Research*, 109:42–53.

Dheeru, D. and Karra Taniskidou, E. (2017). UCI machine learning repository.

Dias, G. and Liberti, L. (2019). Exploiting symmetries in mathematical programming via orbital independence. *Annals of Operations Research*.

Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213.

Donti, P., Amos, B., and Kolter, J. Z. (2017). Task-based end-to-end model learning in stochastic optimization. In *Advances in Neural Information Processing Systems 30*, pages 5484–5494. Curran Associates, Inc.

Du, K. and Kearfott, R. B. (1994). The cluster problem in multivariate global optimization. *Journal of Global Optimization*, 5(3):253–265.

Duran, M. A. and Grossmann, I. E. (1986a). A mixed-integer nonlinear programming algorithm for process systems synthesis. *AIChE Journal*, 32(4):592–606.

Duran, M. A. and Grossmann, I. E. (1986b). An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36(3):307–339.

Easton, K., Nemhauser, G., and Trick, M. (2003). Solving the travelling tournament problem: A combined integer programming and constraint programming approach. In *Practice and Theory of Automated Timetabling IV*, pages 100–109.

Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43.

Eén, N. and Sörensson, N. (2006). Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26.

Elmachtoub, A. N. and Grigas, P. (2017). Smart "Predict, then Optimize". *arXiv e-prints*. arXiv:1710.08005.

Erdal, H. I. (2013). Two-level and hybrid ensembles of decision trees for high performance concrete compressive strength prediction. *Engineering Applications of Artificial Intelligence*, 26(7):1689–1697.

Faria, D. C. and Bagajewicz, M. J. (2012). A new approach for global optimization of a class of MINLP problems with applications to water management and pooling problems. *AIChE Journal*, 58(8):2320–2335.

Fattahi, A., Elaoud, S., Azer, E. S., and Turkay, M. (2014). A novel integer programming formulation with logic cuts for the U-shaped assembly line balancing problem. *Internation Journal of Production Research*, 52(5):1318–1333.

Fernández, J. and Tóth, B. (2009). Obtaining the efficient set of nonlinear biobjective optimization problems via interval branch-and-bound methods. *Computational Optimization and Applications*, 42(3):393–419.

Fischetti, M. and Lodi, A. (2011). *Heuristics in Mixed Integer Programming*. Wiley.

Fischetti, M. and Monaci, M. (2012). Branching on nonchimerical fractionalities. *Operations Research Letters*, 40(3):159–164.

Fischetti, M. and Monaci, M. (2013). Backdoor branching. *INFORMS Journal on Computing*, 25(4):693–700.

Fisher, M. L. (1981). The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18.

Fletcher, R. and Leyffer, S. (1994). Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66(1):327–349.

Frangioni, A. and Gentile, C. (2006). Perspective cuts for a class of convex 0-1 mixed integer programs. *Mathematical Programming*, 106(2):225–236.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378.

Furman, K. C. and Sahinidis, N. V. (2002). A critical review and annotated bibliography for heat exchanger network synthesis in the 20th century. *Industrial & Engineering Chemistry Research*, 41(10):2335–2370.

Furman, K. C., Sawaya, N. W., and Grossmann, I. E. (2020). A computationally useful algebraic representation of nonlinear disjunctive convex sets using the perspective function. *Computational Optimization and Applications*, 76(2):589–614.

Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., and Tinelli, C. (2004). *Computer Aided Verification: 16th International Conference, CAV 2004*, chapter DPLL(T): Fast Decision Procedures, pages 175–188. Springer Berlin Heidelberg.

Gao, S., Avigad, J., and Clarke, E. M. (2012). δ-complete decision procedures for satisfiabilityÂăoverÂăthe reals. In Gramlich, B., Miller, D., and Sattler, U., editors, *Automated Reasoning*, pages 286–300. Springer Berlin Heidelberg.

Gao, S., Kong, S., and Clarke, E. M. (2013). dreal: An smt solver for nonlinear theories over the reals. In Bonacina, M. P., editor, *Automated Deduction – CADE-24*, pages 208–214. Springer Berlin Heidelberg.

Gendron, B., ScutellÃă, M. G., Garroppo, R. G., Nencioni, G., and Tavanti, L. (2016). A branch-and-benders-cut method for nonlinear power design in green wireless local area networks. *European Journal of Operational Research*, 255(1):151–162.

Geoffrion, A. M. (1974). *Lagrangean relaxation for integer programming*, pages 82–114. Springer Berlin Heidelberg.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1):3–42.

Gilpin, A. and Sandholm, T. (2011). Information-theoretic approaches to branching in search. *Discrete Optimization*, 8(2):147–159.

Glover, F. and Laguna, M. (1998). *Tabu Search*, pages 2093–2229. Springer US, Boston, MA.

Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278.

Grossmann, I. E. and Lee, S. (2003). Generalized convex disjunctive programming: Nonlinear convex hull relaxation. *Computational Optimization and Applications*, 26(1):83–100.

Grossmann, I. E. and Ruiz, J. P. (2012). Generalized disjunctive programming: A framework for formulation and alternative algorithms for MINLP optimization. In Lee, J. and Leyffer, S., editors, *Mixed Integer Nonlinear Programming*, pages 93–115, New York, NY. Springer New York.

Grossmann, I. E. and Trespalacios, F. (2013). Systematic modeling of discrete-continuous optimization models through generalized disjunctive programming. *AIChE Journal*, 59(9):3276–3295.

Günlük, O. and Linderoth, J. (2012). Perspective reformulation and applications. In Lee, J. and Leyffer, S., editors, *Mixed Integer Nonlinear Programming*, pages 61–89.

Gupta, O. K. and Ravindran, A. (1985). Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31(12):1533–1546.

Hart, W. E., Laird, C., Watson, J.-P., and Woodruff, D. L. (2012). *Pyomo–optimization modeling in Python*, volume 67. Springer Science & Business Media.

Hart, W. E., Laird, C. D., Watson, J.-P., Woodruff, D. L., Hackebeil, G. A., Nicholson, B. L., and Siirola, J. D. (2017). *Pyomo–optimization modeling in Python*, volume 67. Springer Science & Business Media, second edition.

Hart, W. E., Watson, J.-P., and Woodruff, D. L. (2011). Pyomo: modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3(3):219–260.

Hartmann, S. and Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*. Springer-Verlag New York, second edition.

Hijazi, H. and Liberti, L. (2016). Constraint qualification failure in action. *Operations Research Letters*, 44(4):503–506.

Hiriart-Urruty, J.-B. and Lemaréchal, C. (1993). *Convex Analysis and Minimization Algorithms I*. Springer-Verlag.

Hooker, J. N. (2002). Logic, optimization, and constraint programming. *INFORMS Journal on Computing*, 14(4):295–321.

Hooker, J. N. (2007). Planning and scheduling by logic-based Benders decomposition. *Operations Research*, 55(3):588–602.

Hooker, J. N. and Osorio, M. A. (1999). Mixed logical-linear programming. *Discrete Applied Mathematics*, 96-97:395–442.

Hooker, J. N. and Ottoson, G. (2003). Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60.

Jackson, J. R. and Grossmann, I. E. (2001). A disjunctive programming approach for the optimal design of reactive distillation columns. *Computers & Chemical Engineering*, 25(11):1661–1673.

Jain, V. and Grossmann, I. E. (2001). Algorithms for Hybrid MILP/CP Models for a Class of Optimization Problems. *INFORMS Journal on Computing*, 13(4):258–276.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning*. Springer-Verlag New York.

Jeroslow, R. (1977). Cutting-plane theory: Disjunctive methods. In Hammer, P., Johnson, E., Korte, B., and Nemhauser, G., editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 293–330. Elsevier.

Jolliffe, I. T. (2002). *Principal Component Analysis*. Springer-Verlag New York, second edition.

Jonuzaj, S., Akula, P. T., Kleniati, P.-M., and Adjiman, C. S. (2016). The formulation of optimal mixtures with generalized disjunctive programming: A solvent design case study. *AIChE Journal*, 62(5):1616–1633.

Kaibel, V., Peinhardt, M., and Pfetsch, M. E. (2011). Orbitopal fixing. *Discrete Optimization*, 8(4):595–610.

Kannan, R. and Barton, P. I. (2017). The cluster problem in constrained global optimization. *Journal of Global Optimization*, 69(3):629–676.

Karamanov, M. and Cornuéjols, G. (2011). Branching on general disjunctions. *Mathematical Programming*, 128(1):403–436.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 302–311, New York, NY, USA. ACM.

Karuppiah, R. and Grossmann, I. E. (2006). Global optimization for the synthesis of integrated water systems in chemical processes. *Computers & Chemical Engineering*, 30(4):650–673.

Kautz, H. and Selman, B. (1992). Planning as satisfiability. In *European Conference on Artificial Intelligence*, ECAI '92, pages 359–363, New York, NY, USA. John Wiley & Sons, Inc.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30*, pages 3149–3157. Curran Associates, Inc.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948.

Khachiyan, L. G. (1979). A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1096.

Kılınç, M., Linderoth, J., Luedtke, J., and Miller, A. (2014). Strong-branching inequalities for convex mixed integer nonlinear programs. *Computational Optimization and Applications*, 59(3):639–665.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Kirst, P., Rigterink, F., and Stein, O. (2017). Global optimization of disjunctive programs. *Journal of Global Optimization*, 69(2):283–307.

Klabjan, D., Johnson, E. L., Nemhauser, G. L., Gelman, E., and Ramaswamy, S. (2001). Solving large airline crew scheduling problems: Random pairing generation and strong branching. *Computational Optimization and Applications*, 20(1):73–91.

Klee, V. and Minty, G. (1972). How good is the simplex algorithm? In *Inequalities*. Academic Press.

Konno, H. and Wijayanayake, A. (2001). Portfolio optimization problem under concave transaction costs and minimal transaction unit constraints. *Mathematical Programming*, 89(2):233–250.

Kouyialis, G. and Misener, R. (2017). Detecting symmetry in designing heat exchanger networks. In *Proceedings of the International Conference of Foundations of Computer-Aided Process Operations-FOCAPO/CPC*.

Kouyialis, G., Wang, X., and Misener, R. (2019). Symmetry detection for quadratic optimization using binary layered graphs. *Processes*, 7(11):838.

Kronqvist, J., Bernal, D. E., Lundell, A., and Grossmann, I. E. (2019). A review and comparison of solvers for convex MINLP. *Optimization and Engineering*, 20(2):397–455.

Kronqvist, J., Lundell, A., and Westerlund, T. (2016). The extended supporting hyperplane algorithm for convex mixed-integer nonlinear programming. *Journal of Global Optimization*, 64(2):249–272.

Kuhn, M. (2008). Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5):1–26.

Kulkarni, R. and Bhave, P. (1985). Integer programming formulations of vehicle routing problems. *European Journal of Operational Research*, 20(1):58–67.

Lagriffoul, F. and Andres, B. (2016). Combining task and motion planning: A culprit detection problem. *The International Journal of Robotics Research*, 35(8):890–927.

Land, A. H. and Doig, A. G. (1960). An automatic method for solving discrete programming problems. *Econometrica*, 28(1):497–520.

Lee, S. and Grossmann, I. E. (2000). New algorithms for nonlinear generalized disjunctive programming. *Computers & Chemical Engineering*, 24(9-10):2125–2141.

Lee, S. and Grossmann, I. E. (2001). A global optimization algorithm for nonconvex generalized disjunctive programming and applications to process systems. *Computers & Chemical Engineering*, 25(11):1675–1697.

Letsios, D., Baltean-Lugojan, R., Ceccon, F., Mistry, M., Wiebe, J., and Misener, R. (2020). Approximation algorithms for process systems engineering. *Computers & Chemical Engineering*, 132:106599.

Lhomme, O. (2003). An efficient filtering algorithm for disjunction of constraints. In Rossi, F., editor, *Principles and Practice of Constraint Programming – CP 2003*, pages 904–908. Springer Berlin Heidelberg.

Li, H. and Womer, K. (2008). Scheduling projects with multi-skilled personnel by a hybrid MILP/CP Benders decomposition algorithm. *Journal of Scheduling*, 12(3):281–298.

Liaw, A. and Wiener, M. (2002). Classification and regression by randomForest. *R News*, pages 18–22.

Liberti, L. (2012). Symmetry in mathematical programming. In Lee, J. and Leyffer, S., editors, *Mixed Integer Nonlinear Programming*, pages 263–283. Springer New York.

Liberti, Leo (2019). Undecidability and hardness in mixed-integer nonlinear programming. *RAIRO-Oper. Res.*, 53(1):81–109.

Liu, W. B. and Floudas, C. A. (1993). A remark on the GOP algorithm for global optimization. *Journal of Global Optimization*, 3(4):519–521.

Lodi, A., Martello, S., and Vigo, D. (1999). Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11(4):345–357.

Lombardi, M. and Milano, M. (2018). Boosting combinatorial problem modeling with machine learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 5472–5478.

Lombardi, M., Milano, M., and Bartolini, A. (2017). Empirical decision model learning. *Artificial Intelligence*, 244:343–367. Combining Constraint Solving with Mining and Learning.

Lundell, A., Kronqvist, J., and Westerlund, T. (2017). SHOT – a global solver for convex MINLP in Wolfram Mathematica. In EspuÃśa, A., Graells, M., and Puigjaner, L., editors, *27th European Symposium on Computer Aided Process Engineering*, volume 40 of *Computer Aided Chemical Engineering*, pages 2137–2142. Elsevier.

Malik, S. and Zhang, L. (2009). Boolean satisfiability from theoretical hardness to practical success. *Communications of the ACM*, 52(8):76–82.

Manolios, P. and Papavasileiou, V. (2013). *Computer Aided Verification: 25th International Conference, CAV 2013*, chapter ILP Modulo Theories, pages 662–677. Springer Berlin Heidelberg.

Maranas, C. D. and Floudas, C. A. (1992). A global optimization approach for LennardâĂŘJones microclusters. *The Journal of Chemical Physics*, 97(10):7667–7678.

Maranas, C. D. and Floudas, C. A. (1995). Finding all solutions of nonlinearly constrained systems of equations. *Journal of Global Optimization*, 7(2):143–182.

Maravelias, C. T. (2006). A decomposition framework for the scheduling of single- and multi-stage processes. *Computers & Chemical Engineering*, 30(3):407–420.

Maravelias, C. T. and Grossmann, I. E. (2004). A hybrid MILP/CP decomposition approach for the continuous time scheduling of multipurpose batch plants. *Computers & Chemical Engineering*, 28(10):1921–1949.

Maravelias, C. T. and Sung, C. (2009). Integration of production planning and scheduling: Overview, challenges and opportunities. *Computers & Chemical Engineering*, 33(12):1919–1930.

Margot, F. (2002). Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94(1):71–90.

Margot, F. (2003). Small covering designs by branch-and-cut. *Mathematical Programming*, 94(2):207–220.

Margot, F. (2010). *Symmetry in Integer Linear Programming*, pages 647–686. Springer Berlin Heidelberg.

Marques-Silva, J., Lynce, I., and Malik, S. (2009). *Conflict-driven clause learning SAT solvers*, pages 131–153. Number 1 in Frontiers in Artificial Intelligence and Applications. IOS Press, Netherlands, 1 edition.

Marques-Silva, J. P. and Sakallah, K. A. (1996). GRASP — a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '96, pages 220–227, Washington, DC, USA. IEEE Computer Society.

Martello, S., Pisinger, D., Vigo, D., Boef, E. D., and Korst, J. (2007). Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem. *ACM Transactions on Mathematical Software*, 33(1):7–es.

Martello, S. and Vigo, D. (1998). Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3):388–399.

McCormick, G. P. (1976). Computability of global solutions to factorable nonconvex programs: Part I — convex underestimating problems. *Mathematical Programming*, 10(1):147–175.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092.

Miranda, L. J. V. (2018). PySwarms: A research toolkit for Particle Swarm Optimization in Python. *The Journal of Open Source Software*, 3.

Misener, R. and Floudas, C. A. (2010). Piecewise-linear approximations of multidimensional functions. *Journal of Optimization Theory and Applications*, 145(1):120–147.

Misener, R. and Floudas, C. A. (2013). GloMIQO: Global Mixed-Integer Quadratic Optimizer. *Journal of Global Optimization*, 57(1):3–50.

Misener, R. and Floudas, C. A. (2014). ANTIGONE: Algorithms for continuous / integer global optimization of nonlinear equations. *Journal of Global Optimization*, 59(2):503–526.

Misener, R., Gounaris, C. E., and Floudas, C. A. (2009). Global optimization of gas lifting operations: A comparative study of piecewise linear formulations. *Industrial & Engineering Chemistry Research*, 48(13):6098–6104.

Mistry, M., Callia D'Iddio, A., Huth, M., and Misener, R. (2018a). Satisfiability modulo theories for process systems engineering. *Computers & Chemical Engineering*, 113:98–114.

Mistry, M., Letsios, D., Krennrich, G., Lee, R. M., and Misener, R. (2018b). Mixed-integer convex nonlinear optimization with gradient-boosted trees embedded. *arXiv e-prints*. arXiv:1803.00952.

Mistry, M. and Misener, R. (2016). Optimising heat exchanger network synthesis using convexity properties of the logarithmic mean temperature difference. *Computers & Chemical Engineering*, 94:1–17.

Mišić, V. V. (2017). Optimization of Tree Ensembles. *ArXiv e-prints*. arXiv:1705.10883.

Mizutani, F. T., Pessoa, F. L. P., Queiroz, E. M., Hauan, S., and Grossmann, I. E. (2003). Mathematical programming model for heat-exchanger network synthesis including detailed heat-exchanger designs. 1. shell-and-tube heat-exchanger design. *Industrial & Engineering Chemistry Research*, 42(17):4009–4018.

Moreno, A., Munari, P., and Alem, D. (2019). A branch-and-Benders-cut algorithm for the crew scheduling and routing problem in road restoration. *European Journal of Operational Research*, 275(1):16–34.

Morrison, D. R., Jacobson, S. H., Sauppe, J. J., and Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102.

Morrison, D. R., Sauppe, J. J., Sewell, E. C., and Jacobson, S. H. (2014). A wide branching strategy for the graph coloring problem. *INFORMS Journal on Computing*, 26(4):704–717.

Nelson, G. and Oppen, D. C. (1979). Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257.

Nelson, G. and Oppen, D. C. (1980). Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364.

Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, USA.

Nesterov, Y. and Nemirovskii, A. (1994). *Interior Point Polynomial Methods in Convex Programming*. SIAM.

Niebling, J. and Eichfelder, G. (2016). A branch-and-bound algorithm for bi-objective problems. In *Proceedings of the XIII Global Optimization Workshop*, pages 57–60.

Niebling, J. and Eichfelder, G. (2019). A branch–and–bound-based algorithm for nonconvex multiobjective optimization. *SIAM Journal on Optimization*, 29(1):794–821.

Nocedal, J. and Wright, S. J. (2006). *Sequential Quadratic Programming*, pages 529–562. Springer New York.

Orman, A. and Williams, H. (2007). A survey of different integer programming formulations of the travelling salesman problem. In Kontoghiorghes, E. J. and Gatu, C., editors, *Optimisation, Econometric and Financial Analysis*, pages 91–104. Springer Berlin Heidelberg.

Ostrowski, J., Anjos, M. F., and Vannelli, A. (2015). Modified orbital branching for structured symmetry with an application to unit commitment. *Mathematical Programming*, 150(1):99–129.

Ostrowski, J., Linderoth, J., Rossi, F., and Smriglio, S. (2011). Orbital branching. *Mathematical Programming*, 126(1):147–178.

Papageorgiou, D. J. and Trespalacios, F. (2018). Pseudo basic steps: bound improvement guarantees from Lagrangian decomposition in convex disjunctive programming. *EURO Journal on Computational Optimization*, 6(1):55–83.

Park, M., Park, S., Mele, F. D., and Grossmann, I. E. (2006). Modeling of purchase and sales contracts in supply chain optimization. In *2006 SICE-ICASE International Joint Conference*, pages 5727–5732.

Pisinger, D. and Sigurd, M. (2007). Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS Journal on Computing*, 19(1):36–51.

Plaisted, D. A. and Greenbaum, S. (1986). A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304.

Puget, J.-F. (1993). On the satisfiability of symmetrical constrained satisfaction problems. In Komorowski, J. and Raś, Z. W., editors, *Methodologies for Intelligent Systems*, pages 350–361. Springer Berlin Heidelberg.

Puget, J.-F. (2002). Symmetry breaking revisited. In Van Hentenryck, P., editor, *Principles and Practice of Constraint Programming - CP 2002*, pages 446–461. Springer Berlin Heidelberg.

Puget, J.-F. (2005a). Breaking symmetries in all different problems. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 272âĂŞ–277, San Francisco, USA. Morgan Kaufmann Publishers Inc.

Puget, J.-F. (2005b). Symmetry breaking revisited. *Constraints*, 10(1):23–46.

Quesada, I. and Grossmann, I. E. (1993). Global optimization algorithm for heat exchanger networks. *Industrial & Engineering Chemistry Research*, 32(3):487–499.

Rado, F. (1963). Linear programming with logic conditions. *Communicarile Academiei Republicil Populare Romine*, 13:1039–1041.

Raman, R. and Grossmann, I. E. (1994). Modelling and computational techniques for logic based integer programming. *Computers & Chemical Engineering*, 18(7):563–578.

Ridgeway, G. (2017). Package 'gbm'.

Rockafellar, R. T. (1970). *Convex Analsis*. Princeton University Press.

Rodriguez, M. A. and Vecchietti, A. (2009). Logical and generalized disjunctive programming for supplier and contract selection under provision uncertainty. *Industrial & Engineering Chemistry Research*, 48(11):5506–5521.

Rossi, F., Van Beek, P., and Walsh, T. (2006). *Handbook of constraint programming*. Elsevier.

Rostami, M. and Bagherpour, M. (2017). A Lagrangian relaxation algorithm for facility location of resource-constrained decentralized multi-project scheduling problems. *Operations Research*.

Ruiz, J. P. and Grossmann, I. E. (2010). Strengthening of lower bounds in the global optimization of bilinear and concave generalized disjunctive programs. *Computers & Chemical Engineering*, 34(6):914–930.

Ruiz, J. P. and Grossmann, I. E. (2012). A hierarchy of relaxations for nonlinear convex generalized disjunctive programming. *European Journal of Operational Research*, 218(1):38–47.

Ruiz, J. P. and Grossmann, I. E. (2013). Using convex nonlinear relaxations in the global optimization of nonconvex generalized disjunctive programs. *Computers & Chemical Engineering*, 49:70–84.

Ryoo, H. S. and Sahinidis, N. V. (1996). A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138.

Sahinidis, N. V. (1996). BARON: general purpose global optimization software package. *Journal of Global Optimization*, 8(2):201–205.

Sawaya, N. (2006). *Reformulations, relaxations and cutting planes for generalized disjunctive programming*. PhD thesis, Carnegie Mellon University.

Sawaya, N. and Grossmann, I. (2012). A hierarchy of relaxations for linear generalized disjunctive programming. *European Journal of Operational Research*, 216(1):70–82.

Sawaya, N. W. and Grossmann, I. E. (2005). A cutting plane method for solving linear generalized disjunctive programming problems. *Computers & Chemical Engineering*, 29(9):1891–1913.

Sawaya, N. W. and Grossmann, I. E. (2007). Computational implementation of non-linear convex hull reformulation. *Computers & Chemical Engineering*, 31(7):856–866.

Schulte, C. and Stuckey, P. J. (2004). Speeding up constraint propagation. In Wallace, M., editor, *Principles and Practice of Constraint Programming – CP 2004*, pages 619–633. Springer Berlin Heidelberg.

Schulte, C. and Stuckey, P. J. (2008). Efficient constraint propagation engines. *ACM Trans. Program. Lang. Syst.*, 31(1).

Schweidtmann, A. M. and Mitsos, A. (2019). Deterministic global optimization with artificial neural networks embedded. *Journal of Optimization Theory and Applications*, 180(3):925–948.

Sebastiani, R. (2007). Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3:141–224.

Sebastiani, R. and Tomasi, S. (2015). Optimization modulo theories with linear rational costs. *ACM Transactions on Computational Logic*, 16(2):1–43.

Sebastiani, R. and Trentin, P. (2015). *Computer Aided Verification: 27th International Conference, CAV 2015*, chapter OptiMathSAT: A Tool for Optimization Modulo Theories, pages 447–454. Springer International Publishing.

Sharma, S., Knudsen, B. R., and Grimstad, B. (2016). Towards an objective feasibility pump for convex MINLPs. *Computational Optimization and Applications*, 63(3):737–753.

Shostak, R. E. (1979). A practical decision procedure for arithmetic with function symbols. *Journal of the ACM*, 26(2):351–360.

Shostak, R. E. (1982). *6th Conference on Automated Deduction: New York, USA*, chapter Deciding combinations of theories, pages 209–222. Springer Berlin Heidelberg, Berlin, Heidelberg.

Silva, J. P. M. and Sakallah, K. A. (1996). Grasp: a new search algorithm for satisfiability. In *IEEE/ACM International Conference on Computer-aided Design*, ICCAD '96, pages 220–227, Washington, DC, USA. IEEE Computer Society.

Singer, A. B., Taylor, J. W., Barton, P. I., and Green, W. H. (2006). Global dynamic optimization for parameter estimation in chemical kinetics. *The Journal of Physical Chemistry A*, 110(3):971–976.

Sitek, P. (2014). A hybrid CP/MP approach to supply chain modelling, optimization and analysis. In *Computer Science and Information Systems (FedCSIS)*, pages 1345–1352.

Slater, M. (1959). Lagrange multipliers revisited. Cowles Foundation Discussion Papers 80, Cowles Foundation for Research in Economics, Yale University.

Smith, E. M. and Pantelides, C. C. (1997). Global optimisation of nonconvex MINLPs. *Computers & Chemical Engineering*, 21:S791–S796. Supplement to Computers and Chemical Engineering.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc.

Sra, S., Nowozin, S., and Wright, S. J. (2012). *Optimization for Machine Learning*. MIT Press.

Stallman, R. M. and Sussman, G. J. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135–196.

Stubbs, R. A. and Mehrotra, S. (1999). A branch-and-cut method for 0-1 mixed convex programming. *Mathematical Programming*, 86(3):515–532.

Tanaka, S. and Araki, M. (2008). A branch-and-bound algorithm with Lagrangian relaxation to minimize total tardiness on identical parallel machines. *Internation Journal of Production Economics*, 113(1):446–458.

Tawarmalani, M. and Sahinidis, N. (2002). *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming*. Springer, US.

Tawarmalani, M. and Sahinidis, N. V. (2005). A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103:225–249.

Thorsteinsson, E. S. (2001). Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. *Principles and Practice of Constraint Programming - CP 2001*, 2239:16–30.

Trespalacios, F. and Grossmann, I. E. (2014). Review of mixed-integer nonlinear and generalized disjunctive programming methods. *Chemie Ingenieur Technik*, 86(7).

Trespalacios, F. and Grossmann, I. E. (2015a). Algorithmic approach for improved mixed-integer reformulations of convex generalized disjunctive programs. *INFORMS Journal on Computing*, 27(1):59–74.

Trespalacios, F. and Grossmann, I. E. (2015b). Improved big-M reformulation for generalized disjunctive programs. *Computers & Chemical Engineering*, 76:98–103.

Trespalacios, F. and Grossmann, I. E. (2016a). Cutting plane algorithm for convex generalized disjunctive programs. *INFORMS Journal on Computing*, 28(2):209–222.

Trespalacios, F. and Grossmann, I. E. (2016b). Lagrangean relaxation of the hull-reformulation of linear generalized disjunctive programs and its use in disjunctive branch and bound. *European Journal of Operational Research*, 253(2):314–327.

Tsang, E. (1993). *Foundations of Constraint Satisfaction*. Academic Press, London, UK.

Tseitin, G. S. (1983). *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer Berlin Heidelberg.

Türkay, M. and Grossmann, I. E. (1996). Logic-based MINLP algorithms for the optimal synthesis of process networks. *Computers & Chemical Engineering*, 20(8):959–978.

van Beek, P. (2006). Chapter 4 - backtracking search algorithms. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 85–134. Elsevier.

Van Hentenryck, P. (1989). *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge, MA, USA.

Van Hentenryck, P., Saraswat, V., and Deville, Y. (1998). Design, implementation, and evaluation of the constraint language cc(fd). *The Journal of Logic Programming*, 37(1):139–164.

Vanderbeck, F. (2011). Branching in branch-and-price: a generic scheme. *Mathematical Programming*, 130(2):249–294.

Vaswani, N., Bouwmans, T., Javed, S., and Narayanamurthy, P. (2018). Robust subspace learning: Robust pca, robust subspace tracking, and robust subspace recovery. *IEEE Signal Processing Magazine*, 35(4):32–55.

Vecchietti, A., Lee, S., and Grossmann, I. E. (2003). Modeling of discrete/continuous optimization problems: Characterization and formulation of disjunctions and their relaxations. *Computers & Chemical Engineering*, 27(3):433–448.

Verwer, S., Zhang, Y., and Ye, Q. C. (2017). Auction optimization using regression trees and linear models as integer programs. *Artificial Intelligence*, 244:368–395. Combining Constraint Solving with Mining and Learning.

Vielma, J. P., Ahmed, S., and Nemhauser, G. (2010). Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions. *Operations Research*, 58(2):303–315.

Vielma, J. P., Dunning, I., Huchette, J., and Lubin, M. (2017). Extended formulations in mixed integer conic quadratic programming. *Mathematical Programming Computation*, 9(3):369–418.

Vigerske, S. (2012). *Decomposition in Multistage Stochastic Programming and a Constraint Integer Programming Approach to Mixed-Integer Nonlinear Programming*. PhD in Mathematics, Humboldt-University Berlin.

Vo-Thanh, N., Jans, R., Schoen, E. D., and Goos, P. (2018). Symmetry breaking in mixed integer linear programming formulations for blocking two-level orthogonal experimental designs. *Computers & Operations Research*, 97:96–110.

Wallace, R. J. and Freuder, E. C. (1992). Ordering heuristics for arc consistency algorithms. In *Proceedings of the Ninth Canadian Conference on Artificial Intelligence*, pages 163–169.

Wechsung, A., Schaber, S. D., and Barton, P. I. (2014). The cluster problem revisited. *Journal of Global Optimization*, 58(3):429–438.

Westerlund, T. and Pettersson, F. (1995). An extended cutting plane method for solving convex MINLP problems. *Computers & Chemical Engineering*, 19:131–136.

Wilder, B., Dilkina, B., and Tambe, M. (2019). Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, pages 1658–1666.

Williams, H. P. (1990). *Model building in mathematical programming*. Wiley.

Würtz, J. and Müller, T. (1996). Constructive disjunction revisited. In Görz, G. and Hölldobler, S., editors, *KI-96: Advances in Artificial Intelligence*, pages 377–386. Springer Berlin Heidelberg.

Xiang, Y., Gubian, S., Suomela, B., and Hoeng, J. (2013). Generalized simulated annealing for efficient global optimization: the GenSA package for R. *The R Journal*, 5.

Yee, T. and Grossmann, I. (1990). Simultaneous optimization models for heat integrationâĂŤII. heat exchanger network synthesis. *Computers & Chemical Engineering*, 14(10):1165–1184.

Yeh, I.-C. (1998). Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research*, 28(12):1797–1808.

Zhang, H. (2002). Generating college conference basketball schedules by a SAT solver. In *Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing*, pages 281–291.

Zhang, L., Madigan, C. F., Moskewicz, M. H., and Malik, S. (2001). Efficient conflict driven learning in a Boolean satisfiability solver. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*, pages 279–285.

Zhu, W. (2006). Unsolvability of some optimization problems. *Applied Mathematics and Computation*, 174(2):921–926.

Appendix A

Hull reformulation of Constrained Layout Convex Generalized Disjunctive Programming Formulation

We state the hull reformulation of the constrained layout convex GDP Problem 6.12 (Sawaya, 2006).

$$
\begin{aligned}
\min \quad & \sum_{\substack{i_1,i_2\in[n_r]\\ i_1<i_2}} F_{i_1,i_2}\left(d_1^{i_1,i_2}+d_2^{i_1,i_2}\right) && \\
\text{subject to} \quad & d_{1,i_1,i_2} \geq x_{1,i_1}-x_{1,i_2}, && \forall i_1,i_2\in[n_r], i_1<i_2\\
& d_{1,i_1,i_2} \geq x_{1,i_2}-x_{1,i_1}, && \forall i_1,i_2\in[n_r], i_1<i_2\\
& d_{2,i_1,i_2} \geq x_{2,i_1}-x_{2,i_2}, && \forall i_1,i_2\in[n_r], i_1<i_2\\
& d_{2,i_1,i_2} \geq x_{2,i_2}-x_{2,i_1}, && \forall i_1,i_2\in[n_r], i_1<i_2\\
& \nu_{1,i_1,i_2,i_1,1}+\frac{W_{i_1}}{2}y^r_{i_1,i_2,1} \leq \nu_{1,i_1,i_2,i_2,1}+\frac{W_{i_2}}{2}y^r_{i_1,i_2,1}, && \forall i_1,i_2\in[n_r], i_1<i_2\\
& \nu_{1,i_1,i_2,i_2,2}+\frac{W_{i_2}}{2}y^r_{i_1,i_2,2} \leq \nu_{1,i_1,i_2,i_1,2}+\frac{W_{i_1}}{2}y^r_{i_1,i_2,2}, && \forall i_1,i_2\in[n_r], i_1<i_2\\
& \nu_{2,i_1,i_2,i_1,3}+\frac{H_{i_1}}{2}y^r_{i_1,i_2,3} \leq \nu_{2,i_1,i_2,i_2,3}+\frac{H_{i_2}}{2}y^r_{i_1,i_2,3}, && \forall i_1,i_2\in[n_r], i_1<i_2\\
& \nu_{2,i_1,i_2,i_2,4}+\frac{H_{i_2}}{2}y^r_{i_1,i_2,4} \leq \nu_{2,i_1,i_2,i_1,4}+\frac{H_{i_1}}{2}y^r_{i_1,i_2,4}, && \forall i_1,i_2\in[n_r], i_1<i_2
\end{aligned}
$$

$$\sum_{j\in\{1,2,3,4\}} y^r_{i_1,i_2,j} = 1, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2$$

$$x_{1,i_1} = \sum_{j\in\{1,2,3,4\}} \nu_{1,i_1,i_2,i_1,j}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2$$

$$x_{1,i_2} = \sum_{j\in\{1,2,3,4\}} \nu_{1,i_1,i_2,i_2,j}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2$$

$$x_{2,i_1} = \sum_{j\in\{1,2,3,4\}} \nu_{2,i_1,i_2,i_1,j}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2$$

$$x_{2,i_2} = \sum_{j\in\{1,2,3,4\}} \nu_{2,i_1,i_2,i_2,j}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2$$

$$x^L_{1,i_1} y^r_{i_1,i_2,j} \leq \nu_{1,i_1,i_2,i_1,j} \leq x^U_{1,i_1} y^r_{i_1,i_2,j}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2, j \in [4]$$

$$x^L_{1,i_2} y^r_{i_1,i_2,j} \leq \nu_{1,i_1,i_2,i_2,j} \leq x^U_{1,i_2} y^r_{i_1,i_2,j}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2, j \in [4]$$

$$x^L_{2,i_1} y^r_{i_1,i_2,j} \leq \nu_{2,i_1,i_2,i_1,j} \leq x^U_{2,i_1} y^r_{i_1,i_2,j}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2, j \in [4]$$

$$x^L_{2,i_2} y^r_{i_1,i_2,j} \leq \nu_{2,i_1,i_2,i_2,j} \leq x^U_{2,i_2} y^r_{i_1,i_2,j}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2, j \in [4]$$

$$y^c_{i,j}\left(\nu_{1,i,j} - \frac{W_i}{2} - c_{1,j}\right)^2 + y^c_{i,j}\left(\nu_{2,i,j} - \frac{H_i}{2} - c_{2,j}\right)^2 \leq r_j^2 y^c_{i,j}, \qquad \forall, i \in [n_r], j \in [n_c]$$

$$y^c_{i,j}\left(\nu_{1,i,j} - \frac{W_i}{2} - c_{1,j}\right)^2 + y^c_{i,j}\left(\nu_{2,i,j} + \frac{H_i}{2} - c_{2,j}\right)^2 \leq r_j^2 y^c_{i,j}, \qquad \forall, i \in [n_r], j \in [n_c]$$

$$y^c_{i,j}\left(\nu_{1,i,j} + \frac{W_i}{2} - c_{1,j}\right)^2 + y^c_{i,j}\left(\nu_{2,i,j} - \frac{H_i}{2} - c_{2,j}\right)^2 \leq r_j^2 y^c_{i,j}, \qquad \forall, i \in [n_r], j \in [n_c]$$

$$y^c_{i,j}\left(\nu_{1,i,j} + \frac{W_i}{2} - c_{1,j}\right)^2 + y^c_{i,j}\left(\nu_{2,i,j} + \frac{H_i}{2} - c_{2,j}\right)^2 \leq r_j^2 y^c_{i,j}, \qquad \forall, i \in [n_r], j \in [n_c]$$

$$x_{1,i} = \sum_{j\in[n_c]} \nu_{1,i,j}, \qquad \forall i \in [n_r]$$

$$x_{2,i} = \sum_{j\in[n_c]} \nu_{2,i,j}, \qquad \forall i \in [n_r]$$

$$x^L_{1,i} y^c_{i,j} \leq \nu_{1,i,j} \leq x^U_{1,i} y^c_{i,j}, \qquad \forall i \in [n_r], j \in [n_c]$$

$$x^L_{2,i} y^c_{i,j} \leq \nu_{2,i,j} \leq x^U_{2,i} y^c_{i,j}, \qquad \forall i \in [n_r], j \in [n_c]$$

$$\sum_{j\in[n_c]} y^c_{i,j} = 1, \qquad \forall i \in [n_r]$$

$$x^L_{1,i} \leq x_{1,i} \leq x^U_{1,i}, x^L_{2,i} \leq x_{2,i} \leq x^U_{2,i}, \qquad \forall i \in [n_r]$$

$$d_{1,i_1,i_2}, d_{2,i_1,i_2} \in \mathbb{R}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2$$

$$x_{1,i}, x_{2,i} \in \mathbb{R}, \qquad \forall i \in [n_r]$$

$$y^r_{i_1,i_2,1}, y^r_{i_1,i_2,2}, y^r_{i_1,i_2,3}, y^r_{i_1,i_2,4} \in \{0,1\}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2$$

$$\nu_{1,i_1,i_2,i_1,j}, \nu_{1,i_1,i_2,i_2,j}, \nu_{2,i_1,i_2,i_1,j}, \nu_{2,i_1,i_2,i_2,j} \in \mathbb{R}, \qquad \forall i_1, i_2 \in [n_r], i_1 < i_2, j \in [4]$$

$$\nu_{1,i,j}, \nu_{2,i,j} \in \mathbb{R}, \qquad \forall i \in [n_r], j \in [n_c].$$