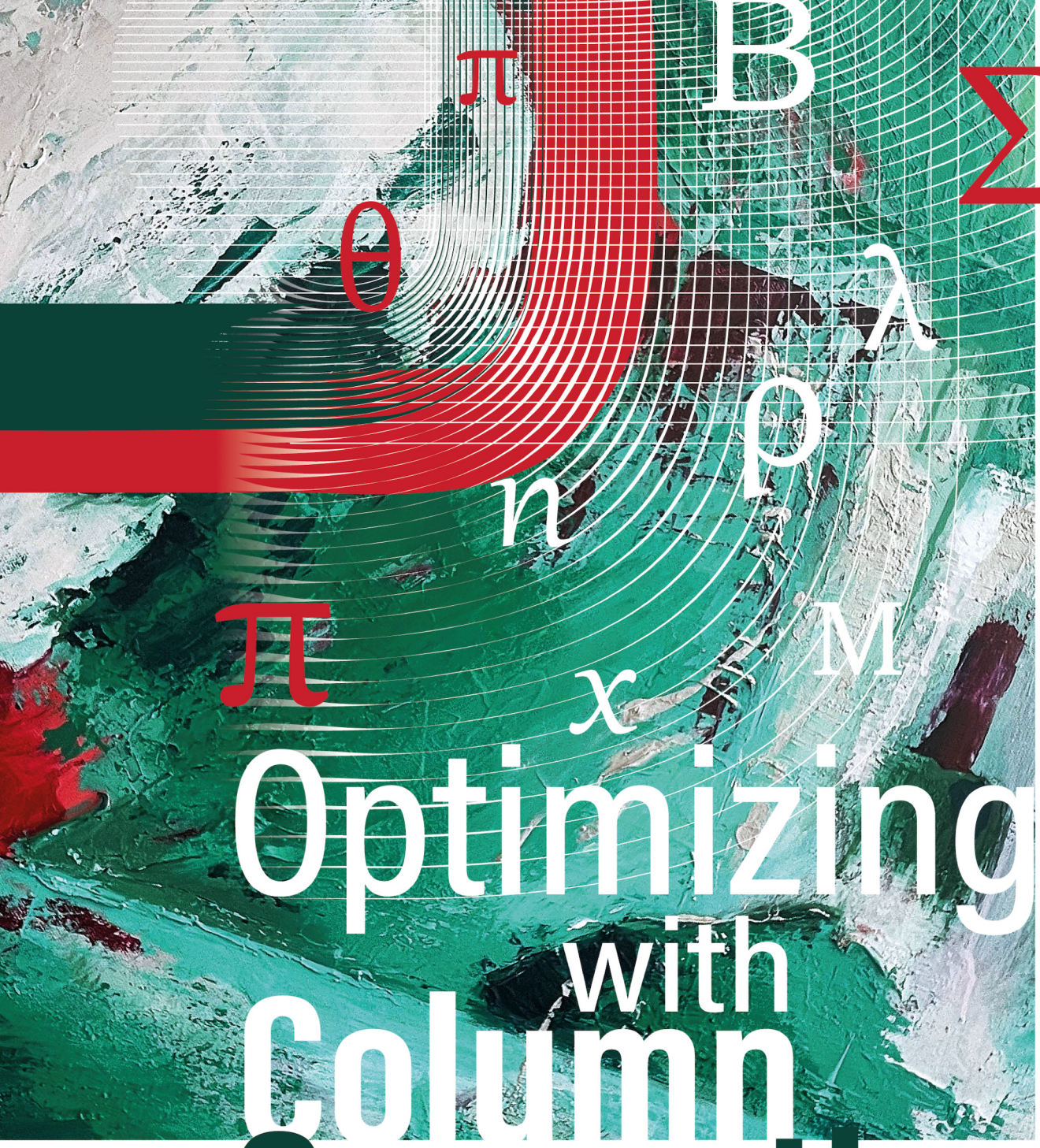


Optimizing with Column Generation

Eduardo Uchoa | Artur Pessoa | Lorenza Moreno

Cadernos do LOGIS

Optimizing with Column Generation:

Advanced Branch-Cut-and-Price Algorithms

Part I

Eduardo Uchoa, Artur Pessoa
and Lorenza Moreno

Volume 2024, Number 3
August, 2024

LOGIS
Núcleo de Logística Integrada e Sistemas

Universidade Federal Fluminense

Optimizing with Column Generation

Advanced Branch-Cut-and-Price Algorithms

Eduardo Uchoa
Universidade Federal Fluminense
INRIA International Chair (2022–2026)

Artur Pessoa
Universidade Federal Fluminense

Lorenza Moreno
Universidade Federal de Juiz de Fora

Optimizing with Column Generation

Book Cover by Leonardo Viana over his acrylic on canvas painting “Ciclo”
Available at: https://OptimizingWithColumnGeneration.github.io

A Note to Our Readers

We are excited to present the early release of Part I of our book "Optimizing with Column Generation: advanced Branch-Cut-and-Price Algorithms". While the book's ultimate goal, as suggested by its subtitle, is to describe cutting-edge techniques in these algorithms, this objective is primarily addressed in the forthcoming Part II. However, we feel that the completed first part, covering the fundamentals of Column Generation and representing nearly two years of dedicated work, is already a valuable contribution to the community.

The intentions of this early release go further. We have set up a GitHub repository (https://OptimizingWithColumnGeneration.github.io) where readers can ask for help from other readers, share solutions to exercises (perhaps coded in different programming languages), and have discussions. They may also report errors and suggest topics to be covered in Part II. By observing that community feedback, we believe the final book will be significantly better than if written in isolation.

Thank you for your interest and support!

August 29th, 2024

Contents

Optimizing with Column Generation

Acknowledgments

We begin by acknowledging some significant contributions by Ruslan Sadykov to the text of this book, mainly in the historical research on the Column Generation technique. Notably, he co-authored the rediscovery of Kantorovich and Zalgaller's 1951 book (in Russian) on the Cutting Stock Problem, which is revealed to contain a complete column generation algorithm.

We express our heartfelt gratitude to Marcus Poggi de Aragão, Ruslan Sadykov, and François Vanderbeck for more than a decade of highly productive collaboration, involving numerous exchanges between the Brazilian "Rio de Janeiro–Niterói group" and the French "Bordeaux group". We learned a great deal from them and with them. Indeed, several advances in Branch-Cut-and-Price algorithms described in this book emerged from that partnership. We only wish they could have joined us in writing this book.

We are thankful to Isaac Balster, Teobaldo Bulhões, Vinícius Loti de Lima, Hugo Kramer, Matheus Ota, Eduardo Queiroga, Marcos Roboredo, and João Marcos Pereira Silva for their contributions. They include proofreading, insightful feedback, and efforts in developing solutions for the exercises. Special recognition goes to João for his extensive help on LaTeX issues and for preparing some figures that enrich our text.

We are deeply indebted to our friend and exceptional artist, Leonardo Viana, for designing the book's cover art. Its background is a 1m $\times$ 1m acrylic on canvas named Ciclo, painted by him especially for this book! His creative prowess has added a distinctive visual dimension to our "so-abstract" subject matter. *Muito Obrigado, Leo!*

Lastly, we extend our sincere thanks to Anand Subramanian for his unwavering encouragement throughout this long and challenging project.

U. P. M.

Optimizing with Column Generation

Acronyms

To avoid a too big list, only "non-local acronyms", those that are also used far from where they are first defined in the text, appear here.

BB/BBA	Branch-and-Bound / BB Algorithm
BC/BCA	Branch-and-Cut / BC Algorithm
BCP/BCPA	Branch-Cut-and-Price / BCP Algorithm
BP/BPA	Branch-and-Price / BP Algorithm
BPP	Bin Packing Problem
CG/CGA	Column Generation / CG Algorithm
COP	Combinatorial Optimization Problem
CGC	Chvátal-Gomory Cut
CSP	Cutting Stock Problem
CVRP	Capacitated Vehicle Routing Problem
DP/DPA	Dynamic Programming / DP Algorithm
DW	Dantzig-Wolfe
GAP	Generalized Assignment Problem
GCP	Graph Coloring Problem
IP	Integer Program
ISP	Independent Set Problem
KGG	Kantorovich-Gilmore-Gomory
KP	Knapsack Problem
LCOP	Linear Combinatorial Optimization Problem

LDF/LDP	Lagrangian Dual Function / Lagrangian Dual Problem
LHS	Left Hand Side
LP	Linear Program
LR	Lagrangian Relaxation
LS	Lagrangian Subproblem
MIP	Mixed-Integer Program
MLP	Master Linear Program
NFP/MNFP	Network Flow Problem / Multi-commodity NFP
R1C	Rank-1 (Chvátal-Gomory) Cut
RCC	Rounded Capacity Cut
R&FB	Ryan & Foster Branching
RLDP	Restricted Lagrangian Dual Problem
RMLP	Restricted Master Linear Program
RSA	Revised Simplex Algorithm
RHS	Right Hand Side
SCP	Set Covering Problem
SPcP	Set Packing Problem
SPP	Set Partitioning Problem
SPG	Steiner Problem in Graphs
TSP	Traveling Salesperson Problem
VRP	Vehicle Routing Problem
VRPTW	VRP with Time Windows

Notation

Matrices are represented by uppercase boldface letters. Vectors are represented by lowercase boldface letters. Unless stated otherwise, n-dimensional vectors are column vectors associated with points in the $\mathbb{R}^n$ space. Yet, some vectors will be defined as row vectors. A $1 \times n$ row vector may be defined as belonging to $\mathbb{R}^{1\times n}$ and an $m \times n$ matrix as belonging to $\mathbb{R}^{m\times n}$. Matrices and vectors are notated with parentheses (instead of brackets), and the symbol $^\intercal$ denotes matrix transposition. The j-th column in matrix $\boldsymbol{A}$ is denoted as $\boldsymbol{a}_j$ and the element in its i-row and j-th column is denoted as a_{ij} (uppercase bold $\boldsymbol{A}$ becoming lowercase bold $\boldsymbol{a}$ and lowercase a). The i-th element of a vector $\boldsymbol{p}$ (either a row or column vector) is denoted as p_i. Sometimes a vector $\boldsymbol{p}$ is associated with elements of a set N and its elements may also be denoted as p_e, for $e \in N$. In that context, the *incidence vector* of $N' \subseteq N$, denoted as $\boldsymbol{\chi}(N')$, is the $|N|$-dimensional binary vector $\boldsymbol{p}$ where $p_e = 1$ if and only if $e \in N'$. When we have variables or vectors of variables, the symbols *, $\hat{}$, $\bar{}$, $'$, and $''$ may be used to refer to specific values for those variables. In particular, the star symbol * is preferred to indicate some specific optimal solution. The symbols $\mathbf{0}$ and $\mathbf{1}$ represent a vector with appropriate dimensions having 0 and 1, respectively, in all its elements. The p-norm of an n-dimensional vector $\boldsymbol{x}$ is $||\boldsymbol{x}||_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p}$; $||\boldsymbol{x}||$ denotes the 2-norm (a.k.a. Euclidean norm).

Sets are represented by uppercase letters. An exception is the symbol $[\cdot]$ that represents the set of integer numbers from 1 until its positive integer argument. For example, $[n] = \{1, \ldots, n\}$. Using that notation, we write summations like $\sum_{j=1}^{n}$ as $\sum_{j\in[n]}$, which saves vertical space in displayed formulas. The standard sets are $\mathbb{R}$ (real numbers), $\mathbb{R}_+$ (non-negative real numbers), $\mathbb{R}_-$ (non-positive real numbers), $\mathbb{Z}$ (integer numbers), $\mathbb{Z}_+$ (non-negative integer numbers), and $\mathbb{B} = \{0, 1\}$. The interval $\{x \in \mathbb{R} \,|\, a \leq x \leq b\}$ may be denoted as $[a, b]$. The standard set operators $\in$ (in), $\notin$ (not in), $\cup$ (union), $\cap$ (intersection), $\setminus$ (set minus), $\subset$ (subset), $\supset$ (superset) and $|\cdot|$ (set cardinality) are used. If z^* is defined as $\min f(\boldsymbol{x})$ subject to $\boldsymbol{x} \in X$, then,

by convention, $z^* = \infty$ if $X = \emptyset$ and $z^* = -\infty$ if the minimum is unbounded. If z^* is finite, then $\arg\min f(\boldsymbol{x})$ s.t. $\boldsymbol{x} \in X$ denotes an arbitrary $\boldsymbol{x}^* \in X$ such that $f(\boldsymbol{x}^*) = z^*$. If $z^* = \max f(\boldsymbol{x})$ s.t. $\boldsymbol{x} \in X$, then $z^* = -\infty$ if $X = \emptyset$ and $z^* = \infty$ if the maximum is unbounded; while arg max is defined similarly to arg min. A set $X \subseteq \mathbb{R}^n$ or $X \subseteq \mathbb{R}^{1\times n}$ is *convex* when, for every $\boldsymbol{x}, \boldsymbol{y} \in X$, and $\alpha \in [0,1]$, $(\alpha\boldsymbol{x}+(1-\alpha)\boldsymbol{y}) \in X$. A function $f : X \mapsto \mathbb{R}$, where $X \subseteq \mathbb{R}^n$ or $X \subseteq \mathbb{R}^{1\times n}$ is a convex set, is *convex* when, for every $\boldsymbol{x}, \boldsymbol{y} \in X$, and $\alpha \in [0,1]$, $f(\alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{y}) \leq \alpha f(\boldsymbol{x}) + (1-\alpha)f(\boldsymbol{y})$. By reversing the sense of the previous inequality, we obtain the condition to classify function f as *concave*. When the previous inequalities hold as strict inequalities, we may add the qualifier *strictly* to convex or concave.

Graphs are notated as follows. An undirected graph $G = (V, E)$ is supposed to be simple (without parallel edges or loops), the elements of the edge-set E are sets of the form $e = \{u, v\}$, such that $u, v \in V$. For a vertex-set $S \subseteq V$, its edge cutset is $\delta(S) = \{e \in E : |e \cap S| = 1\}$ and its interior edge set is $E(S) = \{e \in E : e \subseteq S\}$. In contrast, directed graphs are not always simple. For a directed graph $G = (V, A)$ without parallel arcs, the arc-set A has as elements 2-tuples (a.k.a. ordered pairs) $a = (u, v)$, such that $u, v \in V$. Otherwise, arcs in A are represented by 3-tuples $a = (u, v, j)$, where the third element j is an identifier for differentiating parallel arcs between vertices u and v. In that case, the standard graph operators over a vertex-set $S \subseteq V$ are $\delta^+(S) = \{a = (u, v, \cdot) \in A : u \in S, v \notin S\}$, $\delta^-(S) = \{a = (u, v, \cdot) \in A : u \notin S, v \in S\}$, and $A(S) = \{a = (u, v, \cdot) \in A : u, v \in S\}$. Given an undirected graph $G = (V, E)$, we denote by $G_D = (V, A)$ the directed graph obtained by replacing every edge $\{u, v\} \in E$ by a pair of opposite arcs (u, v) and (v, u).

The class $\mathcal{P}$ is formed by the decision problems that can be solved in deterministic polynomial time. A decision problem is said to be $\mathcal{NP}$-complete if: (i) it belongs to class $\mathcal{NP}$ (the decision problems that can be solved in non-deterministic polynomial time, or, equivalently, the decision problems where a "yes" answer always has a certificate that can be checked in polynomial time), and (ii) it is $\mathcal{NP}$-hard (a problem that if solved in deterministic polynomial time would imply that all problems in $\mathcal{NP}$ can also be solved in deterministic polynomial time). This book deals with *optimization problems*, not decision problems. A widespread and time-honored convention [Garey and Johnson, 1979] is that the optimization problems whose decision versions are $\mathcal{NP}$-complete, like the Traveling Salesperson Problem (TSP), should be classified as $\mathcal{NP}$-hard, never as $\mathcal{NP}$-complete. That convention

is certainly correct but has the following drawback: problems that are likely to be much harder than the TSP, even proved undecidable problems, are also $\mathcal{NP}$-hard. There is no standard notation for differentiating between the "ordinary $\mathcal{NP}$-hard optimization problems" from some harder optimization problems that are also considered in this book. For example, some separation problems where only checking if a separated cut is indeed valid is already an $\mathcal{NP}$-hard problem. Two authors of this book suggested *defining* "optimization problem X is $\mathcal{NP}$-complete" as a *shorthand* to "the decision version of optimization problem X is $\mathcal{NP}$-complete". That suggestion made a third author cringe in horror. So, we decided to abide by the convention. However, whenever the decision version of some $\mathcal{NP}$-hard optimization problem is not $\mathcal{NP}$-complete that will be indicated. However, we may commit the abuse of saying that an optimization problem X belongs to $\mathcal{P}$ if its decision version belongs to $\mathcal{P}$.

Optimizing with Column Generation

Introduction

Column Generation is a method to solve linear programming problems with a very large number of variables. It dynamically generates variables (and the corresponding matrix columns, hence the name) by solving auxiliary optimization problems known as pricing subproblems. It can also be very effective in integer programming, forming the backbone of algorithms like Branch-and-Price and Branch-Cut-and-Price. This technique has been successfully applied to many types of vehicle routing, cutting and packing, airline planning, timetabling, crew scheduling, graph coloring, clustering, lot sizing, and machine scheduling, among other problems. In many of those cases, it vastly outperforms all competing methods including compact formulations solved by general MIP solvers and specialized Branch-and-Cut algorithms.

Column Generation is a thriving field, with hundreds of relevant papers published annually. The current most advanced Branch-Cut-and-Price algorithms incorporate several recently proposed elements and are much more powerful than the typical Branch-and-Price algorithms of 20 years ago. All major Operations Research conferences have sessions on it and there is even a periodical Column Generation Workshop. Column Generation also found its way into the industry, where it is routinely applied (usually as part of highly effective heuristics, optimality being a secondary concern) for handling complex optimization problems where many millions of dollars are at stake.

Yet, paradoxically, Column Generation is still a well-kept secret.

Every time we receive new students who are already acquainted with linear and integer programming but want to start working with Branch-Cut-and-Price algorithms, their first question is: "Which book should I read?". Our answer: "There are chapters and surveys that you definitely must read and will provide a nice overview of Column Generation. However, to understand the techniques used in the advanced Branch-Cut-and-Price algorithms, you will also need to read many research articles." This is not the ideal. Research articles are not beginner-friendly, they assume a lot of previous knowledge. Moreover, each article uses its own mathe-

matical notation. Even some basic concepts and nomenclature are not standardized, which can be confusing.

This educational barrier is particularly serious due to another major obstacle: computational implementation. While current commercial MIP solvers (in alphabetic order: COPT, CPLEX, Gurobi and Xpress) offer superior performance and convenient interfaces for implementing Branch-and-Cut algorithms, at the time of writing, they do not support Branch-and-Price or Branch-Cut-and-Price. While existing open-source frameworks (like ABACUS, BaPCod, Coluna, DIP, and SCIP GCG) may help a lot with building those advanced algorithms, they still have serious limitations that may force their users to code if they want to achieve state-of-the-art performance.

As a result of these difficulties, the technique is much less used than it could be. *The primary goal of this book is to lower these educational barriers and encourage wider application of Column Generation, not only organizing and summarizing existing literature but also sharing insights from decades of practical experience, offering unique guidance not found elsewhere.* We aim to provide not just theoretical explanations, but also advice on effective implementation. Throughout the book, readers will find selected examples and case studies that demonstrate the potential of applying column generation in diverse scenarios. We also provide context and commentary on how the field has evolved, highlighting key innovations and shifts in thinking.

Before explaining the book's organization, let us consider two seemly contradictory quotes:

> *"If you can't explain it simply, you don't understand it well enough."*
> — Folklore, often apocryphally attributed to Albert Einstein or Richard Feynman

> *"If I could explain it to the average person, it wouldn't have been worth the Nobel Prize!"* — What Feymann really said

Both quotes contain elements of truth. We did our best efforts to make this book simple and accessible to its target audience which not only includes fellow researchers but also students (even undergrads, if sufficiently motivated) and optimization practitioners in the industry. Those efforts also align with our view that mathematical complexity is not always required for rigor and is not necessarily an indicator of

mathematical depth. Yet, there are limits to the simplicity. We do not claim that column generation is as difficult as quantum electrodynamics, but the subject is extensive and some points in it are subtle. There are advanced concepts that can only be understood after more basic material is mastered. The proposed book organization is intended to avoid overburdening the reader with too much information at once. Its first level is the division into two parts:

- Part I — Column Generation Basics — comprises five chapters covering the fundamental principles of Column Generation. Those chapters are devised to be read in sequence. This part is more beginner-friendly, with concepts illustrated by detailed numerical examples. All mathematical proofs are short and relatively simple.
- Part II — Topics in Column Generation — consists of eight chapters. Some of those chapters present the more advanced techniques that play a crucial role in state-of-the-art Branch-Cut-and-Price algorithms. Part II assumes that the reader is familiar with the material in Part I. Otherwise, its chapters can be read in any order, according to the interests of the reader.

The second level of organization is intra-chapter. Each chapter contains:

- A main text, where the most essential material is developed. A chapter is divided into sections and subsections that are intended to be read in sequence. Some of those sections are case studies. The main text of a chapter concludes with a boxed text summarizing its contents and implications, in a less formal language.
- A series of notes. A few of these are brief, merely providing appropriate credit for the results referred to in the main text. However, most notes are not so brief and present important content not included in the main text to avoid overloading it. Some notes even take the form of mini-articles, providing more personal perspectives from the authors on specific topics or in-depth historical explorations. Notably, Note 4.10 is about our "rediscovery" while writing this book of the first Column Generation algorithm, published (in Russian) in 1951 and largely unknown in the Western world. Notes may be read in any order and may be skipped in the first reading of the chapter.

- A list of exercises. There are *Numerical Exercises* involving the application of the learned techniques to toy instances. The *Conceptual Exercises* are more algebraic. In fact, some of those exercises provide additional content to the book. For example, a certain theorem may have been proved by us only for the most typical case. Proofs for other more generic cases may be asked as exercises. This can be doubly beneficial. First, it avoids burdening the main text or the notes with more complex and technical proofs (needed to cover all the cases) that may obscure their essential argument. Second, it gives the readers an opportunity to actively test their understanding of the presented proofs by generalizing them. There are some *Open Exercises*, characterized by not having a single and clear correct answer, where we may ask the reader to discuss some point. Finally, there are *Project Exercises* where we ask for complete implementations of certain column generation based methods for some classic problems. Unless the reader is willing to code extensively, it is recommended to employ some framework for helping with those exercises. By the way, the last chapter presents an overview of existing Column Generation frameworks. *The book has a companion website where answers to most exercises can be found.*

Overall, the authors view writing this book as a culmination of their years of struggling research and hands-on experience, motivated by a deep passion for the field. We hope that it will be useful to both newcomers seeking to learn column generation and seasoned practitioners looking to deepen their understanding and enhance their skills.

Part I

Column Generation Basics

Chapter 1

The Revised Simplex Algorithm

This chapter is not intended to work as a first exposition to linear programming. There are several excellent textbooks for that, including Bertsimas and Tsitsiklis [1997], Bazaraa et al. [2010], Vanderbei [2014]. The authors of this book are particularly fond of Chvátal [1983]. We present here the revised simplex algorithm because we believe that it provides the most accessible entrance door to column generation.

In 1947, George B. Dantzig developed the simplex algorithm for solving linear programming problems. In 1953, Dantzig himself proposed the revised simplex algorithm. As in the original simplex, revised simplex assumes that the problem to be solved is first converted into standard format: all variables should be non-negative and all the remaining constraints are equalities. For example, the following LP (Linear Program),

$$
\begin{array}{lrcrcrcrcr}
\min z = & 24x_1 & + & 29x_2 & + & 10x_3 & + & 38x_4 & & \\
\text{s.t.} & x_1 & + & 4x_2 & + & 5x_3 & & & = & 60 \\
 & & & 2x_2 & + & x_3 & & & \leq & 12 \\
 & 2x_1 & + & x_2 & - & x_3 & + & 4x_4 & \geq & 10 \\
 & x_1, & & x_2, & & x_3, & & x_4 & \geq & 0,
\end{array}
$$

can be put into standard format by adding slack variable x_5 and surplus variable x_6:

$$
\begin{array}{lrcrcrcrcrcrcr}
\min z = & 24x_1 & + & 29x_2 & + & 10x_3 & + & 38x_4 & & & & & \\
\text{s.t.} & x_1 & + & 4x_2 & + & 5x_3 & & & & & & = & 60 \\
 & & & 2x_2 & + & x_3 & & & + & x_5 & & = & 12 \\
 & 2x_1 & + & x_2 & - & x_3 & + & 4x_4 & & & - \; x_6 & = & 10 \\
 & x_1, & & x_2, & & x_3, & & x_4, & & x_5, & x_6 & \geq & 0,
\end{array}
$$

A general LP in standard format can be written in matrix notation as:

$$\min z = \boldsymbol{c}\boldsymbol{x} \tag{1.1}$$

$$\text{s.t.} \qquad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{1.2}$$

$$\boldsymbol{x} \geq \boldsymbol{0}, \tag{1.3}$$

where the objective function vector (a.k.a. cost vector) $\boldsymbol{c}$ has dimension $1 \times n$, variable vector $\boldsymbol{x}$ has dimension $n \times 1$, constraint coefficient matrix $\boldsymbol{A}$ has dimension $m \times n$, and Right Hand Side (RHS) vector $\boldsymbol{b}$ has dimension $m \times 1$. In the above example, the matrices are:

$$\boldsymbol{c} = \begin{pmatrix} 24 & 29 & 10 & 38 & 0 & 0 \end{pmatrix} \qquad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} \qquad \boldsymbol{b} = \begin{pmatrix} 60 \\ 12 \\ 10 \end{pmatrix}.$$

$$\boldsymbol{A} = \begin{pmatrix} 1 & 4 & 5 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 2 & 1 & -1 & 4 & 0 & -1 \end{pmatrix}$$

In a slight abuse of notation, we may omit the transpose symbol in text and write a column vector like $\boldsymbol{b}$ as $(\,60\ 12\ 10\,)$, rather than $(\,60\ 12\ 10\,)^\intercal$. The objective function value z is also known as the solution cost.

Definition 1.1: Basis, basic feasible solution, basic variables, degeneracy. Let $(\boldsymbol{B}\ \boldsymbol{N})$ be a partition of the columns in $\boldsymbol{A}$, such that $\boldsymbol{B}$ has dimension $m \times m$ and is invertible. In those conditions, submatrix $\boldsymbol{B}$ is said to define a *basis*. Let $\boldsymbol{x} = (\boldsymbol{x_B}\ \boldsymbol{x_N})$ and $\boldsymbol{c} = (\boldsymbol{c_B}\ \boldsymbol{c_N})$ be the corresponding partitions of components of $\boldsymbol{x}$ and $\boldsymbol{c}$. Variables in $\boldsymbol{x_B}$ are *basic variables*, those in $\boldsymbol{x_N}$ are *non-basic variables*. The corresponding *basic solution* is $\boldsymbol{x} = (\boldsymbol{x_B} = \boldsymbol{B^{-1}b}\ \ \boldsymbol{x_N} = \boldsymbol{0})$, which is *feasible* if $\boldsymbol{x_B} \geq \boldsymbol{0}$. A basic feasible solution is *degenerate* if at least one basic variable has value zero.

Theorem 1.1: *If an LP in standard format has optimal solutions then at least one of those optimal solutions should be basic feasible.*

That fundamental result indicates that the search for an optimal solution can be restricted to the finite set of basic feasible solutions. The Revised Simplex Algorithm (RSA) provides a systematic way of moving from a basic feasible solution to another basic feasible solution with a better cost (except perhaps in case of degeneracy) until an optimal solution is found. The following definition is also needed for describing the RSA.

Definition 1.2: Reduced cost. Let $\boldsymbol{\pi} = (\pi_1 \ldots \pi_m)$ be the $1 \times m$ vector with the dual variables associated to Constraints (1.2). The *reduced cost* of variable x_j, $j \in [n] = \{1, \ldots, n\}$, is defined as $\bar{c}_j = c_j - \boldsymbol{\pi} \boldsymbol{a}_j$, where $\boldsymbol{a}_j$ is the j-th column of $\boldsymbol{A}$.

The steps of RSA are now given:

RSA Step 1: Find an initial basic feasible solution.

In general, it may be difficult to find a feasible basis $\boldsymbol{B}$ from the columns of $\boldsymbol{A}$. Indeed, an LP may have no solutions at all, i.e., be infeasible, and that may not be obvious. However, to easily execute Step 1, it is always possible to enlarge $\boldsymbol{A}$ with additional columns corresponding to *artificial variables* with very large (see Note 1.5) positive costs (or very large negative costs in case of a maximization LP). For example, if $\boldsymbol{b} \geq \boldsymbol{0}$, $\boldsymbol{B}$ can be defined as an identity matrix corresponding to m artificial variables.

In our example, suppose that it is discovered (no matter how) that variables x_1, x_3, and x_5 can be chosen to obtain a basic feasible solution, which is the unique solution of the following linear system:

$$\boldsymbol{B}\boldsymbol{x}_{\boldsymbol{B}} = \begin{pmatrix} 1 & 5 & 0 \\ 0 & 1 & 1 \\ 2 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_3 \\ x_5 \end{pmatrix} = \begin{pmatrix} 60 \\ 12 \\ 10 \end{pmatrix} \Leftrightarrow \begin{cases} x_1 + 5x_3 = 60 \\ x_3 + x_5 = 12 \\ 2x_1 - x_3 = 10 \end{cases} \Rightarrow \begin{array}{l} x_1 = 10 \\ x_3 = 10 \\ x_5 = 2. \end{array}$$

The non-basic variables x_2, x_4, and x_6 have value zero and $z = 340$.

RSA Step 2: Calculate the corresponding dual solution.

Step 2 relies on the simplex property that says that the basic variables should have zero reduced cost. This means that $\boldsymbol{\pi}$ should satisfy $\boldsymbol{c}_{\boldsymbol{B}} - \boldsymbol{\pi}\boldsymbol{B} = \boldsymbol{0}$ and, therefore, is the unique solution of the linear system $\boldsymbol{\pi}\boldsymbol{B} = \boldsymbol{c}_{\boldsymbol{B}}$.

In our example, we have that:

$$\boldsymbol{\pi B} = \begin{pmatrix}\pi_1\\ \pi_2\\ \pi_3\end{pmatrix}^{\intercal}\begin{pmatrix}1 & 5 & 0\\ 0 & 1 & 1\\ 2 & -1 & 0\end{pmatrix} = \begin{pmatrix}24\\ 10\\ 0\end{pmatrix}^{\intercal} \Leftrightarrow \begin{cases} \pi_1 + 2\pi_3 = 24 \\ 5\pi_1 + \pi_2 - \pi_3 = 10 \\ \pi_2 = 0 \end{cases} \Rightarrow \begin{array}{l}\pi_1 = 4\\ \pi_2 = 0\\ \pi_3 = 10.\end{array}$$

RSA Step 3 (Pricing): Evaluate the reduced cost of the non-basic variables and choose the entering variable.

Variables with strictly negative reduced cost (strictly positive reduced cost in case of a maximization LP) are eligible for entering the basis. If there is no eligible variable, *the algorithm stops and the current basic feasible solution is optimal*. However, if that solution still contains some artificial variables (that may have been introduced in Step 1) with a positive value, this actually means that *the original problem is infeasible*.

In our example, the reduced costs are:

$$\begin{aligned}\bar{c}_2 &= 29 - 4\pi_1 - 2\pi_2 - \pi_3 = 3\\ \bar{c}_4 &= 38 - 4\pi_3 = -2\\ \bar{c}_6 &= 0 + \pi_3 = 10\end{aligned}$$

This means that only x_4 is eligible and should be chosen for entering the basis.

RSA Step 4: Calculate the direction of change.

The direction of change is the vector $\boldsymbol{d}$ that indicates how much each unit of increase in the entering variable x_j will change the current basic variables in order to keep feasibility. In other words, direction $\boldsymbol{d}$ should be such that $\boldsymbol{B}(\boldsymbol{x_B} - \boldsymbol{d}x_j) + \boldsymbol{a}_j x_j = \boldsymbol{b}$, where $\boldsymbol{a}_j$ is the column of $\boldsymbol{A}$ corresponding to the entering variable. As $\boldsymbol{Bx_B} = \boldsymbol{b}$, $\boldsymbol{Bx_B} - \boldsymbol{Bd}x_j + \boldsymbol{a}_j x_j = \boldsymbol{b}$ is equivalent to $\boldsymbol{Bd}x_j = \boldsymbol{a}_j x_j$. So, $\boldsymbol{d}$ is obtained as the unique solution of the linear system $\boldsymbol{Bd} = \boldsymbol{a}_j$.

In our example:

$$\boldsymbol{Bd} = \begin{pmatrix}1 & 5 & 0\\ 0 & 1 & 1\\ 2 & -1 & 0\end{pmatrix}\begin{pmatrix}d_1\\ d_3\\ d_5\end{pmatrix} = \begin{pmatrix}0\\ 0\\ 4\end{pmatrix} \Rightarrow \boldsymbol{d} = \begin{pmatrix}20/11\\ -4/11\\ 4/11\end{pmatrix}.$$

RSA Step 5: Choose a variable to leave the basis.

One should increase the entering variable as much as possible. Its new value is

given by $x_j = \theta^* = \max\{\theta \in \mathbb{R} : \boldsymbol{x_B} - \theta \cdot \boldsymbol{d} \geq \boldsymbol{0}\}$. The variables that most limited θ^* (i.e., those that will have value zero) are eligible to be chosen to leave the basis. If no variable limits θ^*, *the algorithm stops because the LP is unbounded.* If θ^* is finite and greater than zero, then a new basic feasible solution with a better cost will be obtained. If $\theta^* = 0$, which can only happen if the current solution is degenerate, the resulting basic feasible solution will still have the same cost. By taking some care in the choice of the leaving variable (for example, using the lexicographic anti-cycling rule [Dantzig et al., 1955]), the resulting basic feasible solution will be new even in case of degeneracy. So, it is possible to guarantee that the RSA stops in a finite number of iterations.

In our example:

$$x_4 = \theta^* = \max \theta \text{ such that } \begin{pmatrix} 10 \\ 10 \\ 2 \end{pmatrix} - \theta \begin{pmatrix} 20/11 \\ -4/11 \\ 4/11 \end{pmatrix} \geq \boldsymbol{0} \Longrightarrow x_4 = \theta^* = 5.5.$$

Variables x_1 and x_5 are eligible to leave the basis. Assume that x_1 is chosen.

RSA Step 6: Update $\boldsymbol{B}$ and $\boldsymbol{x_B}$. Go to Step 2.

Basis $\boldsymbol{B}$ is updated by replacing the column of the leaving variable with the column of the entering variable. As θ^* and $\boldsymbol{d}$ are known, the new basic feasible solution $\boldsymbol{x_B}$ is readily calculated from the previous solution. Go to Step 2 for starting a new RSA iteration.

In our example:

$$\boldsymbol{B} = \begin{pmatrix} 0 & 5 & 0 \\ 0 & 1 & 1 \\ 4 & -1 & 0 \end{pmatrix} \qquad \boldsymbol{x_B} = \begin{pmatrix} x_4 \\ x_3 \\ x_5 \end{pmatrix} = \begin{pmatrix} 5.5 \\ 12 \\ 0 \end{pmatrix}$$

The non-basic variables x_1, x_2, and x_6 have value zero and $z = 329$. Step 2 in the next RSA iteration would obtain dual variables $\boldsymbol{\pi} = (\ 3.9\ 0\ 9.5\ )$. Step 3 would calculate the following reduced costs: $\bar{c}_1 = 1.1$, $\bar{c}_2 = 3.9$, and $\bar{c}_6 = 9.5$. So, the current solution is optimal. By the way, that optimal solution is degenerate.

Why Dantzig decided to introduce the RSA only a few years after the original simplex algorithm? *He realized that n is significantly larger than m in most LPs.* Note that in order to put an LP into standard format, each inequality is first trans-

formed into an equality by introducing additional slack/surplus variables. Artificial variables may further increase the value of n.

The RSA performs much better than Dantzig's original simplex algorithm when n is much larger than m. Each iteration of the original simplex algorithm executes an expensive pivot operation that has a complexity that depends both on n and m. On the other hand, the only step in a RSA iteration that has a complexity that depends on n is Step 3 (pricing). The pricing step is a straightforward evaluation of $n - m$ linear expressions. Actually, it is not even necessary to price all non-basic variables in every iteration, this is only mandatory in the last iteration, in order to make sure that the final basic feasible solution is indeed optimal. The most expensive steps in RSA are usually steps 2 and 4, both of them have a complexity that only depends on m, since they require solving a $m \times m$ linear system.

As a consequence of that analysis, we arrive at a fundamental insight, that will be explored in the next chapter:

The RSA makes it possible to solve LPs with not so many constraints but with an astronomically huge number of variables, as long as those variables have a special structure that allows their efficient pricing. Instead of calculating reduced costs for each individual variable (an impossible task), the whole pricing step should be efficiently solved as another optimization problem!

Notes

1.1. The origins of LP. The concept of linear programming was independently devised by Soviet Leonid V. Kantorovich in 1939 and a few years later by Dutch-American Tjalling Koopmans, in the context of models for the optimum use of resources. They shared the 1975 Nobel Memorial Prize in Economic Sciences for those contributions. Some information about Kantorovich's Method of Resolving Multipliers and the problematic early years of linear pro-

gramming in the USSR is given in Note 4.9.

However, the simplex algorithm, the first fully developed and widely used method for solving LPs was created in 1947 by George B. Dantzig (he received many honors for that, but many people, including the authors of this book, think that he should also have been one of the recipients of the 1975 Nobel Prize). The early publications on the simplex algorithm are only abstracts [Dantzig, 1948] or internal reports that are now difficult to consult. The proof of the fundamental Theorem 1.1 can be found in Dantzig [1951] (reprinted in Cottle [2003]). The RSA was proposed in Dantzig [1953].

1.2. LP duality. Consider a primal LP in the following general format:

$$\min z = \boldsymbol{cx} \tag{1.4a}$$
$$\text{s.t.} \quad \boldsymbol{Ax} = \boldsymbol{b} \tag{1.4b}$$
$$\boldsymbol{Dx} \geq \boldsymbol{d} \tag{1.4c}$$
$$\boldsymbol{Fx} \leq \boldsymbol{f} \tag{1.4d}$$
$$\boldsymbol{x} \geq \boldsymbol{0}, \tag{1.4e}$$

where $\boldsymbol{A}$, $\boldsymbol{D}$, and $\boldsymbol{F}$ have dimensions $m \times n$, $r \times n$, and $s \times n$ respectively; the other vectors have compatible dimensions. Its dual LP is:

$$\max w = \boldsymbol{\pi b} + \boldsymbol{\rho d} + \boldsymbol{\theta f} \tag{1.5a}$$
$$\text{s.t.} \quad \boldsymbol{\pi A} + \boldsymbol{\rho D} + \boldsymbol{\theta F} \leq \boldsymbol{c} \tag{1.5b}$$
$$\boldsymbol{\rho} \geq \boldsymbol{0} \tag{1.5c}$$
$$\boldsymbol{\theta} \leq \boldsymbol{0}, \tag{1.5d}$$

where the dual variable vectors $\boldsymbol{\pi}$, $\boldsymbol{\rho}$, and $\boldsymbol{\theta}$ are associated with the constraints (1.4b), (1.4c), and (1.4d) of the primal LP respectively. For equality constraints, the dual variables have no sign restrictions. In a minimization LP, the $\geq$ constraints result in non-negative dual variables, while the $\leq$ constraints lead to non-positive dual variables. We state the three fundamental results on LP duality (proofs can be found in any standard linear programming textbook):

Theorem 1.2: Weak duality. *If $\boldsymbol{x}'$ is a feasible solution to the primal LP (1.4) and $(\boldsymbol{\pi}', \boldsymbol{\rho}', \boldsymbol{\theta}')$ is a feasible solution to the dual LP (1.5), then $z' = \boldsymbol{c}\boldsymbol{x}' \geq w' = \boldsymbol{\pi}'\boldsymbol{b} + \boldsymbol{\rho}'\boldsymbol{d} + \boldsymbol{\theta}'\boldsymbol{f}$.*

The above result implies that if the primal problem is unbounded, then its dual is infeasible, and vice-versa. However, both primal and dual LPs can be infeasible.

Theorem 1.3: Strong duality. *The primal LP (1.4) has optimal solutions if and only if the dual LP (1.5) has optimal solutions. Moreover, in such a case the optimal primal objective function value z^* is equal to the optimal dual objective function value w^*.*

All currently used LP-solving algorithms furnish both optimal primal and dual solutions. In the RSA, an optimal dual solution is obtained in Step 2 of the last iteration.

Theorem 1.4: Complementary Slackness. *Let $\boldsymbol{x}'$ be a feasible solution of primal LP (1.4) and $(\boldsymbol{\pi}', \boldsymbol{\rho}', \boldsymbol{\theta}')$ be a feasible solution of dual LP (1.5). Necessary and sufficient conditions for their simultaneous optimality are:*

$$\textit{(1)}\ (\boldsymbol{c} - \boldsymbol{\pi}'\boldsymbol{A} - \boldsymbol{\rho}'\boldsymbol{D} - \boldsymbol{\theta}'\boldsymbol{F})^\intercal \odot \boldsymbol{x}' = \boldsymbol{0};\ \textit{and,}$$

$$\textit{(2)}\ (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}')^\intercal \odot \boldsymbol{\pi}' = \boldsymbol{0},\ (\boldsymbol{d} - \boldsymbol{D}\boldsymbol{x}')^\intercal \odot \boldsymbol{\rho}' = \boldsymbol{0},\ (\boldsymbol{f} - \boldsymbol{F}\boldsymbol{x}')^\intercal \odot \boldsymbol{\theta}' = \boldsymbol{0},$$

where symbol $\odot$ denotes the Hadamard product (a.k.a. element-wise matrix/ vector product).

Condition (1) can be less formally stated as: each primal variable should have value zero or (non exclusive or) have reduced cost zero. Condition (2) can be stated as: each dual variable should have value zero or its corresponding primal constraint should be tight. Note that *the reduced cost of a variable is the slack of its corresponding dual constraint.*

1.3. Interpretation of dual variables and pricing. The value of a dual variable

indicates the rate at which the optimal objective function value would change with a sufficiently small change in the RHS of the corresponding constraint (a simple proof can be found on page 156 of Bertsimas and Tsitsiklis [1997]). The economic interpretation of those rates as *prices*, sometimes known as marginal prices or shadow prices, is related to some prototypical LP applications (like the one illustrated in Exercise E 1.1) where the goal is maximizing profits subject to resource availability. In those contexts, each dual variable indeed corresponds to the maximum price that one should be willing to pay for an extra unit of the corresponding resource, or equivalently, to the minimum price that one should ask for selling one unit of that resource.

The use of the name *pricing* to denote reduced cost evaluation also started in the early days of linear programming. One of the meanings of the word pricing found in dictionaries is indeed *evaluating*. However, in the LP context, the name is even more apt, as one is *using dual prices for evaluating variables*. For example, in the context of Exercise E 1.1, let variable x_j be associated with some new product j and let column $\boldsymbol{a}_j$ indicate how many units of each resource are required for manufacturing one unit of j. Therefore, $\overline{c}_j = c_j - \boldsymbol{\pi}\boldsymbol{a}_j$ expresses the net effect of manufacturing one unit of that product, which would sell for c_j but would reduce the profits obtainable from manufacturing the other products by $\boldsymbol{\pi}\boldsymbol{a}_j$. If $\overline{c}_j > 0$ then manufacturing product j can lead to an increased profit.

1.4. Modern simplex implementations. The original simplex algorithm only survives as a teaching device in textbooks. The modern competitive implementations of the RSA are far more sophisticated than the essential scheme presented in this chapter. In particular, there are several techniques for taking advantage of the fact that the linear systems that have to be solved in a certain iteration are very similar to the linear systems already solved in the previous iterations. Moreover, there are advanced techniques for choosing the entering variable in each iteration. Merely choosing the variable with the most negative reduced cost (the so-called *Dantzig's rule*) is considered to be quite naive. Many modern codes use *steepest-edge* strategies that try to estimate (exact evaluation being too costly), for each variable with negative reduced cost, the angle between the cost vector and the actual direction of change that

would be obtained in RSA Step 4 [Forrest and Goldfarb, 1992].

1.5. Choosing costs for artificial variables, Two-phase method. How large should artificial variables "large costs" should be in order to make sure that an optimal solution with positive value for some artificial variable indeed indicates that the original LP is infeasible? This can be a complex matter, see Section 4.4 in Bazaraa et al. [2010]. In practice, one can use a value M that is several orders of magnitude larger (say, 10^6 times larger) than the expected optimal solution value. If there is some reason to believe that this had not be enough, it is possible to increase the value M (by some orders of magnitude) only for the artificial variables with positive value in the optimal solution and check whether that solution remains optimal. The potential difficulty with that approach is that if M becomes too many orders of magnitude larger, numerical issues (related to the fact that computers store numbers with a finite precision) may appear.

An alternative simplex initialization that is not subject to numerical issues is the *two-phase method*. The Phase One LP (always set as a minimization problem) ignores the original costs, attributing unitary costs to all artificial variables and zero to the remaining variables. If the optimal solution value of that LP is larger than zero, it is proved that the original LP is infeasible. Otherwise, a feasible basis is found. Phase Two LP optimizes the original objective function starting from that basis.

1.6. LPs with variable bounds. Consider an LP in format $\min z = \boldsymbol{cx}$ subject to $\boldsymbol{Ax} = \boldsymbol{b}$, $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$, where $\boldsymbol{A}$ has dimension $m \times n$. A slightly generalized RSA (described in several LP textbooks, like Chvátal [1983]) can treat those bounds on individual variables in a special way, so they do not make the algorithm slower. In particular, bases still have dimension $m \times m$.

1.7. Primal simplex, dual simplex, hot-start. A *primal simplex* algorithm, like the RSA presented in this chapter, is characterized by the fact that the intermediate primal solutions obtained along its iterations are indeed feasible for the complete LP. On the other hand, the intermediate dual solutions,

obtained by only considering the current basic variables, are not feasible for the complete LP. Truly, the variables with negative reduced cost obtained in the pricing correspond to violated dual constraints. Only in the last iteration, where no variables with negative cost exist, both primal and dual solutions are feasible, and therefore, optimal. A *dual simplex* algorithm operates in the opposite way, intermediate solutions are dual feasible but only the last one is also primal feasible.

A primal simplex algorithm is convenient for re-solving an LP after new variables are added since the previous optimal solution is still primal feasible and can provide a *hot-start* basis. On the other hand, a dual simplex algorithm is suited for the fast re-solve of an LP after new constraints are added, since the previous optimal solution is still dual feasible.

1.8. Simplex convergence. The *convergence* of the simplex algorithm is related to the number of iterations until an optimal solution is found. In the worst case, the simplex algorithm may visit every basic feasible solution, requiring exponentially many iterations. Happily, this only happens on artificially constructed instances, like those in Klee and Minty [1972]. The practical convergence of the simplex algorithm is quite good. The typical number of iterations grows linearly with m, but much more slowly with n [Shamir, 1987]. That characteristic is essential for making the RSA a practical tool for solving the LPs with small m but huge n arising from column generation. Anyway, convergence can still be a serious issue in column generation. The topic is covered in depth in Chapter 7.

Exercises

E 1.1. A factory currently manufactures two different products, namely A and B. The production is limited by the availability of two resources: R1 and R2. Each unit of product A, sold at a profit of \$15, requires 3 units of R1 and 2 units of R2. Each unit of B, sold at a profit of \$9, requires 1 unit of R1 and 2 units of R2. The factory has a weekly supply of 600 units of R1

and 900 units of R2. So, the current optimal production plan is to make 75 units of A and 375 units of B, obtaining a total weakly profit of $4500. The management is considering the possibility of also manufacturing two new products, namely C and D. Each unit of C would be sold at a profit of $11 and would require 2 units of R1 and 1 unit of R2. Each unit of D would be sold at a profit of $5 and would require 1 unit of R1 and 1 unit of R2. Determine whether any of those two new products could lead to an increased total profit. If so, find a new improved production plan using the RSA.

Chapter 2

Dantzig-Wolfe Decomposition and Column Generation for Linear Programming

The Dantzig-Wolfe decomposition provides a technique for reformulating an LP and solving it by an iterative process that alternates between solving a Restricted Master LP and solving a single or multiple subproblems. As the subproblems generate new variables that are added to the Restricted Master LP, the technique became known as Column Generation. The overall goal is exploiting some particular matrix structures that may make those subproblems substantially smaller and/or easier to solve than the original LP.

2.1. Dantzig-Wolfe Reformulation for LP

Definition 2.1: Polyhedron, bounded polyhedron. A *polyhedron* is the set of points in the intersection of the half-spaces defined by a finite number of linear inequalities. So, a polyhedron P in an n-dimensional space can be defined as $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{D}\boldsymbol{x} \geq \boldsymbol{d}\}$ for some matrix $\boldsymbol{D}$ and vector $\boldsymbol{d}$. As equality can be seen as the intersection of two inequalities, the definition of a polyhedron may also include linear equalities. For every LP, its set of constraints (or any subset of those constraints) defines a polyhedron. A polyhedron P is *bounded* if there is a finite bound to the distance from the origin $\mathbf{0}$ to any point $\boldsymbol{x} \in P$.

Definition 2.2: Convex combination, convex hull. Given a finite set $X = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_t\}$ of points in an n-dimensional space, $\boldsymbol{x}$ is a *convex combination* of the

points in X if $\boldsymbol{x} = \sum_{j\in[t]} \boldsymbol{p}_j \lambda_j$ for some value of $\boldsymbol{\lambda} \in \mathbb{R}^t_+$ such that $\sum_{j\in[t]} \lambda_j = 1$. The *convex hull* of a set $X \in \mathbb{R}^n$ (finite or not), denoted by $Conv(X)$, is the set formed by all points obtainable as a convex combination of some finite subset of points in X.

Definition 2.3: Extreme point. A point in a polyhedron P is *extreme* if it is not a convex combination of other points in P. The set of the extreme points of P is denoted as $Ext(P)$.

Some elementary results in polyhedral theory are stated without proof.

Theorem 2.1: *For any finite set $X \in \mathbb{R}^n$, $Conv(X)$ is a bounded polyhedron.*

Theorem 2.2: *For any polyhedron P, $Ext(P)$ is finite.*

Theorem 2.3: *Let P be the polyhedron defined by the set of constraints of an LP in standard format. Every basic feasible solution of that LP corresponds to an extreme point of P.*

Theorem 2.4: *If P is a bounded polyhedron, $P = Conv(Ext(P))$. In other words, $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} = \sum_{\boldsymbol{q}\in Q} \boldsymbol{q}\lambda_{\boldsymbol{q}}, \sum_{\boldsymbol{q}\in Q} \lambda_{\boldsymbol{q}} = 1, \boldsymbol{\lambda} \geq \boldsymbol{0}\}$, where $Q = Ext(P)$ (note that we are using the vector $\boldsymbol{q} \in Q$ itself as the index of its corresponding λ variable).*

We now can introduce the Dantzig-Wolfe reformulation. Consider an LP in the following format:

$$\min z = \boldsymbol{cx} \tag{2.1a}$$

$$\text{s.t.} \quad \boldsymbol{Ax} = \boldsymbol{b} \tag{2.1b}$$

$$\boldsymbol{x} \in P, \tag{2.1c}$$

where $\boldsymbol{A}$ has dimension $m \times n$, and P is a *bounded* polyhedron represented by a set of linear constraints. We can replace (2.1c) by the requirement that $\boldsymbol{x}$ is a convex combination of the points in $Q = Ext(P)$. This can be done by including additional

variables λ in the LP and rewriting it as:

$$\min z = \boldsymbol{cx} \tag{2.2a}$$

$$\text{s.t.} \quad \boldsymbol{Ax} = \boldsymbol{b} \tag{2.2b}$$

$$\boldsymbol{x} = \sum_{\boldsymbol{q} \in Q} \boldsymbol{q}\ \lambda_{\boldsymbol{q}} \tag{2.2c}$$

$$\sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} = 1 \tag{2.2d}$$

$$\boldsymbol{\lambda} \geq \boldsymbol{0}. \tag{2.2e}$$

That LP, having both x and λ variables, is known as the *explicit reformulated LP* or *explicit Master LP*. Finally, we can eliminate the original variables x, by substituting their occurrences in (2.2a) and (2.2b), using (2.2c), obtaining the following *Dantzig-Wolfe reformulated LP* (a.k.a. *Master LP*):

$$z_{\text{M}} = \min \quad \sum_{\boldsymbol{q} \in Q} (\boldsymbol{cq}) \lambda_{\boldsymbol{q}} \tag{2.3a}$$

$$\text{s.t.} \quad \sum_{\boldsymbol{q} \in Q} (\boldsymbol{Aq}) \lambda_{\boldsymbol{q}} = \boldsymbol{b} \tag{2.3b}$$

$$\sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} = 1 \tag{2.3c}$$

$$\boldsymbol{\lambda} \geq \boldsymbol{0}. \tag{2.3d}$$

In the objective function (2.3a), the scalar number $\boldsymbol{cq}$ is the cost of the extreme point $\boldsymbol{q} \in Ext(P)$. In Constraints (2.3b), the $m \times 1$ vector $\boldsymbol{Aq}$ is the linear transformation of $\boldsymbol{q}$ over $\boldsymbol{A}$. Note that it is possible that $\boldsymbol{Aq} \neq \boldsymbol{b}$, for all $\boldsymbol{q} \in Q$; meaning that no point in $Ext(P)$ satisfies (2.1b). However, Constraints (2.3b) state that the linear combination of those vectors given by $\boldsymbol{\lambda}$ should be equal to $\boldsymbol{b}$. Equation (2.3c) is known as the *convexity constraint* and make sure that the linear combination is also a convex combination.

Consider the following example:

$$\begin{array}{lrcrcl} \min\ z = & -5x_1 & + & 3x_2 & & \\ \text{s.t.} & x_1 & + & 2x_2 & = & 6 \\ & x_1 & + & x_2 & \leq & 4 \\ & 2x_1 & & & \leq & 7 \\ & & & \boldsymbol{x} & \geq & \mathbf{0}. \end{array} \tag{2.4}$$

Let $\boldsymbol{Ax} = \boldsymbol{b}$ be defined by the first equation and P by the remaining inequalities (including the variable non-negativities). Bounded polyhedron P has four extreme points, $Ext(P) = \{(\,0\ 0\,), (\,0\ 4\,), (\,3.5\ 0.5\,), (\,3.5\ 0\,)\}$. The explicit reformulated LP is:

$$\begin{array}{lrcrcrcrcl} \min z = & -5x_1 & + 3x_2 & & & & & & & \\ \text{s.t.} & x_1 & + 2x_2 & & & & & & & = 6 \\ & x_1 & & & & & - 3.5\lambda_{(3.5\ 0.5)} & - 3.5\lambda_{(3.5\ 0)} & & = 0 \\ & & x_2 & & - 4\lambda_{(0\ 4)} & & - 0.5\lambda_{(3.5\ 0.5)} & & & = 0 \\ & & & \lambda_{(0\ 0)} & + \lambda_{(0\ 4)} & + & \lambda_{(3.5\ 0.5)} & + \lambda_{(3.5\ 0)} & & = 1 \\ & & & & & & & & \boldsymbol{\lambda} & \geq \mathbf{0}. \end{array}$$

Eliminating the x variables, the resulting Dantzig-Wolfe reformulation is:

$$\begin{array}{lrcrcrcrcl} \min z = & & & 12\lambda_{(0\ 4)} & - & 16\lambda_{(3.5\ 0.5)} & - & 17.5\lambda_{(3.5\ 0)} & & \\ \text{s.t.} & & & 8\lambda_{(0\ 4)} & + & 4.5\lambda_{(3.5\ 0.5)} & + & 3.5\lambda_{(3.5\ 0)} & = & 6 \\ & \lambda_{(0\ 0)} & + & \lambda_{(0\ 4)} & + & \lambda_{(3.5\ 0.5)} & + & \lambda_{(3.5\ 0)} & = & 1 \\ & & & & & & & \boldsymbol{\lambda} & \geq & \mathbf{0}. \end{array} \tag{2.5}$$

Its optimal solution is $\lambda_{(0\,0)} = 0$, $\lambda_{(0\,4)} = 3/7$, $\lambda_{(3.5\,0.5)} = 4/7$, and $\lambda_{(3.5\,0)} = 0$, and with $z = -4$. The optimal solution to the original LP can be recovered using (2.2c), so $\boldsymbol{x} = (\,x_1\ x_2\,) = 3/7\,(\,0\ 4\,) + 4/7\,(\,3.5\ 0.5\,) = (\,2\ 2\,)$.

As that original LP has two variables, the reformulation procedure can be depicted graphically in a cartesian plane. Figure 2.1 shows in light blue the set P. The set of LP solutions (the line segment between $(\,0\ 3\,)$ and $(\,2\ 2\,)$) is marked in black. It can be seen that point $(\,0\ 0\,)$ belongs to the line $x_1 + 2x_2 = 0$, $(\,0\ 4\,)$ belongs to the line $x_1 + 2x_2 = 8$, $(\,3.5\ 0.5\,)$ belongs to the line $x_1 + 2x_2 = 4.5$, and $(\,3.5\ 0\,)$ belongs to the line $x_1 + 2x_2 = 3.5$. Therefore, a convex combination of those four points belongs to line $x_1 + 2x_2 = 6$ if and only if $0\lambda_{(0\,0)} + 8\lambda_{(0\,4)} + 4.5\lambda_{(3.5\,0.5)} + 3.5\lambda_{(3.5\,0)} = 6$.

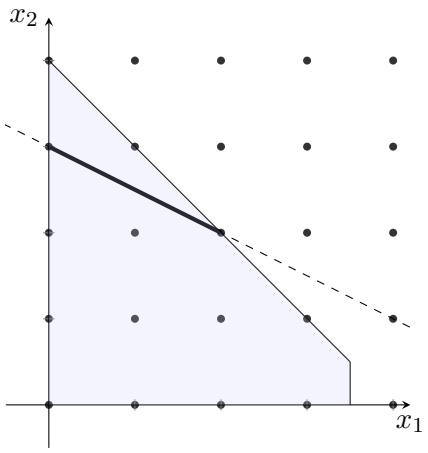


Figure 2.1: DW reformulation of the LP (2.4)

Consider this second example:

$$\begin{array}{lrcrcrcl} \min z = & 8x_1 & + & 13x_2 & - & 5x_3 & & \\ \text{s.t.} & 4x_1 & + & x_2 & + & 2x_3 & = & 5 \\ & x_1 & + & x_2 & & & = & 1 \\ & -2x_1 & + & 3x_2 & + & 3x_3 & \leq & 3 \\ & 3x_1 & - & x_2 & + & 6x_3 & \leq & 6 \\ & & & & & \boldsymbol{x} & \geq & \mathbf{0}. \end{array}$$

Let $\boldsymbol{Ax} = \boldsymbol{b}$ be defined by the first two equations and P by the remaining inequalities (including the variable non-negativities). Bounded polyhedron P has five extreme points, $Ext(P) = \{(0\ 0\ 0), (2\ 0\ 0), (0\ 1\ 0), (3\ 3\ 0), (0\ 0\ 1)\}$. The resulting Dantzig-Wolfe reformulation is:

$$\begin{array}{lrcrcrcrcl} \min z = & & & 16\lambda_{(200)} & + & 13\lambda_{(010)} & + & 63\lambda_{(330)} & - & 5\lambda_{(001)} & & \\ \text{s.t.} & & & 8\lambda_{(200)} & + & \lambda_{(010)} & + & 15\lambda_{(330)} & + & 2\lambda_{(001)} & = & 5 \\ & & & 2\lambda_{(200)} & + & \lambda_{(010)} & + & 6\lambda_{(330)} & & & = & 1 \\ & \lambda_{(000)} & + & \lambda_{(200)} & + & \lambda_{(010)} & + & \lambda_{(330)} & + & \lambda_{(001)} & = & 1 \\ & & & & & & & & & \boldsymbol{\lambda} & \geq & \mathbf{0}. \end{array} \tag{2.6}$$

Its optimal solution is $\lambda_{(200)} = \lambda_{(001)} = 0.5$, and $\lambda_{(000)} = \lambda_{(010)} = \lambda_{(330)} = 0$, with $z = 5.5$. The optimal solution to the original LP can be recovered using (2.2c), so $\boldsymbol{x} = (x_1\ x_2\ x_3) = 0.5\,(2\ 0\ 0) + 0.5\,(0\ 0\ 1) = (1\ 0\ 0.5)$.

2.2. Solving the Reformulated LP

2.2.1. Solving using the Revised Simplex Algorithm

A reformulated LP (2.3) has $m+1$ equalities, but $|Ext(P)|$ variables, which can be a huge number even if the original problem (2.1) has a moderate size. Indeed, except for small-sized examples, it is practically impossible to even compute (2.3) explicitly, let alone solve it directly. So, Dantzig and Wolfe had to propose a special algorithm for solving their reformulated LPs. That algorithm was the RSA, presented in the previous chapter, with the following modifications:

- **Modified RSA Step 1:** A basis $\boldsymbol{B}$ for (2.3) has dimension $(m+1) \times (m+1)$. Since the variables in that LP are not explicitly known, it may be more difficult to find an initial basis $\boldsymbol{B}$ with actual columns from the problem. However, the step can always be performed by defining $\boldsymbol{B}$ as an identity matrix corresponding to $m+1$ artificial variables with very large costs.

- **Modified RSA Step 3:** Let $(\boldsymbol{\pi}^*, \nu^*)$ be the optimal dual solution calculated in Step 2, where $\boldsymbol{\pi} = (\pi_1 \ \ldots \ \pi_m)$ corresponds to the Constraints (2.3b) and ν to the Constraint (2.3c). The pricing is done by applying the simplex algorithm to the following subproblem LP:

$$\bar{c}^* = \min \quad (\boldsymbol{c} - \boldsymbol{\pi}^* \boldsymbol{A})\, \boldsymbol{x} - \nu^* \tag{2.7a}$$

$$\text{s.t.} \qquad \boldsymbol{x} \in P. \tag{2.7b}$$

  The use of the symbol * helps to remind us that $\boldsymbol{\pi}^*$ and ν^* are constants in (2.7a), and only $\boldsymbol{x}$ is a vector of variables. If the subproblem is infeasible, i.e., $P = \emptyset$ and $\bar{c}^* = \infty$, then *the algorithm stops (in its first iteration), since the original problem should be infeasible too.* Otherwise, as P is bounded, an optimal solution $\boldsymbol{x}^*$ will be found. By Theorem 2.3, $\boldsymbol{x}^* \in Ext(P)$. If $\bar{c}^* < 0$ then the variable $\lambda_{\boldsymbol{x}^*}$ having cost $\boldsymbol{c}\boldsymbol{x}^*$ and corresponding to the column $\begin{pmatrix} \boldsymbol{A}\boldsymbol{x}^* \\ 1 \end{pmatrix}$ should be chosen as the entering variable. If $\bar{c}^* \geq 0$, then *the algorithm stops and the current basic feasible solution is optimal.* However, if the current solution still contains some artificial variables with a positive value, this actually means that *the original LP is infeasible.*

Theorem 2.5: *The Modified RSA solves* (2.3).

Proof. It only has to be shown that Modified Step 3 correctly performs the pricing. Consider variable $\lambda_{\boldsymbol{q}}$, $\boldsymbol{q} \in Q$. Its reduced cost at a given iteration is given by:

$$\overline{c}_{\boldsymbol{q}} = \boldsymbol{c}\boldsymbol{q} - \begin{pmatrix} \boldsymbol{\pi}^* & \nu^* \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{A}\boldsymbol{q} \\ 1 \end{pmatrix} = \boldsymbol{c}\boldsymbol{q} - \boldsymbol{\pi}^*\boldsymbol{A}\boldsymbol{q} - \nu^* = (\boldsymbol{c} - \boldsymbol{\pi}^*\boldsymbol{A})\boldsymbol{q} - \nu^*.$$

Therefore, $\overline{c}^* = \min\{\overline{c}_{\boldsymbol{q}} : \boldsymbol{q} \in Q\}$. Moreover, the point $\boldsymbol{x}^* \in Ext(P)$ generates a variable (and the corresponding column) with reduced cost equal to $\overline{c}^*$. If $\overline{c}^* < 0$, that variable $\lambda_{\boldsymbol{x}^*}$ is indeed eligible for entering in the basis. If $\overline{c}^* \geq 0$, it is proven that no variable has a negative reduced cost. □

Note that the subproblem LP (2.7) finds the minimum reduced cost of all non-artificial variables, basic and non-basic ones. Basic variables have zero reduced cost. So, unless the Modified RSA stops in the first iteration by infeasibility or all basic variables are artificial, it will stop with $\overline{c}^* = 0$.

There is an important practical observation. Using the *generating vector* $\boldsymbol{x}^*$ itself as the index of the *generated variable* $\lambda_{\boldsymbol{x}^*}$ is conceptually correct and algebraically convenient. However, it is not so convenient to implement the algorithm. This happens because those vectors can be quite long and may contain fractional elements. Therefore, it is much more common to number the λ variables following their order of generation. So, the first generated variable would be λ_1, the second λ_2, and so on. Of course, there must be a side table linking the numerical index of each generated variable to its corresponding generating point in $Ext(P)$. That table will be needed in order to recover the solution to the original LP (2.1) using (2.2c).

2.2.2. Solving using Restricted Master LPs: the Column Generation Algorithm

Solving the reformulated problem using the Modified RSA is a "low-level approach". In order to do that, one would have to implement efficiently all RSA steps, which is a complex task. Happily, assuming that an LP solver is available (a very reasonable assumption, considering that Modified Step 3 already requires an LP solver), a simpler higher-level algorithm can be used.

Definition 2.4: Master LP, Restricted Master LP. The reformulated LP (2.3) is called the *Master LP* (MLP). A *Restricted Master LP* (RMLP) is the restriction to (2.3) obtained by only keeping the variables corresponding to the points in a subset S of $Ext(P)$. An RMLP is:

$$z_{\text{RM}} = \min \quad \sum_{\boldsymbol{q}\in S} (\boldsymbol{cq})\lambda_{\boldsymbol{q}} \tag{2.8a}$$

$$\text{s.t.} \quad \sum_{\boldsymbol{q}\in S} (\boldsymbol{Aq})\lambda_{\boldsymbol{q}} = \boldsymbol{b} \tag{2.8b}$$

$$\sum_{\boldsymbol{q}\in S} \lambda_{\boldsymbol{q}} = 1 \tag{2.8c}$$

$$\boldsymbol{\lambda} \geq \boldsymbol{0}. \tag{2.8d}$$

Usually, $|S| \ll |Ext(P)|$. The RMLP may also contain additional artificial variables with very large costs.

The following algorithm for solving the MLP will be called the *Column Generation Algorithm* (CGA):

CGA Step 1: Set an initial feasible RMLP.

Sometimes it is possible to find a suitably small set of variables from the MLP for that purpose. However, in many cases, it is more practical to introduce artificial variables. Alternatively, one can start with an empty RMLP and use the Farkas pricing (see Note 2.5) while the current RMLP is infeasible.

CGA Step 2: Solve the current RMLP.

Solve the RMLP and let $(\boldsymbol{\pi}^*, \nu^*)$ be the obtained optimal dual solution.

CGA Step 3 (Pricing): Solve the Subproblem LP.

Solve the LP (2.7).

CGA Step 4: Either stop or update RMLP and go to Step 2.

Let $\boldsymbol{x}^*$ be the optimal subproblem solution found (if the subproblem is infeasible the algorithm stops in the first iteration and *the original LP is infeasible*). If $\overline{c}^* < 0$ then a new variable $\lambda_{\boldsymbol{x}^*}$ (usually notated as λ_j, where j is its sequential generation number) should be added to the RMLP. That variable has cost $\boldsymbol{cx}^*$ and the corresponding column is $\begin{pmatrix} \boldsymbol{Ax}^* \\ 1 \end{pmatrix}$. The CGA should then go to Step 2 for starting a new iteration. However, if $\overline{c}^* \geq 0$, then *the algorithm stops*. In that case, if the RMLP

primal solution contains some artificial variables with positive value, *the original LP is infeasible*. Otherwise, *the current RMLP provides an optimal solution to the MLP*. Let $\boldsymbol{\lambda}^* \in \mathbb{R}^{|S|}$ be the optimal primal solution of the RMLP. The optimal solution to the original problem is:

$$\boldsymbol{x} = \sum_{\boldsymbol{q} \in S} \lambda^*_{\boldsymbol{q}} \boldsymbol{q}. \tag{2.9}$$

Theorem 2.6: *At any iteration of the CGA for solving* (2.3) *(even if the RMLP contains artificial variables), the value* $z_{RM} + \overline{c}^*$ *is a lower bound on its optimal cost* z_M.

Proof. The dual of the MLP (2.3) is:

$$\begin{aligned} \max \quad & \boldsymbol{\pi}\boldsymbol{b} + \nu && && \text{(2.10a)} \\ \text{s.t.} \quad & \boldsymbol{\pi}\boldsymbol{A}\boldsymbol{q} + \nu \leq \boldsymbol{c}\boldsymbol{q} && \boldsymbol{q} \in Q. && \text{(2.10b)} \end{aligned}$$

Let $(\boldsymbol{\pi}^*, \nu^*)$ be an optimal dual solution with value $z_{\text{RM}} = \boldsymbol{\pi}^*\boldsymbol{b} + \nu^*$ of some RMLP and let $\overline{c}^* = \min\{\boldsymbol{c}\boldsymbol{q} - \boldsymbol{\pi}^*\boldsymbol{A}\boldsymbol{q} - \nu^* : \boldsymbol{q} \in Q\}$ be the optimal solution value of the corresponding pricing subproblem. If $\overline{c}^* < 0$ then $(\boldsymbol{\pi}^*, \nu^*)$ is not feasible for (2.10) and $-\overline{c}^*$ is the maximum violation of a constraint in (2.10b). Therefore, $(\boldsymbol{\pi}^*, \nu^* + \overline{c}^*)$ is feasible for (2.10) and has value $z_{\text{RM}} + \overline{c}^*$. □

The above result may be very useful for estimating the optimal MLP value before the CGA fully converges. Moreover, it also proves that the CGA finishes correctly. Consider an RMLP with value z_{RM} and such that its dual solution leads to $\overline{c}^* \geq 0$. If the RMLP primal solution contains very costly artificial variables with a positive value, $z_{\text{RM}} + \overline{c}^*$ is a very large lower bound on z_{M}, showing that the MLP should be infeasible. Otherwise, z_{RM} should be an upper bound on z_{M}, so $\overline{c}^*$ should be zero and z_{RM} should be equal to z_{M}.

Let us illustrate the Column Generation Algorithm for solving the MLP (2.6).

The subproblem has the following format:

$$\overline{c}^* = \min \quad \left(\begin{pmatrix} 8 & 13 & -5 \end{pmatrix} - \boldsymbol{\pi}^* \begin{pmatrix} 4 & 1 & 2 \\ 1 & 1 & 0 \end{pmatrix} \right) \boldsymbol{x} - \nu^* \quad =$$

$$\begin{array}{rrrrrrl} & (8 - 4\pi_1^* - \pi_2^*)x_1 & & +(13 - \pi_1^* - \pi_2^*)x_2 & & +(-5 - 2\pi_1^*)x_3 & -\nu^* \\ \text{s.t.} & -\ 2\ x_1 & & +\ 3\ x_2 & & +\ 3\ x_3 & \leq\ 3 \\ & 3\ x_1 & & -\ \ x_2 & & +\ 6\ x_3 & \leq\ 6 \\ & x_1, & & x_2, & & x_3 & \geq\ 0. \end{array}$$

The constraints in that LP correspond to the polyhedron P. Suppose that one realizes that point $(\,0\ 0\ 0\,) \in Ext(P)$, so it can provide variable $\lambda_{(0\,0\,0)}$, more conveniently notated as λ_1. The first RMLP is initialized using two additional artificial variables, a_1 and a_2, both having "large cost" 99:

$$\begin{array}{rrrrrrrrrrr} z_{\text{RM}} = & \min & 99a_1 & + & 99a_2 & + & 0\lambda_1 & & & & \\ & \text{s.t.} & a_1 & & & & & & = & & 5 \\ & & & & & & a_2 & & = & & 1 \\ & & & & & & & \lambda_1 & = & & 1 \\ & & a_1 & , & & a_2 & , & \lambda_1 & \geq & & 0. \end{array}$$

By solving this first RMLP, one obtains $z_{\text{RM}} = 594$. The optimal dual solution is $\boldsymbol{\pi}^* = (\,99\ 99\,)$ and $\nu^* = 0$ (the primal solution of the RMLP has no use at this point). So, the first subproblem LP is:

$$\begin{array}{rl} \overline{c}^* = \min & -487x_1 - 185x_2 - 203x_3 \\ \text{s.t.} & \boldsymbol{x} \in P. \end{array}$$

Its optimal solution is $\boldsymbol{x}^* = (\,3\ 3\ 0\,)$, with $\overline{c}^* = -2016$. According to Theorem 2.6, $z_{\text{RM}} + \overline{c}^* = -1422$ is a valid lower bound on z_{M} (the bound is very poor because the artificial variables are still being used by the RMLP solution). The column

corresponding to generating point $\boldsymbol{x}^*$ is inserted into the RMLP, that becomes:

$$\begin{array}{rlrcrcrcrcl} z_{\mathrm{RM}} = & \min & 99a_1 & + & 99a_2 & + & 0\lambda_1 & + & 63\lambda_2 & & \\ & \text{s.t.} & a_1 & & & & & + & 15\lambda_2 & = & 5 \\ & & & & a_2 & & & + & 6\lambda_2 & = & 1 \\ & & & & & & \lambda_1 & + & \lambda_2 & = & 1 \\ & & a_1, & & a_2, & & \lambda_1, & & \lambda_2 & \geq & 0. \end{array}$$

By reoptimizing it, one gets $z_{\mathrm{RM}} = 258$, $\boldsymbol{\pi}^* = (\,99\ -237\,)$ and $\nu^* = 0$. The second subproblem LP is $\min -151x_1 + 151x_2 - 203x_3$, subject to $\boldsymbol{x} \in P$. Its solution yields $\boldsymbol{x}^* = (\,2\ 0\ 0\,)$, with $\overline{c}^* = -302$, and the obtained lower bound is $-44 \leq z_{\mathrm{M}}$. The corresponding column is inserted into the RMLP, which becomes:

$$\begin{array}{rlrcrcrcrcrcl} z_{\mathrm{RM}} = & \min & 99a_1 & + & 99a_2 & + & 0\lambda_1 & + & 63\lambda_2 & + & 16\lambda_3 & & \\ & \text{s.t.} & a_1 & & & & & + & 15\lambda_2 & + & 8\lambda_3 & = & 5 \\ & & & & a_2 & & & + & 6\lambda_2 & + & 2\lambda_3 & = & 1 \\ & & & & & & \lambda_1 & + & \lambda_2 & + & \lambda_3 & = & 1 \\ & & a_1, & & a_2, & & \lambda_1, & & \lambda_2, & & \lambda_3 & \geq & 0. \end{array}$$

By reoptimizing the RMLP, one gets $z_{\mathrm{RM}} = 107$, $\boldsymbol{\pi}^* = (\,99\ -388\,)$ and $\nu^* = 0$. The third subproblem LP is $\min 302x_2 - 203x_3$, subject to $\boldsymbol{x} \in P$. Its solution yields $\boldsymbol{x}^* = (\,0\ 0\ 1\,)$, with $\overline{c}^* = -203$, so the lower bound $-96 \leq z_{\mathrm{M}}$ at this iteration is worse than the bound obtained at the previous iteration. This is normal. While the sequence of upper bounds on z_{M} given by successive z_{RM} values is monotonically non-increasing, the sequence of lower bounds provided by Theorem 2.6 is *not* monotonically non-decreasing. The column corresponding to $\boldsymbol{x}^*$ is inserted into the RMLP:

$$\begin{array}{rlrcrcrcrcrcrcl} z_{\mathrm{RM}} = & \min & 99a_1 & + & 99a_2 & + & 0\lambda_1 & + & 63\lambda_2 & + & 16\lambda_3 & - & 5\lambda_4 & & \\ & \text{s.t.} & a_1 & & & & & + & 15\lambda_2 & + & 8\lambda_3 & + & 2\lambda_4 & = & 5 \\ & & & & a_2 & & & + & 6\lambda_2 & + & 2\lambda_3 & & & = & 1 \\ & & & & & & \lambda_1 & + & \lambda_2 & + & \lambda_3 & + & \lambda_4 & = & 1 \\ & & a_1, & & a_2, & & \lambda_1, & & \lambda_2, & & \lambda_3, & & \lambda_4 & \geq & 0. \end{array}$$

By reoptimizing the RMLP, one gets $z_{\mathrm{RM}} = 5.5$, $\boldsymbol{\pi}^* = (\,3.5\ 0\,)$ and $\nu^* = -12$ (there are alternative optimal dual solutions like $(\boldsymbol{\pi}^*, \nu^*) = (\,0\ 10.5\ -5\,)$ as well as $(\boldsymbol{\pi}^*, \nu^*) = (\,99\ -286.5\ -203)$). The fourth subproblem is $\min -6x_1 + 9.5x_2 - 12x_3 + 12$, subject to $\boldsymbol{x} \in P$. We have that $\overline{c}^* = 0$ (the LP actually solved has optimal

cost -12, the constant term corresponding to $-\nu^*$ is added later). This proves that $z_{\mathrm{RM}} = z_{\mathrm{M}} = 5.5$ and that the current RMLP contains an optimal MLP solution. Now, the primal RMLP solution $\lambda_1 = \lambda_2 = 0$, and $\lambda_3 = \lambda_4 = 0.5$ can be used for computing the optimal solution of the original LP (it is necessary to retrieve the points that generated λ_3 and λ_4 from a side table): $\boldsymbol{x} = 0.5\,(\,2\;0\;0\,) + 0.5\,(\,0\;0\;1\,) = (\,1\;0\;0.5\,)$.

In this small example, the final RMLP contains four out of the five columns in the MLP. Such a high proportion is very unlikely to be observed in larger problems. In fact, the Column Generation Algorithm is expected to be practical only because the final RMLP usually contains only a tiny fraction of the columns in MLP.

The CGA actually has a fairly large degree of freedom, as indicated in the following observations:

- If Constraints (2.1b) in the original LP happen to contain inequalities, it is not mandatory to convert them to equalities. In that case, the constraints in the MLP will have the same sense of the corresponding constraints in (2.1b).

- The subproblem LP may have a particular structure that makes possible the use of specialized algorithms that are faster than generic LP algorithms. For example, it may correspond to a *minimum cost network flow problem*, an especially favorable situation that will be covered in detail in Section 2.5. In other cases, the points in $Ext(P)$ correspond to the solutions of certain combinatorial optimization problems, for which highly efficient polynomial-time algorithms are known. For example, if the points in $Ext(P)$ correspond to spanning trees, the Prim–Jarník algorithm can solve the pricing.

- Non-optimal solutions to Subproblem (2.7a), as long as they yield columns with negative reduced cost, can be used in the pricing step. So, it is possible to use fast heuristics for the pricing, especially in the first iterations, where it is easier to find columns with negative reduced costs. Indeed, in order to speed up the CGA convergence (see Chapter 7), it is possible to add multiple columns in the same iteration. In fact, it is not even necessary that all the generating points belong to $Ext(P)$, other points in P may also be used for generating variables. Anyway, when heuristics fail, it is mandatory to solve the pricing to optimality for checking whether there are still columns with negative reduced cost.

- The re-optimization of the RMLPs after new columns are introduced is not performed from scratch. A *primal* simplex algorithm (Note 1.7) can be hot-started with the optimal basis of the previous RMLP. Truly, this happens automatically in most LP solver packages.

- If the RMLP starts to get big, it is possible to perform a *clean-up*, removing from it older columns that are not likely to be useful anymore. There are several clean-up criteria (also discussed in Chapter 7). Columns in the current basis, even if with value zero, should not be removed since that may interfere with the hot-start capability of the LP solver.

2.3. Multiple Subproblems

The most important structure that can be exploited with Dantzig-Wolfe reformulation is the case where the Subproblem LP is decomposable into multiple independent subproblems.

2.3.1. General Case

Consider an LP in the following format:

$$\min \; z = \; \boldsymbol{c}^1\boldsymbol{x}^1 + \cdots + \boldsymbol{c}^U\boldsymbol{x}^U \tag{2.11a}$$

$$\text{s.t.} \quad \boldsymbol{A}^1\boldsymbol{x}^1 + \cdots + \boldsymbol{A}^U\boldsymbol{x}^U = \boldsymbol{b} \tag{2.11b}$$

$$\boldsymbol{x}^u \in P^u \qquad\qquad u \in [U]. \tag{2.11c}$$

For each $u \in [U]$, $\boldsymbol{A}^u$ has dimension $m \times n^u$ and P^u is a bounded polyhedron defined by a set of linear constraints. Constraints (2.11b) are a generic set of m linear equations over $n = \sum_{u\in[U]} n^u$ variables. The special structure in this LP is that (2.11c) consists of U independent sets of constraints, each such set being expressed over a distinct subset of the variables. We can replace (2.11c) by the requirement that each $\boldsymbol{x}^u$ is a convex combination of the points in $Q^u = Ext(P^u)$. This can be done by including additional variables θ and writing the explicit master

LP as:

$$\min\ z = \quad \boldsymbol{c}^1\boldsymbol{x}^1 + \cdots + \boldsymbol{c}^U\boldsymbol{x}^U \tag{2.12a}$$

$$\text{s.t.} \quad \boldsymbol{A}^1\boldsymbol{x}^1 + \cdots + \boldsymbol{A}^U\boldsymbol{x}^U = \boldsymbol{b} \tag{2.12b}$$

$$\boldsymbol{x}^u = \sum_{\boldsymbol{q}\in Q^u} \boldsymbol{q}\theta^u_{\boldsymbol{q}} \qquad u \in [U] \tag{2.12c}$$

$$\sum_{\boldsymbol{q}\in Q^u} \theta^u_{\boldsymbol{q}} = 1 \qquad u \in [U] \tag{2.12d}$$

$$\boldsymbol{\theta} \geq \boldsymbol{0}. \tag{2.12e}$$

Finally, by eliminating the original variables x, substituting their occurrences in (2.12a) and (2.12b), using (2.12c), we obtain the Master LP:

$$z_{\text{M}} = \min \quad \sum_{u\in[U]} \sum_{\boldsymbol{q}\in Q^u} (\boldsymbol{c}^u\boldsymbol{q})\theta^u_{\boldsymbol{q}} \tag{2.13a}$$

$$\text{s.t.} \quad \sum_{u\in[U]} \sum_{\boldsymbol{q}\in Q^u} (\boldsymbol{A}^u\boldsymbol{q})\theta^u_{\boldsymbol{q}} = \boldsymbol{b} \tag{2.13b}$$

$$\sum_{\boldsymbol{q}\in Q^u} \theta^u_{\boldsymbol{q}} = 1 \qquad u \in [U] \tag{2.13c}$$

$$\boldsymbol{\theta} \geq \boldsymbol{0}. \tag{2.13d}$$

This MLP can be solved by a slightly generalized Column Generation Algorithm (the algorithm presented in the previous section corresponds to the case $U = 1$). Let a Restricted Master LP be any feasible LP obtained by only keeping a subset of the columns of the MLP and (possibly) by artificial variables with very large costs. Let $\boldsymbol{\pi} = (\pi_1 \ldots \pi_m)$ be the vector with the dual variables associated to (2.13b) and $\boldsymbol{\nu} = (\nu_1 \ldots \nu_U)$ be the vector with the dual variables associated to (2.13c). Let $(\boldsymbol{\pi}^*, \boldsymbol{\nu}^*)$ be the optimal RMLP dual solution obtained in some iteration. The main generalization is that the pricing step is performed by solving U subproblems. Subproblem u, $u \in [U]$, is defined as:

$$\overline{c}^{u*} = \min \quad (\boldsymbol{c}^u - \boldsymbol{\pi}^*\boldsymbol{A}^u)\boldsymbol{x}^u - \nu^*_u \tag{2.14a}$$

$$\text{s.t.} \quad \boldsymbol{x}^u \in P^u. \tag{2.14b}$$

Let $\boldsymbol{x}^{u*}$ be the optimal solution found (if the subproblem is infeasible the algorithm stops in the first iteration and *the original problem is infeasible*). If $\overline{c}^{u*} < 0$, then a

new variable $\theta^u_{\boldsymbol{x}^{u*}}$ (often notated as θ^u_j, where j is its sequential generation number in subproblem u) should be added to the RMLP. That variable has cost $\boldsymbol{c}^u\boldsymbol{x}^{u*}$ and the corresponding column is $\begin{pmatrix} \boldsymbol{A}^u\boldsymbol{x}^{u*} \\ \boldsymbol{e}_u \end{pmatrix}$, where $\boldsymbol{e}_u$ is a $U \times 1$ vector having 1 in row u and 0 in the other rows. If for all u, $u \in [U]$, $\overline{c}^{u*} \geq 0$ then *the algorithm stops*. In that case, if the RMLP primal solution contains some artificial variables with a positive value, *the original problem is infeasible*. Otherwise, *the current RMLP provides an optimal solution to the MLP*.

2.3.2. Identical Subproblems

In some cases, the Dantzig-Wolfe reformulation may lead to identical subproblems. Subproblems $i, j \in [U]$ are said to be identical if $\boldsymbol{c}^i = \boldsymbol{c}^j$, $\boldsymbol{A}^i = \boldsymbol{A}^j$, and $P^i = P^j$. At first sight, this may look like a very exceptional case. However, as will be shown in Chapter 4, there are many important applications even leading to the case where *all subproblems are identical*. Solving identical subproblems is wasteful. The columns that would be generated are essentially the same, differing only in the $\boldsymbol{e}_u$ part of the vector.

However, it is not necessary to do that. Identical subproblems can be aggregated. Consider that a maximal set of identical subproblems defines a group and that there are K such groups. Group $k \in [K]$ is composed by U^k subproblems. Assume w.l.o.g. that the columns in the original problem (2.11) are permutated in such a way that the first K subproblems are distinct and the remaining $U - K$ subproblems are identical to some of those first K subproblems. This means that for a group $k \in [K]$, $\boldsymbol{c}^k$, $\boldsymbol{A}^k$, and $Q^k = Ext(P^k)$ can be used for representing the cost vectors, submatrices of $\boldsymbol{A}$, and set of extreme points associated to any subproblem in that group. Consider the following Master LP:

$$z_{\mathrm{M}} = \min \quad \sum_{k\in[K]} \sum_{\boldsymbol{q}\in Q^k} (\boldsymbol{c}^k\boldsymbol{q})\lambda^k_{\boldsymbol{q}} \tag{2.15a}$$

$$\text{s.t.} \quad \sum_{k\in[K]} \sum_{\boldsymbol{q}\in Q^k} (\boldsymbol{A}^k\boldsymbol{q})\lambda^k_{\boldsymbol{q}} = \boldsymbol{b} \tag{2.15b}$$

$$\sum_{\boldsymbol{q}\in Q^k} \lambda^k_{\boldsymbol{q}} = U^k \qquad k \in [K] \tag{2.15c}$$

$$\boldsymbol{\lambda} \geq \boldsymbol{0}. \tag{2.15d}$$

This LP is obtained from (2.13) by defining aggregated variables $\lambda^k_{\boldsymbol{q}} = \sum_{u \in U(k)} \theta^u_{\boldsymbol{q}}$, $\boldsymbol{q} \in Q^k$, where $U(k) \subseteq [U]$ are the indices associated to group $k \in [K]$. Then, by variable substitution, (2.15a) is equivalent to (2.13a) and (2.15b) is equivalent to (2.13b). However, the k-th aggregated convexity constraint (2.15c) is obtained by *summing* U^k equations in (2.13c) (in general, replacing a set of equations by the single equation corresponding to their sum is a relaxation and may increase the set of feasible solutions of an LP). So, the following result is not obvious and should be proved:

Theorem 2.7: *An optimal solution of* (2.15) *yields an optimal solution of* (2.13).

Proof. Given an optimal solution $\boldsymbol{\lambda}^*$ of (2.15) with value z^*, we claim that any $\boldsymbol{\theta}$ such that

$$\sum_{u \in U(k)} \theta^u_{\boldsymbol{q}} = \lambda^{k*}_{\boldsymbol{q}} \qquad k \in [K],\ q \in Q^k \tag{2.16a}$$

$$\sum_{\boldsymbol{q} \in Q^u} \theta^u_{\boldsymbol{q}} = 1 \qquad u \in [U] \tag{2.16b}$$

$$\boldsymbol{\theta} \geq \mathbf{0} \tag{2.16c}$$

is a solution of (2.13) with value z^*. Indeed, (2.16a) and (2.15a) imply that $\boldsymbol{\theta}$ costs z^* with respect to objective function (2.13a), while (2.16a) and (2.15b) imply that constraints (2.13b) are satisfied. Finally, (2.16b) directly implies that (2.13c) is satisfied. It remains to show that (2.16) always has solutions. This is true, and one such solution is $\theta^u_{\boldsymbol{q}} = \lambda^{k*}_{\boldsymbol{q}} / U^k$, for $u \in [U]$ and $\boldsymbol{q} \in Q^u$. □

The Master LP (2.15) can be solved by the Column Generation Algorithm, where subproblem k, $k \in [K]$, is defined as:

$$\bar{c}^{k*} = \min\ (\boldsymbol{c}^k - \boldsymbol{\pi}^* \boldsymbol{A}^k)\boldsymbol{x}^k - \nu^*_k \tag{2.17a}$$

$$\text{s.t.} \qquad \boldsymbol{x}^k \in P^k, \tag{2.17b}$$

where $\boldsymbol{\nu}$ is the vector of dual variables corresponding to (2.15c).

Theorem 2.8: *At any iteration of the CGA for solving* (2.15)*, the value* $z_{RM} + \sum_{k \in [K]} U^k \bar{c}^{k*}$ *is a lower bound on its optimal cost.*

Proof. The dual of the MLP (2.15) is:

$$\max \quad \boldsymbol{\pi b} + \sum_{k\in[K]} U^k \nu_k \tag{2.18a}$$

$$\text{s.t.} \quad \boldsymbol{\pi A}^k \boldsymbol{q} + \nu_k \leq \boldsymbol{c}^k \boldsymbol{q} \qquad k \in [K],\, \boldsymbol{q} \in Q^k. \tag{2.18b}$$

Let $(\boldsymbol{\pi}^*, \boldsymbol{\nu}^*)$ be an optimal dual solution with value $z_{\text{RM}} = \boldsymbol{\pi}^* \boldsymbol{b} + \sum_{k\in[K]} U^k \nu_k^*$ of some RMLP. By the same reasoning used in the proof of Theorem 2.6, $(\boldsymbol{\pi}^*, \boldsymbol{\nu}^* + \overline{\boldsymbol{c}}^*)$, where $\overline{\boldsymbol{c}}^* = (\bar{c}^{1*} \ \ldots \ \bar{c}^{K*})$, is feasible for (2.18) and has value $z_{\text{RM}} + \sum_{k\in[K]} U^k \bar{c}^{k*}$. □

The MLP format (2.15) includes the situations where all subproblems are distinct ($U = K$) and even the single subproblem ($U = K = 1$) as particular cases. We make some remarks about the cases where $U > K$, i.e., when there are identical subproblems:

- Let $\boldsymbol{\lambda}^*$ be an optimal solution of (2.15). If $\boldsymbol{\lambda}^{k*} \neq \mathbf{0}$ for some $k \in [K]$ such that $U^k > 1$, then (2.16) has infinitely many solutions. Each such solution provides an alternative optimal solution to (2.13) and (by conversion using (2.12c)) an alternative optimal solution to the original LP. Note that the existence of those alternative solutions is an intrinsic feature of a symmetric original LP, not something that is created by the Dantzig-Wolfe reformulation. Truly, (2.11) can be rewritten as:

$$\min \ z = \sum_{k\in[K]} \boldsymbol{c}^k \boldsymbol{y}^k \tag{2.19a}$$

$$\text{s.t.} \quad \sum_{k\in[K]} \boldsymbol{A}^k \boldsymbol{y}^k = \boldsymbol{b} \tag{2.19b}$$

$$\boldsymbol{y}^k = \sum_{u\in U(k)} \boldsymbol{x}^u \qquad k \in [K] \tag{2.19c}$$

$$\boldsymbol{x}^u \in P^u \qquad u \in [U]. \tag{2.19d}$$

  This format makes clear that the objective function (2.19a) and constraints (2.19b) only depend on asymmetric *aggregated original variables* y and that there is a lot of freedom in choosing $\boldsymbol{x}^u$ values. For example, consider a situ-

ation where $K = 1$, i.e., all the U subproblems are identical. Given a solution $\boldsymbol{x} = (\boldsymbol{x}^1 \ \ldots \ \boldsymbol{x}^U)$ to (2.13), up to $U!$ distinct equivalent symmetric solutions can be obtained by only permuting the u indices. The fact that the identical subproblems case leads to many alternative symmetric original solutions has important negative consequences in some contexts, as will be shown in Chapters 4 and 8. Anyway, the aggregated original variables $\boldsymbol{y}^k$, $k \in [K]$, are uniquely determined from the solution of (2.15) by:

$$\boldsymbol{y}^k = \sum_{\boldsymbol{q} \in Q^k} \boldsymbol{q} \lambda_{\boldsymbol{q}}. \tag{2.20}$$

Consider the following example:

$$\begin{array}{lrcrcrcrcrcrcl}
\min z = & 3x_1 & + & 7x_2 & + & 2x_3 & - & x_4 & + & 3x_5 & + & 7x_6 & & \\
\text{s.t.} & & & 2x_2 & + & x_3 & - & x_4 & & & + & 2x_6 & \geq & 7 \\
& x_1 & + & x_2 & + & 2x_3 & + & 3x_4 & + & x_5 & + & x_6 & = & 12 \\
& x_1 & + & x_2 & & & & & & & & & \leq & 4 \\
& 3x_1 & + & x_2 & & & & & & & & & \leq & 6 \\
& & & & & x_3 & + & x_4 & & & & & \leq & 5 \\
& & & & & & & & & x_5 & + & x_6 & \leq & 4 \\
& & & & & & & & & 3x_5 & + & x_6 & \leq & 6 \\
& & & & & & & & & & & \boldsymbol{x} & \geq & \boldsymbol{0}.
\end{array} \tag{2.21}$$

By *keeping the first two constraints in the MLP* (i.e., by letting those constraints define (2.11b)), the remaining constraints decompose into three independent subproblems, defined by the following subsets of variables: $\boldsymbol{x}^1 = (x_1\, x_2)$, $\boldsymbol{x}^2 = (x_3\, x_4)$, and $\boldsymbol{x}^3 = (x_5\, x_6)$. Moreover, $\boldsymbol{c}^1 = \boldsymbol{c}^3 = (3\ 7)$, $A^1 = A^3 = \left(\begin{smallmatrix} 0 & 2 \\ 1 & 1 \end{smallmatrix}\right)$, and $P^1 = P^3$. So, the first and third subproblems are identical and define a group. The second subproblem alone defines a second group. So, we have $U = 3$, $K = 2$, $U^1 = 2$, and $U^2 = 1$. The resulting subproblems that indeed have to be solved are:

$$\begin{array}{lrcrcrcl}
\bar{c}^{1*} = & \min & (3 - \pi_2^*)\, x_1 & + & (7 - 2\pi_1^* - \pi_2^*)\, x_2 & - \nu_1 & & \\
& \text{s.t.} & x_1 & + & x_2 & \leq & 4 & \\
& & 3\, x_1 & + & x_2 & \leq & 6 & \\
& & & & \boldsymbol{x} & \geq & \boldsymbol{0}, &
\end{array}$$

and

$$\begin{array}{rlrcl} \overline{c}^{2*} = & \min & (2-\pi_1^*-2\pi_2^*)\,x_3 \;+\; (-1+\pi_1^*-3\pi_2^*)\,x_4-\nu_2 & & \\ & \text{s.t.} & x_3 \;+\; x_4 & \leq & 5 \\ & & \boldsymbol{x} & \geq & \mathbf{0}. \end{array}$$

Let $\lambda^1_{\boldsymbol{q}} = \theta^1_{\boldsymbol{q}} + \theta^3_{\boldsymbol{q}}$, $\boldsymbol{q} \in Q^1$, $\lambda^2_{\boldsymbol{q}} = \theta^2_{\boldsymbol{q}}$, $\boldsymbol{q} \in Q^2$, denote the aggregated master variables. Note that $(\,0\;0\,)$ clearly belongs to both P^1 and P^2 and could provide initial variables $\lambda^1_{(000)}$ and $\lambda^2_{(000)}$ that would be enough for satisfying the convexity constraints. However, whenever is known that a subproblem admits $\mathbf{0}$ as a solution it is usual to just relax the corresponding convexity constraint to $\leq$. This is correct because the slack variables of the relaxed constraints are equivalent to the variables generated by the $\mathbf{0}$ solutions. By also including two artificial variables, the first RMLP is:

$$\begin{array}{rlrcrcl} z_{\text{RM}} = & \min & 99a_1 \;+\; 99a_2 & & & & \\ & \text{s.t.} & a_1 & & & \geq & 7 \\ & & & & a_2 & = & 12 \\ & & & & & \leq & 2 \\ & & & & & \leq & 1 \\ & & & & \boldsymbol{a} & \geq & \mathbf{0}. \end{array}$$

Constraints of format $\leq b$, with a void left-hand side and $b \geq 0$ are always satisfied and, by convention, have a dual variable with value 0. So, the solution of the RMLP yields $z_{\text{RM}} = 1881$, $\boldsymbol{\pi}^* = (\,99\;99\,)$ and $\boldsymbol{\nu}^* = (\,0\;0\,)$. Subproblem $\overline{c}^{1*} = \min -96x_1 - 290x_2$, subject to $\boldsymbol{x}^1 \in P^1$ yields $\boldsymbol{x}^{1*} = (\,0\;\;4\,)$, with $\overline{c}^{1*} = -1160$. Subproblem $\overline{c}^{2*} = \min -295x_3 - 199x_4$, subject to $\boldsymbol{x}^2 \in P^2$ yields $\boldsymbol{x}^{2*} = (\,5\;0\,)$, with $\overline{c}^{2*} = -1475$. The second RMLP is:

$$\begin{array}{rlrcrcrcrcl} z_{\text{RM}} = & \min & 99a_1 \;+\; 99a_2 \;+\; 28\lambda^1_1 \;+\; 10\lambda^2_1 & & & & \\ & \text{s.t.} & a_1 & & +\; 8\lambda^1_1 \;+\; 5\lambda^2_1 & \geq & 7 \\ & & & a_2 \;+\; 4\lambda^1_1 \;+\; 10\lambda^2_1 & = & 12 \\ & & & & \lambda^1_1 & \leq & 2 \\ & & & & \lambda^2_1 & \leq & 1 \\ & & & & (\boldsymbol{a},\boldsymbol{\lambda}) & \geq & \mathbf{0}, \end{array}$$

with $z_{\text{RM}} = 24$, $\boldsymbol{\pi}^* = (\,0\;7\,)$ and $\boldsymbol{\nu}^* = (\,0\;\;-60\,)$. Subproblem $\min -4x_1$, subject to $\boldsymbol{x}^1 \in P^1$ yields $\boldsymbol{x}^{1*} = (\,2\;0\,)$, with $\overline{c}^{1*} = -8$. Subproblem $\min -12x_3 - 22x_4 + 60$,

subject to $\boldsymbol{x}^2 \in P^2$ yields $\boldsymbol{x}^{2*} = (\,0\;5\,)$, with $\overline{c}^{2*} = -50$. Adding the new columns and removing the artificial variables (not needed anymore), the third RMLP is:

$$\begin{array}{rrrrrrrrrrrr}
z_{\mathrm{RM}} = & \min & 28\lambda_1^1 & + & 10\lambda_1^2 & + & 6\lambda_2^1 & - & 5\lambda_2^2 & & \\
& \text{s.t.} & 8\lambda_1^1 & + & 5\lambda_1^2 & & & - & 5\lambda_2^2 & \geq & 7 \\
& & 4\lambda_1^1 & + & 10\lambda_1^2 & + & 2\lambda_2^1 & + & 15\lambda_2^2 & = & 12 \\
& & \lambda_1^1 & & & + & \lambda_2^1 & & & \leq & 2 \\
& & & & \lambda_1^2 & & & + & \lambda_2^2 & \leq & 1 \\
& & & & & & & & \boldsymbol{\lambda} & \geq & \mathbf{0},
\end{array}$$

with $z_{\mathrm{RM}} = 19$, $\boldsymbol{\pi}^* = (\,2.5\;\;2\,)$ and $\boldsymbol{\nu}^* = (\,0\;\;-22.5\,)$. Subproblem $\min x_1$, subject to $\boldsymbol{x}^1 \in P^1$ yields $\overline{c}^{1*} = 0$. Subproblem $\min -4.5x_3 - 4.5x_4 + 22.5$, subject to $\boldsymbol{x}^2 \in P^2$ yields $\overline{c}^{2*} = 0$. So, the primal RMLP solution ($\lambda_1^1 = 0.375$, $\lambda_2^1 = 0$, $\lambda_1^2 = 0.9$, $\lambda_2^2 = 0.1$) gives an optimal solution of the MLP, which can be converted into optimal solutions for the original LP. The variables that defined the unique second subproblem have the unique value $\boldsymbol{x}^2 = (\,x_3\;\;x_4\,) = 0.9\,(\,5\;0\,) + 0.1\,(\,0\;5\,) = (\,4.5\;0.5\,)$. On the other hand, there are multiple possible values for the other variables, depending on how the values of the aggregated variables $\boldsymbol{\lambda}^1$ are distributed to $\boldsymbol{\theta}^1$ and $\boldsymbol{\theta}^2$ variables. The even division used in the proof of Theorem 2.7 yields $\boldsymbol{x}^1 = (\,x_1\;\;x_2\,) = \boldsymbol{x}^3 = (\,x_5\;\;x_6\,) = 0.1875\,(\,0\;4\,) = (\,0\;\;0.75\,)$. Alternative optimal solutions include $(\,x_1\;x_2\;x_5\;x_6\,) = (\,0\;1.5\;0\;0\,)$ and its symmetric solution $(\,x_1\;x_2\;x_5\;x_6\,) = (\,0\;0\;0\;1.5\,)$. The aggregated original variables y corresponding to the identical subproblems could be directly obtained using (2.20): $\boldsymbol{y}^1 = 0.375\,(\,0\;4\,) = (\,0\;1.5\,)$. Of course, all possible solutions satisfy $\boldsymbol{x}^1 + \boldsymbol{x}^3 = \boldsymbol{y}^1$.

2.4. Unbounded Subproblems

The previous sections assumed that the subproblems are always defined by bounded polyhedra. That assumption avoided some technicalities that could obscure the presentation of the main ideas. However, it is quite possible that a Dantzig-Wolfe reformulation leads to unbounded subproblem LPs, even if the original problem is not unbounded.

One way of dealing with that possibility would be by adding lower and up-

per bounds $l_j \leq x_j \leq u_j$, $j \in [n]$, on each individual variable. Those bounds would be part of the polyhedra defining the subproblems, making sure that all of them are bounded. Variable bounds are treated implicitly in modern simplex implementations, they do not make the algorithm slower. If those bounds are not readily available from modeling considerations, one may impose very large bounds $-M_j \leq x_j \leq M_j$ on each variable. Those "artificial bounds" can be viewed as a device for detecting unboundedness: if some variable x_j has value $-M_j$ or M_j in the final solution of the original LP, then this LP is unbounded. This is akin to the use of artificial variables with very large costs for detecting infeasibility.

However, it is possible to handle unbounded subproblems directly, by generalizing the definition of Dantzig-Wolfe reformulation and the Column Generation Algorithm. We need some additional theoretical results for characterizing unbounded polyhedra.

Definition 2.5: Conic combination, conic hull. Given a finite set of points $D = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_t\}$ in an n-dimensional space, $\boldsymbol{d}$ is a *conic combination* of the vectors in D if $\boldsymbol{d} = \sum_{j\in[t]} \boldsymbol{d}_j \mu_j$ for some $\boldsymbol{\mu} \in \mathbb{R}^t_+$. The *conic hull* of a set $D \in \mathbb{R}^n$ (finite or not), denoted by $Cone(D)$, is the set of all points obtainable as a conic combination of some finite subset of points in D.

Definition 2.6: Ray, normalized representation of a ray, extreme ray. A point (interpreted as a direction) $\boldsymbol{d} \in \mathbb{R}^n \setminus \{\boldsymbol{0}\}$ defines a *ray* of the unbounded polyhedron P if for any $\boldsymbol{x} \in P$ and $\theta \geq 0$, $\boldsymbol{x} + \theta\boldsymbol{d}$ also belongs to P. Any positive multiple of $\boldsymbol{d}$, i.e., any direction $\boldsymbol{d'} = \alpha\boldsymbol{d}$, for $\alpha > 0$, defines the same ray. A *normalized representation of a ray* is the unique direction $\boldsymbol{d}$ defining it such that $||\boldsymbol{d}||_1 = \sum_{i\in[n]} |d_i| = 1$ (d_i is the i-th element of vector $\boldsymbol{d}$). A ray of P is *extreme* if its normalized representation is not a conic combination of the normalized representations of other rays of P. Let $ExtRay(P)$ denote the set containing the normalized representation of each extreme ray in P.

Theorem 2.9: *For any polyhedron P, $ExtRay(P)$ is finite.*

Theorem 2.10: *For any polyhedron P, $P = Conv(Ext(P)) + Cone(ExtRay(P))$. In other words, $P = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} = \sum_{\boldsymbol{q}\in Q} \boldsymbol{q}\lambda_{\boldsymbol{q}} + \sum_{\boldsymbol{r}\in R} \boldsymbol{r}\mu_{\boldsymbol{r}}, \sum_{\boldsymbol{q}\in Q} \lambda_{\boldsymbol{q}} = 1, (\boldsymbol{\lambda}, \boldsymbol{\mu}) \geq \boldsymbol{0}\}$, where $Q = Ext(P)$ and $R = ExtRay(P)$ (note that we are using the vector*

$\boldsymbol{r} \in R$ *itself as the index of its corresponding* μ *variable).*

The above result remains correct if $ExtRay(P)$ is replaced by any set containing exactly one direction defining each extreme ray of P. In fact, any point in a polyhedron can be written as a convex combination of its extreme points plus a conic combination of directions defining its extreme rays. Anyway, by Theorem 2.10, if in the original LP (2.1) the polyhedron P is unbounded, the Master LP becomes:

$$z_{\mathrm{M}} \;=\; \min \quad \sum_{\boldsymbol{q}\in Q}(\boldsymbol{cq})\lambda_{\boldsymbol{q}} + \sum_{\boldsymbol{r}\in R}(\boldsymbol{cr})\mu_{\boldsymbol{r}} \tag{2.22a}$$

$$\text{s.t.} \quad \sum_{\boldsymbol{q}\in Q}(\boldsymbol{Aq})\lambda_{\boldsymbol{q}} + \sum_{\boldsymbol{r}\in R}(\boldsymbol{Ar})\mu_{\boldsymbol{r}} = \boldsymbol{b} \tag{2.22b}$$

$$\sum_{\boldsymbol{q}\in Q}\lambda_{\boldsymbol{q}} = 1 \tag{2.22c}$$

$$(\ \boldsymbol{\lambda},\ \boldsymbol{\mu}\ ) \geq \boldsymbol{0}. \tag{2.22d}$$

To solve it, the Restricted Master LP is initialized with a small subset of its columns and/or artificial variables. The Column Generation Algorithm remains the same, except on the iterations where the subproblem LP (2.7) is unbounded. In such a case, it is assumed that the subproblem solver can provide an unbounded direction $\boldsymbol{d}^*$ defining a ray of P.

- For example, if the subproblem is being solved by the RSA, the vector $\boldsymbol{d}$ found in the last RSA Step 4 (page 6) yields such a direction, which has components $-\boldsymbol{d}$ for the current basic variables, 1 for the entering variable and zero for the remaining non-basic variables.

Then, variable $\mu_{\boldsymbol{d}^*}$ (usually notated as μ_j, where j is its sequential generation number) with cost $\boldsymbol{cd}^*$ and column $\left(\begin{smallmatrix}\boldsymbol{Ad}^*\\ 0\end{smallmatrix}\right)$ is added to the Restricted Master LP. The unboundedness of $\boldsymbol{d}^*$ in the subproblem means that for any fixed $\boldsymbol{x} \in P$, $\lim_{\theta\to\infty}(\boldsymbol{c} - \boldsymbol{\pi}^*\boldsymbol{A})(\boldsymbol{x} + \theta\boldsymbol{d}^*) = -\infty$. This implies that $(\boldsymbol{c} - \boldsymbol{\pi}^*\boldsymbol{A})\boldsymbol{d}^* < 0$. So, the added variable has a negative reduced cost. In order to say that the RMLP indeed contains a subset of the columns in (2.22) (i.e., all generating rays are in the normalized set $R = ExtRay(P)$), it would be possible to normalize $\boldsymbol{d}^*$ before inserting the corresponding column. But this is not really necessary, the algorithm remains correct if the generating ray $\boldsymbol{d}^*$ is not normalized. However, the used $\boldsymbol{d}^*$ should be kept in a side table because it may be needed for recovering the solution of the

original LP. *If an RMLP happens to be unbounded, then the MLP and the original LP are also unbounded.* As the λ variables are always limited by (2.22c), RMLP unboundedness is only possible if some μ variables are added to it.

2.5. Subproblems with Network Flow Structure

A network is a directed graph $G = (V, A)$ plus arc costs c_a and arc capacities u_a, $a \in A$, and vertex demands d_i, $i \in V$, such that $\sum_{i \in V} d_i = 0$ (vertices with negative demands are sources, those with positive demands are sinks). The minimum cost network flow problem, or simply, the *Network Flow Problem* (NFP) is the LP:

$$\min \quad z = \sum_{a \in A} c_a x_a \tag{2.23a}$$

$$\text{s.t.} \quad \sum_{a \in \delta^-(i)} x_a - \sum_{a \in \delta^+(i)} x_a = d_i \qquad i \in V \tag{2.23b}$$

$$0 \leq x_a \leq u_a \qquad a \in A, \tag{2.23c}$$

where variable x_a denotes the flow on arc a. Constraints (2.23b) are known as flow conservation equalities. There are some very efficient algorithms tailored for solving this particular kind of LP [Kovács, 2015], one of them being the *network simplex algorithm*. The network flow problem includes as particular cases the shortest path problem, the max-flow/min-cut problem, the assignment problem, the transportation problem, and the transshipment problem. Algorithms even more specialized for those cases can be extremely efficient. A comprehensive book on the topic is Ahuja et al. [1993]. Network flow problems have the following *integrality property*: if all demands and capacities are integer, then all extreme points of the polyhedron defined by (2.23b–2.23c) are integer.

2.5.1. Decomposition into Flows

One of the most promising situations for a Dantzig-Wolfe reformulation in linear programming is certainly when it yields a decomposition into subproblems having a network flow structure. In this section, we will illustrate this using the classic

Multi-commodity Network Flow Problem (MNFP): suppose there are K distinct flows (each flow associated with a commodity) sharing the same arc capacities. In other words, arc costs c_a^k and vertex demands d_i^k may vary depending on the commodity k, $k \in [K]$, but there is a global arc capacity u_a limiting the sum of the flows over a. The LP is:

$$\min\ z\ = \sum_{k\in[K]} \sum_{a\in A} c_a^k x_a^k \tag{2.24a}$$

$$\text{s.t.} \quad \sum_{a\in\delta^-(i)} x_a^k - \sum_{a\in\delta^+(i)} x_a^k = d_i^k \qquad i \in V,\ k \in [K] \tag{2.24b}$$

$$\sum_{k\in[K]} x_a^k \le u_a \qquad a \in A \tag{2.24c}$$

$$x_a^k \ge 0 \qquad a \in A,\ k \in [K], \tag{2.24d}$$

where variable x_a^k denotes the flow of commodity k on arc a. By keeping Constraints (2.24c) in the master, the remaining constraints decompose into K independent blocks. The polyhedra defined by those constraints are not bounded if G contains cycles. In order to avoid the need to handle extreme rays, we may use the global arc capacities as individual variable upper bounds in the subproblems. Therefore, for each $k \in [K]$, define

$$P^k = \{\boldsymbol{x}^k \mid \sum_{a\in\delta^-(i)} x_a^k - \sum_{a\in\delta^+(i)} x_a^k = d_i^k, i \in V; 0 \le x_a^k \le u_a, a \in A\},$$

and $Q^k = Ext(P^k)$. The vectors in Q^k correspond to complete flows for commodity k; for a given $\boldsymbol{f} \in Q^k$, f_a is the flow value for arc $a \in A$. The resulting MLP is:

$$z_{\mathrm{M}} = \min \sum_{k\in K} \sum_{\boldsymbol{f}\in Q^k} \left(\sum_{a\in A} c_a^k f_a \right) \lambda_{\boldsymbol{f}}^k \tag{2.25a}$$

$$\text{s.t.} \quad \sum_{k\in K} \sum_{\boldsymbol{f}\in Q^k} f_a \lambda_{\boldsymbol{f}}^k \le u_a \qquad a \in A \tag{2.25b}$$

$$\sum_{\boldsymbol{f}\in Q^k} \lambda_{\boldsymbol{f}}^k = 1 \qquad k \in [K] \tag{2.25c}$$

$$\boldsymbol{\lambda} \ge 0. \tag{2.25d}$$

Let $\boldsymbol{\pi}$ and $\boldsymbol{\nu}$ be the dual variables associated with capacity constraints (2.25b) and convexity constraints (2.25c) respectively. If $(\boldsymbol{\pi}^*, \boldsymbol{\nu}^*)$ is an optimal dual solution of an RMLP, the pricing subproblem for $k \in [K]$ is:

$$\bar{c}^{k*} = \min \sum_{a \in A} (c_a^k - \pi_a^*)\, x_a^k - \nu_k \tag{2.26a}$$

$$\text{s.t.} \quad \boldsymbol{x}^k \in P^k. \tag{2.26b}$$

It can be solved by any minimum cost network flow algorithm, like the network simplex.

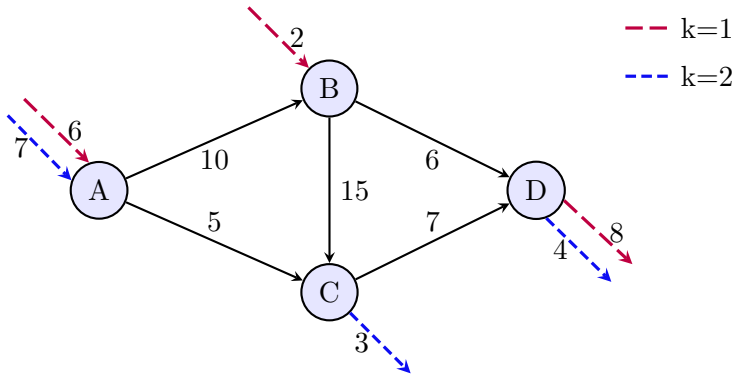


Figure 2.2: A multi-commodity flow instance with two commodities.

As an example, consider a MNFP instance defined over the graph G depicted in Figure 2.2 (the arc capacities are also shown), $K = 2$ and with the following costs and demands:

		arc cost (c_a^k)					vertex demand (d_i^k)			
		AB	**AC**	**BC**	**BD**	**CD**	**A**	**B**	**C**	**D**
k	**1**	3	2	2	4	1	-6	-2	0	8
	2	4	2	1	2	5	-7	0	3	4

Solving the MLP (2.25) using the CGA, the final RMLP is:

$$
\begin{array}{rlrcrcrcrcl}
z_{\mathrm{RM}} = & \min & 24\lambda_1^1 & + & 30\lambda_1^2 & + & 42\lambda_2^1 & + & 39\lambda_2^2 & + & 50\lambda_3^1 & \\
& \text{s.t.} & & & 4\lambda_1^2 & + & 6\lambda_2^1 & + & 7\lambda_2^2 & + & 6\lambda_3^1 & \leq & 10 \\
& & 6\lambda_1^1 & + & 3\lambda_1^2 & & & & & & & \leq & 5 \\
& & 2\lambda_1^1 & & & + & 8\lambda_2^1 & + & 3\lambda_2^2 & & & \leq & 15 \\
& & & & 4\lambda_1^2 & & & + & 4\lambda_2^2 & + & 8\lambda_3^1 & \leq & 6 \\
& & 8\lambda_1^1 & & & + & 8\lambda_2^1 & & & & & \leq & 7 \\
& & \lambda_1^1 & & & & \lambda_2^1 & & & + & \lambda_3^1 & = & 1 \\
& & & & \lambda_1^2 & & & + & \lambda_2^2 & & & = & 1 \\
& & & & & & & & & & \boldsymbol{\lambda} & \geq & \mathbf{0},
\end{array}
$$

$z_{\mathrm{RM}} = z_{\mathrm{M}} = 67$, its optimal dual solution is $\boldsymbol{\pi}^* = (\,0\ -3\ 0\ 0\ -1\,)$, $\boldsymbol{\nu}^* = (\,50\ 39\,)$, while its optimal primal solution $\lambda_1^1 = \frac{5}{6}, \lambda_1^2 = 0, \lambda_2^1 = \frac{1}{24}, \lambda_2^2 = 1, \lambda_3^1 = \frac{1}{8}$ can be converted into original flow variables, obtaining $\boldsymbol{x}^{1*} = (\,x^1_{\mathrm{AB}}\ x^1_{\mathrm{AC}}\ x^1_{\mathrm{BC}}\ x^1_{\mathrm{BD}}\ x^1_{\mathrm{CD}}\,) = (\,1\ 5\ 2\ 1\ 7\,)$, $\boldsymbol{x}^{2*} = (\,x^2_{\mathrm{AB}}\ x^2_{\mathrm{AC}}\ x^2_{\mathrm{BC}}\ x^2_{\mathrm{BD}}\ x^2_{\mathrm{CD}}\,) = (\,7\ 0\ 3\ 4\ 0\,)$.

2.5.2. Decomposition into Paths+Cycles

The standard DW decomposition presented above may *not* lead to the best way of using CG when subproblems have a network flow structure. As already perceived in Ford Jr and Fulkerson [1958] (see Note 2.3), potentially better CGAs can be obtained by a reformulation that also uses another principle: any flow can be decomposed into source-sink paths + cycles. A proof for the following result can be found in Ahuja et al. [1993].

Theorem 2.11: *Consider a network flow problem and its formulation as an LP (2.23). Let Ω be the finite set of all elementary (not repeating vertices) paths in G that start at a source vertex and end at a sink vertex, represented as incidence vectors in $\mathbb{B}^A$. In other words, given a vector $\boldsymbol{p} \in \Omega$, p_a indicates whether arc $a \in A$ belongs to the corresponding path. Let Θ be the finite set of all elementary cycles in G, also represented as incidence vectors in $\mathbb{B}^A$. For any solution $\boldsymbol{x}$ of (2.23) there are vectors $(\boldsymbol{\lambda}, \boldsymbol{\mu}) \in \mathbb{R}_+^{|\Omega|\times|\Theta|}$ such that $\boldsymbol{x} = \sum_{\boldsymbol{p}\in\Omega} \boldsymbol{p}\lambda_{\boldsymbol{p}} + \sum_{\boldsymbol{r}\in\Theta} \boldsymbol{r}\mu_{\boldsymbol{r}}$.*

We present an alternative way of reformulating a MNFP. Assume for the moment that there is a single source s^k and a single sink t^k for each $k \in [K]$and let Ω^k be the set of $s^k - t^k$ elementary paths in G, while Θ is the set of cycles in G. The alternative Master LP is:

$$z_{\mathrm{M}} = \min \sum_{k\in K}\left(\sum_{\boldsymbol{p}\in\Omega^k}\sum_{a\in A} c_a^k p_a \lambda_{\boldsymbol{p}}^k + \sum_{\boldsymbol{r}\in\Theta}\sum_{a\in A} c_a^k r_a \mu_{\boldsymbol{r}}^k\right) \tag{2.27a}$$

$$\text{s.t.} \quad \sum_{k\in K}\left(\sum_{\boldsymbol{p}\in\Omega^k} p_a \lambda_{\boldsymbol{p}}^k + \sum_{\boldsymbol{r}\in\Theta} r_a \mu_{\boldsymbol{r}}^k\right) \le u_a \qquad a \in A \tag{2.27b}$$

$$\sum_{\boldsymbol{p}\in\Omega^k} \lambda_{\boldsymbol{p}}^k = d_{t^k} \qquad k \in [K] \tag{2.27c}$$

$$\boldsymbol{\lambda} \ge 0. \tag{2.27d}$$

Variables $\lambda_{\boldsymbol{p}}^k$ and $\mu_{\boldsymbol{r}}^k$ indicate how many units of flow k are carried over path $\boldsymbol{p}$ or cycle $\boldsymbol{r}$, respectively. The "convexity constraints" (2.27c) enforce that the demands are satisfied. Let $\boldsymbol{\pi}$ and $\boldsymbol{\nu}$ be the dual variables associated with (2.27b) and (2.27c) respectively. The pricing subproblem for each $k \in [K]$ can be split into two parts:

$$\overline{c}^{k1*} = \min \sum_{a\in A}(c_a^k - \pi_a^*)p_a - \nu_k \tag{2.28a}$$

$$\text{s.t.} \quad \boldsymbol{p} \in \Omega^k, \tag{2.28b}$$

and

$$\overline{c}^{k2*} = \min \sum_{a\in A}(c_a^k - \pi_a^*)r_a \tag{2.29a}$$

$$\text{s.t.} \quad \boldsymbol{r} \in \Theta. \tag{2.29b}$$

Both parts can be handled at once by the Bellman-Ford shortest path algorithm. That algorithm runs in $O(|V||E|)$ time and can be applied to a network containing both positive and negative cost arcs: it will either find shortest elementary paths from a single source vertex to all other vertices or it will find a cycle with negative cost if one exists. In this pricing context, the arc costs given to the algorithm are the modified costs $(c_a^k - \pi_a^*)$, $a \in A$. If a negative cost cycle $\boldsymbol{r}^*$ is found, then $\overline{c}^{k2*} < 0$,

variable $\mu^k_{\boldsymbol{r}*}$ has a negative reduced cost and is added to the RMLP. Of course, after the RMLP is re-optimized $\mu^k_{\boldsymbol{r}*}$ will not have negative reduced cost anymore. This means that *the overall effect of the μ variables is canceling possible negative cost cycles*, changing the dual variables in such a way that the Bellman-Ford algorithm will eventually start finding shortest paths.

The advantages of decomposition into paths over decomposition into flows are:

- The pricing subproblems can be solved by the Bellman-Ford algorithm, which is more efficient than the minimum cost flow algorithms. But one may do even better. In many situations, it is possible to know beforehand that there are optimal solutions of (2.23) not using cycles in their decompositions. An obvious such situation is when G is acyclic (so $\Theta = \emptyset$). The shortest path problem in acyclic graphs can be solved in linear $O(|A|)$ time by topological sorting, regardless of the sign of the modified arc costs. In practice, this means that paths can be very quickly priced even on large networks. A second favorable situation is the case where G has cycles but all original arc costs c^k are non-negative. Noting that $\boldsymbol{\pi} \leq 0$, all modified costs $(c^k_a - \pi^*_a)$ will be non-negative too. Therefore, the pricing subproblems can be solved in $O(|A| + |V| \log |V|)$ by Dijkstra's algorithm.

- The convergence of the CGA can be much faster. The reasons will be fully explained in Chapter 7. We may advance that a RMLP from (2.27) having a certain number of path-columns may be equivalent to a RMLP from (2.25) having many more flow-columns, those corresponding to all the flows that can be obtained as a combination of those paths. Another potential advantage is that the mentioned shortest-path algorithms actually obtain the shortest paths from a source to all other vertices in the graph. That information may be used to identify and introduce multiple path-columns with negative reduced costs in a single iteration, speeding up the CGA.

We now generalize this decomposition scheme for the cases where commodities may have multiple sources and/or sinks. Let S^k and T^k be the sets of source and sink vertices for commodity k, $k \in [K]$. Let Ω^k be the union of the sets of all $s^k - t^k$ elementary paths in G, $s^k \in S^k$, $t^k \in T^k$. Let $S(\Omega^k, i)$ and $T(\Omega^k, i)$ be the subsets

of paths in Ω^k that start and end in a vertex i, respectively. The MLP is:

$$\min \quad \sum_{k\in K}\left(\sum_{\boldsymbol{p}\in\Omega^k}\sum_{a\in A} c_a^k p_a \lambda_{\boldsymbol{p}}^k + \sum_{\boldsymbol{r}\in\Theta}\sum_{a\in A} c_a^k r_a \mu_{\boldsymbol{r}}^k\right) \tag{2.30a}$$

$$\text{s.t.} \quad \sum_{k\in K}\left(\sum_{\boldsymbol{p}\in\Omega^k} p_a \lambda_{\boldsymbol{p}}^k + \sum_{\boldsymbol{r}\in\Theta} r_a \mu_{\boldsymbol{r}}^k\right) \leq u_a \qquad a \in A \tag{2.30b}$$

$$\sum_{\boldsymbol{p}\in T(\Omega^k,i)} \lambda_{\boldsymbol{p}}^k = d_i \qquad k \in [K],\ i \in T^k \tag{2.30c}$$

$$\sum_{\boldsymbol{p}\in S(\Omega^k,i)} \lambda_{\boldsymbol{p}}^k = -d_i \qquad k \in [K],\ i \in S^k \tag{2.30d}$$

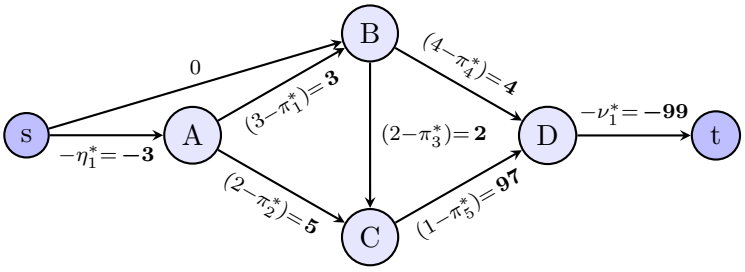
$$\boldsymbol{\lambda} \geq 0. \tag{2.30e}$$

Note that, for each $k \in K$, one constraint in (2.30c) or in (2.30d) may be chosen to be eliminated from the MLP since it is implied by the remaining constraints. Let $(\boldsymbol{\pi}, \boldsymbol{\nu}, \boldsymbol{\eta})$ be the vectors of dual variables corresponding to (2.30b), (2.30c), and (2.30d) respectively. The pricing subproblem $k \in K$ can be solved as a shortest $s-t$ path over a modified graph $G' = (V', A')$, where $V' = V \cup \{s, t\}$ and $A' = A \cup \{(i,t) : i \in T^k\} \cup \{(s,i) : i \in S^k\}$. The cost of an arc $a \in A$ is $(c_a^k - \pi_a^*)$; arc (i,t), $i \in T^k$, receives cost ν_i^*, and arc (s,i), $i \in S^k$, gets cost η_i^*. Note that every $s-t$ path in G' passes by exactly one arc leaving s and by exactly one arc entering t. Therefore, it is possible to avoid possible negative costs in those arcs in $A' \setminus A$ (if the arcs in A have non-negative costs and one wants to use Dijkstra's algorithm) by adding a suitable positive constant to all of them. That would not change the shortest paths. Of course, later it is necessary to subtract two times the added constant from the optimal solution value in order to determine if a path leads to a variable with negative reduced cost.

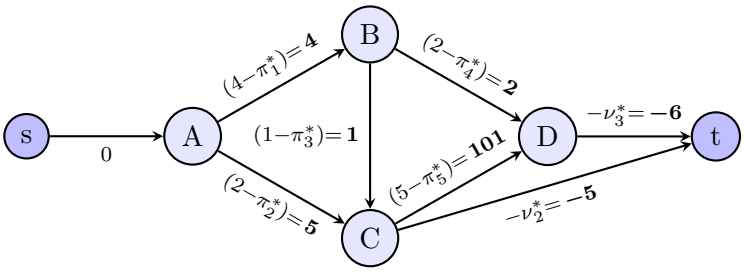
Consider again the MNFP instance depicted in Figure 2.2. Solving the MLP (2.30) using the CGA, the fourth iteration RMLP is:

$$
\begin{array}{rlrrrrrrrrrr}
z_{\mathrm{RM}} = & \min & 99a_1 & 3\lambda_1^1 & +2\lambda_1^2 & +6\lambda_2^1 & +5\lambda_2^2 & +3\lambda_3^1 & +6\lambda_3^2 & & \\
& \text{s.t.} & & & & \lambda_2^1 & +\lambda_2^2 & & +\lambda_3^2 & \leq & 10 \\
& & & \lambda_1^1 & +\lambda_1^2 & & & & & \leq & 5 \\
& & & & & \lambda_2^1 & +\lambda_2^2 & +\lambda_3^1 & & \leq & 15 \\
& & & & & & & & \lambda_3^2 & \leq & 6 \\
& & & \lambda_1^1 & & +\lambda_2^1 & & +\lambda_3^1 & & \leq & 7 \\
& & a_1 & +\lambda_1^1 & & +\lambda_2^1 & & +\lambda_3^1 & & = & 8 \\
& & & & \lambda_1^2 & & +\lambda_2^2 & & & = & 3 \\
& & & & & & & & +\lambda_3^2 & = & 4 \\
& & & \lambda_1^1 & & +\lambda_2^1 & & & & = & 6 \\
& & & & & & & & \boldsymbol{\lambda} & \geq & \mathbf{0},
\end{array}
$$

The RMLP has five capacity constraints, three constraints corresponding to (2.30c) and only one constraint related to (2.30d) (the two omitted constraints in (2.30d), one for each commodity, are redundant). The optimal dual solution is $\boldsymbol{\pi}^* = (\,0\ -3\ 0\ 0\ -96\,)$, $\boldsymbol{\nu}^* = (\,99\ 5\ 6\,)$, $\boldsymbol{\eta}^* = (\,3\,)$ with $z_{\mathrm{RM}} = 162$. Figures 2.3a and 2.3b show the graphs corresponding to the shortest $s-t$-path pricing subproblems, the modified arc costs are also shown. For $k = 1$, a negative cost path $s - B - D - t$ with $\bar{c}^{1*} = -95$ is found. A new variable λ_4^1 corresponding to path $B - D \in \Omega^1$ is added to the RMLP. For $k = 2$, no negative cost path exists. The 5-th iteration RMLP dual solution is $\boldsymbol{\pi}^* = (\,0\ -3\ 0\ 0\ -1\,)$, $\boldsymbol{\nu}^* = (\,4\ 5\ 6\,)$, $\boldsymbol{\eta}^* = (\,3\,)$ with $z_{\mathrm{RM}} = 67$. The subsequent pricing subproblems find no paths with negative cost, so $z_{\mathrm{M}} = z_{\mathrm{RM}}$. The optimal primal solution is $\lambda_1^1 = 5, \lambda_1^2 = 0, \lambda_2^1 = 1, \lambda_2^2 = 3, \lambda_3^1 = 1, \lambda_3^2 = 4, \lambda_4^1 = 1$. By converting it into the original arc flow variables, we obtain: $\boldsymbol{x}^1 = (\,x^1_{\mathrm{AB}}\ x^1_{\mathrm{AC}}\ x^1_{\mathrm{BC}}\ x^1_{\mathrm{BD}}\ x^1_{\mathrm{CD}}\,) = 5(\,0\ 1\ 0\ 0\ 1\,) + (\,1\ 0\ 1\ 0\ 1\,) + (\,0\ 0\ 1\ 0\ 1\,) + (\,0\ 0\ 0\ 1\ 0\,) = (\,1\ 5\ 2\ 1\ 7\,)$; similarly, $\boldsymbol{x}^2 = (\,x^2_{\mathrm{AB}}\ x^2_{\mathrm{AC}}\ x^2_{\mathrm{BC}}\ x^2_{\mathrm{BD}}\ x^2_{\mathrm{CD}}\,) = 3(\,1\ 0\ 1\ 0\ 0\,) + 4(\,1\ 0\ 0\ 1\ 0\,) = (\,7\ 0\ 3\ 4\ 0\,)$.



(a) $k = 1$



(b) $k = 2$

Figure 2.3: Graphs for shortest $s - t$ path pricing subproblems. Modified costs for the 4th CGA iteration are shown.

2.6. Case Study: Public Transportation Planning

Salimifard and Bigharaz [2022] surveys 263 articles published between 2000 and 2019 on the MNFP, divided into applications (mostly in logistics and telecommunications) and methodology. Many of the considered MNFPs have an underlying *space-time network*, which is a modeling device that introduces a temporal dimension into flows. In these networks, individual vertices represent various time instants at each spatial location. This results in much more accurate models in situations where time is a critical factor, but often leads to very large LPs that may need column generation to be solved. In some cases, the application actually requires integer solutions. However, although the MNFP formulation does not have the integrality property it is usually very strong (in the sense defined in Chapter 3), and obtaining near-optimal integer solutions is not hard once an optimal fractional solution is

found. So, even in those cases, the bottleneck is solving LP (2.24).

In order to be more concrete, we will present a case study, describing a recent use of CG in solving some large-scale MNFPs. Lienkamp and Schiffer [2024] consider Munich's Public Transportation System in Germany, a complex interplay of buses, subways, and trams. In that system, buses have a capacity of 60 passengers, subway trains can accommodate up to 940 passengers, and trams can hold 215 passengers. The MNFP over a space-time network model assumes that the transportation schedules are already given and looks for the best ways of utilizing the system to meet the existing set of demands. In fact, the model works as a tactical planning tool, evaluating how possible changes in the fleets and the schedules would impact the quality of service. In that context, as stated by the authors: *"to analyze large transportation system's optima and thus the potential of different transportation systems, finding fractional solutions often suffices"*. Yet, after the MNFP is optimally solved by a CGA, integer solutions of excellent quality can be quickly found by solving the final RMLP as an IP.

There are K demands, which correspond to the commodities. A demand $k \in K$ is characterized by:

- Origin location and time – for example, a suburban block at 7:15.
- Destination location – for example, a downtown block.
- Max transit time – arrivals at the destination after origin time plus max transit time are penalized.
- Number of passengers - the authors actually work with unitary demands but scale the capacities. However, dividing the capacities by a factor b is equivalent to having demands with b passengers.

Note that values of K in the range of hundreds yield a rough model where each demand aggregates many individual passengers. Accurate modeling requires much larger values of K. The space-time network contains walking arcs (for example, starting at 7:15 at a certain origin location and ending at 7:19 at a bus stop A), waiting arcs (say, a 5-minute wait at the bus stop A), and bus/tram/subway transportation arcs (for example, a bus leaving stop A at 7:24 and arriving at stop B at 7:27). Only transportation arcs are capacitated.

We reproduce some computational results. Table 2.1 shows a comparison of the CG approach with a direct solution by commercial solver Gurobi 9.5, stopped at the

first integer solution. The CG results were obtained by a basic implementation that uses Dijkistra's algorithm to solve the pricing subproblems. The reported times are averages over 10 instances. Gurobi ran out of memory in 8 instances with $K = 662$, indicating that it could not handle larger values of K. The substantial time spent on building model (2.24) (longer than the time taken by Gurobi to solve it) is partially explained by the use of the less-performing Python language for that task. However, this is essentially due to the fact that the model contains K copies (with different sources, sinks, and costs) of a large time-space network. On the other hand, that network only needs to be built once for the CG approach.

Table 2.1: Comparison of a MIP solver and CG

K	132		308		486		662	
Method	CG	Gu	CG	Gu	CG	Gu	CG	Gu
Solved instances	10	10	10	10	10	10	10	2
Avg. build time (s)	<1	48	<1	142	<1	315	<1	553
Avg. solving time (s)	1	33	3	109	6	228	10	463
Avg. total time (s)	1	82	3	252	6	542	10	1016

Source: Lienkamp and Schiffer [2024].

Table 2.2 presents results for much larger values of K, up to 56,295. Again, times are averages of over ten instances. Those runs correspond to a more sophisticated CG implementation that uses an A* algorithm in the pricing and a filter for selecting the subproblems that will be solved in each iteration (in the last iteration all subproblems should be solved to ensure optimality). The times to solve the final RMLP as an IP is indicated. In all tests, the integer solution thus found had a value that matched the LP value, so it was optimal. The overall conclusion of that case study is that a CG approach may solve in reasonable time some LPs far too big to even fit into the memory of a modern computer.

2.7. Assessment of Column Generation for solving LPs

We finish this chapter by commenting on the question: Is Dantzig-Wolfe reformulation and Column Generation a good practical alternative for solving an LP, when

Table 2.2: CG results on large instances

K	6,255	18,765	31,275	43,785	56,295
Solved instances	10	10	10	10	10
Avg. total time (s)	66	370	900	1766	2856
Avg. IP time (s)	1.8	4.0	5.0	3.9	4.6
Avg. optimality gap	0.0%	0.0%	0.0%	0.0%	0.0%

Source: Lienkamp and Schiffer [2024].

compared with solving the original LP directly using a simplex or interior-point-based algorithm?

- If the decomposition leads to a single subproblem (instead of leading to multiple, distinct or identical, subproblems) and that problem has to be solved by a general LP solver (no special structure to be exploited), the answer is simple: *"No!"*. As will be seen in Chapter 5 and Chapter 7, the single subproblem case usually leads to a very poor convergence of the CGA. Solving a relatively large subproblem LP many times is just too time-consuming.

- If the decomposition leads to multiple subproblems, the answer is *"Perhaps. You should test."*. The more subproblems, the better the expected convergence of the CGA, and the smaller those subproblems. If the subproblems have a structure that allows them to be solved by faster specialized algorithms, using these algorithms definitely helps a lot. A network flow structure in the subproblems is particularly favorable due to the possibility of using shortest path algorithms for the pricing (like in the case study in Section 2.6). However, the best modern general LP solvers are highly efficient and may still be competitive in solving the original LP, even with a structure favorable to the Dantzig-Wolfe decomposition. In several cases (see Note 2.13), the CGA only beats state-of-the-art simplex-based algorithms if the subproblems are solved in parallel, using multiple processors or cores. The potential for natural and easy parallelization is another important feature of a DW decomposition.

The Dantzig-Wolfe reformulation and column generation for pure linear programming can be viewed as niche techniques, that beat standard LP solvers only in certain kinds of problems. However, those techniques may play a much more central role in integer programming and combinatorial optimization, as will be seen in Chapter 4.

Notes

2.1. Proofs for all basic polyhedral results in this chapter can be found in Bazaraa et al. [2010]. Theorem 2.10 (its particular case for bounded polyhedra was stated as Theorem 2.4) is known as Minkowski-Weyl Theorem.

2.2. The reformulation and decomposition method for general LPs by Dantzig and Wolfe [1960] was inspired by the previous work by Ford Jr and Fulkerson [1958] on the decomposition of LPs having a certain multi-commodity network flow structure. Dantzig and Wolfe proposed solving the Master LP using the modified RSA. Some authors attribute the Column Generation Algorithm, where RMLPs are solved by a black-box LP solver, to Cheney and Goldstein [1959] and Kelley, Jr [1960], as their cutting plane algorithm can be used for solving the dual of a Master LP (see Section 5.3.2). Yet, those original works were intended to solve other kinds of problems not related to Dantzig-Wolfe reformulation. The expression *Column Generating Procedure* already appears in Gilmore and Gomory [1963], but *Column Generation Algorithm* become more popular after Appelgren [1969].

2.3. Ford Jr and Fulkerson [1958] proposed their early CG algorithm for the following maximum multi-commodity flow problem: given a graph $G = (V, A)$ with arc capacities u_a, $a \in A$ and K commodities, commodity $k \in [K]$ having

S^k and T^k as their source and sink sets, respectively; find a maximum total flow. They formulated that problem as:

$$\max \quad \sum_{k \in K} \sum_{\boldsymbol{p} \in \Omega^k} \lambda_{\boldsymbol{p}}^k \tag{2.31a}$$

$$\text{s.t.} \quad \sum_{k \in K} \sum_{\boldsymbol{p} \in \Omega^k} p_a \, \lambda_{\boldsymbol{p}}^k \leq u_a \qquad a \in A \tag{2.31b}$$

$$\boldsymbol{\lambda} \geq 0. \tag{2.31c}$$

where Ω^k is the set of all paths that start at a vertex in S^k and end at a vertex in T^k. They realized that, in spite of the huge number of variables, those LPs could be solved by the RSA, using the shortest path algorithm to perform the pricing step.

Curiously, their main motivation for the CG approach was not reducing CPU time. They noticed that large maximum multi-commodity flow problems could not be handled as standard LPs (2.24) by the simplex method since their base matrices would not even fit in the main memory of the computers then available. They mentioned a hypothetical instance with 50 vertices, 100 arcs, and 20 commodities. In that case, there would be 1000 constraints in (2.24b) and 100 constraints in (2.24c), so a simplex basis would have dimension 1100×1100. On the other hand, the proposed LP with path variables would have bases of dimension 100×100. Ford Jr and Fulkerson [1958] finishes with: *"Except for hand computation for a few small problems, we have no computational experience with the proposed method. Whether the method is practicable ... is a question that can only be settled by experimentation."*

2.4. Dantzig and Wolfe [1960] also does not provide computational experiments on their proposed general LP decomposition method, only pointing out the cases where it *"holds promise for the efficient computation of large-scale systems"*. The article also discusses the economic and game-theoretical implications of the DW decomposition: it shows that it is possible to obtain global optimal decisions in a system where a central planning agency (the Master LP) only communicates with a set of autonomous agents (the subproblems) by iteratively setting prices for shared resources (the dual variables) and receiving

optimal production offers (the columns).

2.5. Farkas Pricing. An alternative to introducing artificial variables with very large costs to an RMLP, avoiding the issues of choosing those costs (see Note 1.5), is the so-called *Farkas pricing*. It is based on Farkas' Lemma (a demonstration can be found in Schrijver [1986]), which affirms that exactly one of the following statements is true: i) there exists a feasible solution $\boldsymbol{\lambda}$ to an RMLP in format (2.8), and ii) there exist multipliers $\boldsymbol{u} \in \mathbb{R}^{1\times m}$ and $v \in \mathbb{R}_+$ such that

$$\boldsymbol{u}\boldsymbol{A}\boldsymbol{q} + v \leq 0,\ \forall \boldsymbol{q} \in S,\ \text{and}\ \boldsymbol{u}\boldsymbol{b} + v > 0. \tag{2.32}$$

Farkas' Lemma, despite being called a "lemma", is a deep result that can even be used for deriving LP strong duality (Theorem 1.3). If an RMLP (2.8) is infeasible, most LP solvers can provide Farkas multipliers $(\boldsymbol{u}^*, v^*)$ satisfying (2.32). To invalidate that certificate of infeasibility, one needs to enlarge S with $\boldsymbol{q}' \in Q \setminus S$, such that $\boldsymbol{u}^*\boldsymbol{A}\boldsymbol{q}' + v^* > 0$. Thus, the Farkas pricing problem is:

$$\begin{aligned} \overline{c}^* \ = \ \min \ & -\boldsymbol{u}^*\boldsymbol{A}\boldsymbol{x} - v^* && \text{(2.33a)} \\ \text{s.t.} \quad & \boldsymbol{x} \in P. && \text{(2.33b)} \end{aligned}$$

If $\overline{c}^* < 0$, the variable $\lambda_{\boldsymbol{x}^*}$ corresponding to the optimal solution $\boldsymbol{x}^*$ is added to the RMLP, otherwise, it is proved that the full Master LP is infeasible.

Gamrath [2010] gives a more detailed explanation of the Farkas pricing. We remark that if an RMLP contains inequalities, it is unnecessary to convert them to equalities to get a vector of Farkas multipliers in case of infeasibility. The only difference is that the multipliers corresponding to $\geq$ constraints should be non-negative, while those corresponding to $\leq$ constraints should be non-positive. In any case, the Farkas multipliers can be used for deriving, from the RMLP constraints, a single inequality that obviously does not have solutions. So, they work as a certificate of infeasibility. For example, consider

the following possible RMLP:

$$\begin{array}{rlrcrcrcl} z_{\text{RM}} = & \min & 4\lambda_1 & - & \lambda_2 & + & 3\lambda_3 & & \\ & \text{s.t.} & 6\lambda_1 & & & - & \lambda_3 & = & 5 \\ & & 3\lambda_1 & + & 2\lambda_2 & + & 2\lambda_3 & \geq & 4 \\ & & \lambda_1 & + & \lambda_2 & + & \lambda_3 & \leq & 1 \\ & & & & & & \boldsymbol{\lambda} & \geq & \mathbf{0}. \end{array}$$

A vector of Farkas multipliers that certificates its infeasibility is $(\boldsymbol{u}^*, v^*) = (\,-1 \;\; 7 \;\; -15\,)$. Applying the multipliers to each constraint and summing them, we get:

$$\begin{array}{rcl} -1 \times (& 6\lambda_1 - \lambda_3 = 5 &) \\ 7 \times (& 3\lambda_1 + 2\lambda_2 + 2\lambda_3 \geq 4 &) \\ -15 \times (& \lambda_1 + \lambda_2 + \lambda_3 \leq 1 &) \\ \hline & 0\lambda_1 - \lambda_2 + 0\lambda_3 \geq 8. & \end{array}$$

As $\boldsymbol{\lambda} \geq \mathbf{0}$, the LHS of the derived inequality can never be positive, so, it is clear that the RMLP is indeed infeasible. So, $(\boldsymbol{u}^*, v^*)$ can be used for defining the next Farkas pricing subproblem (2.33), which will either generate a new column that will invalidate $(\boldsymbol{u}^*, v^*)$ as a certificate of infeasibility or prove that the full MLP is also infeasible.

This elegant initialization scheme is not much adopted in practice. One of the reasons is that, as will be seen in Chapter 7, artificial variables may also be used for improving CG convergence.

2.6. Recovering the original LP dual solution. Suppose that we perform a DW reformulation over an original LP and solve the resulting MLP using the CGA. As already shown, the optimal MLP primal solution can be converted into an optimal primal solution of the original LP. What is less known is that the CGA readily provides an optimal dual solution for the original LP.

Theorem 2.12: *Let OLP* $\equiv \min z = \boldsymbol{cx}$ *subject to* $\boldsymbol{Ax} = \boldsymbol{b}$, $\boldsymbol{Dx} \geq \boldsymbol{d}, \boldsymbol{x} \geq \mathbf{0}$. *Assume that OLP has optimal solutions with value* z^* *and consider the MLP obtained from it by a DW reformulation that keeps* $\boldsymbol{Ax} = \boldsymbol{b}$ *in the master. Let* $(\boldsymbol{\pi}^*, \nu^*)$ *be the optimal dual solution of that MLP found by the CGA and let*

$\boldsymbol{\rho}^*$ be the optimal dual solution of the last subproblem LP (the one that yields $\overline{c}^* = 0$). Then, $(\boldsymbol{\pi}^*, \boldsymbol{\rho}^*)$ is an optimal dual solution to OLP.

Proof. The dual of OLP is DOLP $\equiv \max \boldsymbol{\pi b} + \boldsymbol{\rho d}$ subject to $\boldsymbol{\pi A} + \boldsymbol{\rho D} \leq \boldsymbol{c}$, $\boldsymbol{\rho} \geq \boldsymbol{0}$. The dual of the last subproblem LP is DSLP $\equiv \max \overline{c} = \boldsymbol{\rho d} - \nu^*$ subject to $\boldsymbol{\rho D} \leq \boldsymbol{c} - \boldsymbol{\pi}^* \boldsymbol{A}$, $\boldsymbol{\rho} \geq \boldsymbol{0}$. Therefore, $\boldsymbol{\pi}^* \boldsymbol{A} + \boldsymbol{\rho}^* \boldsymbol{D} \leq \boldsymbol{c}$ and $\boldsymbol{\rho}^* \geq \boldsymbol{0}$. So, $(\boldsymbol{\pi}^*, \boldsymbol{\rho}^*)$ is a feasible solution of DOLP. As $(\boldsymbol{\pi}^*, \nu^*)$ is an optimal MLP dual solution, $z^* = \boldsymbol{\pi}^* \boldsymbol{b} + \nu^*$. As $\overline{c}^* = 0$, $\boldsymbol{\rho}^* \boldsymbol{d} = \nu^*$. Therefore, $\boldsymbol{\pi}^* \boldsymbol{b} + \boldsymbol{\rho}^* \boldsymbol{d} = z^*$ and $(\boldsymbol{\pi}^*, \boldsymbol{\rho}^*)$ is an optimal solution of DOLP. □

The above result remains valid regardless of the actual sense ($\leq$, $=$, or $\geq$) of the constraints that are kept in the master or go to the subproblem. However, its generalization for the case of multiple subproblems in left as Exercise E 2.19.

2.7. Non-subproblem variables. Consider an LP in the following format:

$$\min \; z = \boldsymbol{cx} \; + \; \boldsymbol{dy} \tag{2.34a}$$

$$\text{s.t.} \quad \boldsymbol{Ax} \; + \; \boldsymbol{Dy} = \boldsymbol{b} \tag{2.34b}$$

$$\boldsymbol{x} \; \in P, \tag{2.34c}$$

where P is a bounded polyhedron represented by a set of linear constraints only over the x variables. Performing the DW decomposition of (2.34) that keeps (2.34b) in the master, the resulting Master LP is:

$$z_{\text{M}} \; = \; \min \; \sum_{\boldsymbol{q} \in Q} (\boldsymbol{cq}) \lambda_{\boldsymbol{q}} + \boldsymbol{dy} \tag{2.35a}$$

$$\text{s.t.} \; \sum_{\boldsymbol{q} \in Q} (\boldsymbol{Aq}) \lambda_{\boldsymbol{q}} + \boldsymbol{Dy} = \boldsymbol{b} \tag{2.35b}$$

$$\sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} = 1 \tag{2.35c}$$

$$\boldsymbol{\lambda} \geq \boldsymbol{0}, \tag{2.35d}$$

where $Q = Ext(P)$. This means that there are cases where part of the original variables remain in the MLP. The pricing subproblem for generating the λ variables still has the format (2.7a).

This case illustrates the following taxonomy of the variables involved in a DW reformulation, depicted in Figure 2.4. In the first level, we have the *original variables* and the *Master variables*. The original variables are either *subproblem variables*, those that are used in the definition of P and also appear in the pricing subproblems (notated as the x variables in (2.34)), or *non-subproblem variables* (notated as y variables in (2.34)). The Master variables are either *generated variables*, those that are associated with points in P and may be dynamically generated during the CGA (the λ variables in (2.35)), or *non-subproblem variables* (the same y variables).

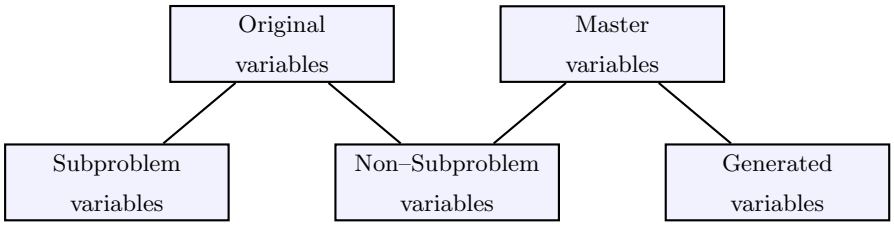


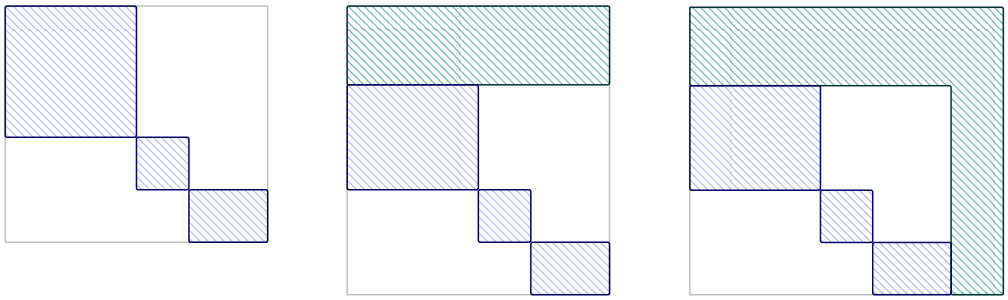
Figure 2.4: Taxonomy of the variables involved in a DW reformulation

2.8. Block-diagonal and block-angular structures. Some authors refer to the multiple subproblems case as the case where the subproblem has a *block-diagonal structure*. In fact, the coefficients in the left-hand side of the linear equations defining the U independent subproblems in (2.11c) correspond to a block-diagonal matrix, a structure schematized in Figure 2.5a. In Example (2.21), that matrix (not including the variable non-negativity constraints) has the following non-zero elements:

$$\begin{pmatrix} 1 & 1 & & & & & \\ 3 & 1 & & & & & \\ & & 1 & 1 & & & \\ & & & & 1 & 1 & \\ & & & & 3 & 1 & \end{pmatrix}.$$

Equivalently, some authors refer to the multiple subproblems case as the case where the full matrix formed by the coefficients in the left-hand side of (2.11b)-

(2.11c) has a *block-angular structure*, as depicted in Figure 2.5b.



(a) Block-diagonal (b) Block-angular (c) Double-block-angular

Figure 2.5: Schematic representation of some matrix structures. The dashed areas indicate where non-zero coefficients may appear.

2.9. Double-block-angular structures. Consider an LP in the following format:

$$\min\ z = \boldsymbol{c}^1\boldsymbol{x}^1 + \cdots + \boldsymbol{c}^U\boldsymbol{x}^U + \boldsymbol{dy} \tag{2.36a}$$

$$\text{s.t.} \quad \boldsymbol{A}^1\boldsymbol{x}^1 + \cdots + \boldsymbol{A}^U\boldsymbol{x}^U + \boldsymbol{Dy} = \boldsymbol{b} \tag{2.36b}$$

$$(\boldsymbol{x}^u, \boldsymbol{y}) \in P^u \qquad u \in [U]. \tag{2.36c}$$

In principle, a DW reformulation keeping (2.36b) in the Master would not lead to a decomposable subproblem, since the y variables are shared between the U blocks. In this case, the full matrix formed by the coefficients in the left-hand side of (2.36b)-(2.36c) has a *double-block-angular structure*, a.k.a. *double-bordered block-diagonal structure* (the rows in (2.36b) and the columns corresponding to the y variables define the "double-borders"), as schematized in Figure 2.5c.

However, assume that there are not so many y variables. A trick for obtaining a decomposition into U blocks (similar to a technique known as Lagrangian decomposition [Guignard and Kim, 1987]) is to introduce $U-1$ copies of the

y variables and rewrite the LP as follows:

$$\begin{aligned}
\min\ z\ &= \boldsymbol{c}^1\boldsymbol{x}^1 + \cdots + \boldsymbol{c}^U\boldsymbol{x}^U + \boldsymbol{d}\boldsymbol{y}^1 && && \text{(2.37a)}\\
\text{s.t.}\quad & \boldsymbol{A}^1\boldsymbol{x}^1 + \cdots + \boldsymbol{A}^U\boldsymbol{x}^U + \boldsymbol{D}\boldsymbol{y}^1 = \boldsymbol{b} && && \text{(2.37b)}\\
& \boldsymbol{y}^u = \boldsymbol{y}^{u+1} && u \in [U-1] && \text{(2.37c)}\\
& (\boldsymbol{x}^u, \boldsymbol{y}^u) \in P^u && u \in [U]. && \text{(2.37d)}
\end{aligned}$$

The problem with having too many y variables in (2.36) is that the additional equalities (2.37c) in the Master LP can make the convergence very slow. An alternative idea [Poggi de Aragão and Uchoa, 2003, Bergner et al., 2015, Chen et al., 2024] is working directly with the following explicit master LP:

$$\begin{aligned}
\min\ z\ &= \boldsymbol{c}^1\boldsymbol{x}^1 + \cdots + \boldsymbol{c}^U\boldsymbol{x}^U + \boldsymbol{d}\boldsymbol{y} && && \text{(2.38a)}\\
\text{s.t.}\quad & \boldsymbol{A}^1\boldsymbol{x}^1 + \cdots + \boldsymbol{A}^U\boldsymbol{x}^U + \boldsymbol{D}\boldsymbol{y} = \boldsymbol{b} && && \text{(2.38b)}\\
& (\boldsymbol{x}^u, \boldsymbol{y}) = \sum_{\boldsymbol{q}\in Q^u} \boldsymbol{q}\theta^u_{\boldsymbol{q}} && u \in [U] && \text{(2.38c)}\\
& \sum_{\boldsymbol{q}\in Q^u} \theta^u_{\boldsymbol{q}} = 1 && u \in [U] && \text{(2.38d)}\\
& \boldsymbol{\theta} \geq \boldsymbol{0}. && && \text{(2.38e)}
\end{aligned}$$

CG Convergence for solving (2.37) may also be problematic (see Note 4.17).

2.10. Benders decomposition for LP. Consider an LP in the following format:

$$\begin{aligned}
\min\quad & \boldsymbol{c}\boldsymbol{x} + \sum_{u\in[U]} \boldsymbol{f}^u\boldsymbol{y}^u && && \text{(2.39a)}\\
\text{s.t.}\quad & \boldsymbol{A}^u\boldsymbol{x} + \boldsymbol{D}^u\boldsymbol{y}^u \geq \boldsymbol{b}^u \quad u \in [U]. && && \text{(2.39b)}
\end{aligned}$$

Note that its constraint matrix has a structure that is the transpose of the block angular structure found in (2.11). Let $\boldsymbol{\pi}^u$, $u \in [U]$, be row vectors of

dual variables associated with (2.39b). The dual of (2.39) is:

$$\max \quad \sum_{u\in[U]} \boldsymbol{\pi}^u \boldsymbol{b}^u \tag{2.40a}$$

$$\text{s.t.} \quad \sum_{u\in[U]} \boldsymbol{\pi}^u \boldsymbol{A}^u = \boldsymbol{c} \tag{2.40b}$$

$$\boldsymbol{\pi}^u \boldsymbol{D}^u = \boldsymbol{f}^u \qquad u \in [U] \tag{2.40c}$$

$$\boldsymbol{\pi}^u \geq \boldsymbol{0} \qquad u \in [U]. \tag{2.40d}$$

A DW reformulation of it that keeps (2.40b) in the master obtains:

$$z_{\text{M}} = \max \quad \sum_{u\in[U]} \sum_{\boldsymbol{q}\in Q^u} (\boldsymbol{q}^\intercal \boldsymbol{b}^u)\lambda^u_{\boldsymbol{q}} + \sum_{u\in[U]} \sum_{\boldsymbol{r}\in R} (\boldsymbol{r}^\intercal \boldsymbol{b}^u)\mu^u_{\boldsymbol{r}} \tag{2.41a}$$

$$\text{s.t.} \quad \sum_{u\in[U]} \sum_{\boldsymbol{q}\in Q} (\boldsymbol{q}^\intercal \boldsymbol{A}^u)\lambda^u_{\boldsymbol{q}} + \sum_{u\in[U]} \sum_{\boldsymbol{r}\in R} (\boldsymbol{r}^\intercal \boldsymbol{A}^u)\mu^u_{\boldsymbol{r}} = \boldsymbol{c} \tag{2.41b}$$

$$\sum_{\boldsymbol{q}\in Q^u} \lambda^u_{\boldsymbol{q}} = 1 \qquad u \in [U] \tag{2.41c}$$

$$\boldsymbol{\lambda},\ \boldsymbol{\mu} \geq \boldsymbol{0}, \tag{2.41d}$$

where $Q^u = Ext(P^u = \{\boldsymbol{\pi}^u \boldsymbol{D}^u = \boldsymbol{f}^u, \boldsymbol{\pi}^u \geq \boldsymbol{0}\})$ and $R^u = ExtRay(P^u)$. The dual of (2.41), considering $\boldsymbol{x}$ and $\boldsymbol{z}$ as column vectors of dual variables associated with (2.41b) and (2.41c), respectively, is:

$$z_{BM} = \min \quad \boldsymbol{c}\boldsymbol{x} + \sum_{u\in[U]} z_u \tag{2.42a}$$

$$\text{s.t.} \quad (\boldsymbol{q}^\intercal \boldsymbol{A}^u)\boldsymbol{x} + z_u \geq (\boldsymbol{q}^\intercal \boldsymbol{b}^u) \qquad u \in [U],\ \boldsymbol{q} \in Q^u \tag{2.42b}$$

$$(\boldsymbol{r}^\intercal \boldsymbol{A}^u)\boldsymbol{x} \geq (\boldsymbol{r}^\intercal \boldsymbol{b}^u) \qquad u \in [U],\ \boldsymbol{r} \in R^u. \tag{2.42c}$$

We have just derived the so-called Benders decomposition for LP, which is fully equivalent to a DW decomposition over the dual problem. The resulting Master Benders LP (2.42) is often expressed in the following format:

$$z_{BM} = \min \quad \boldsymbol{c}\boldsymbol{x} + \sum_{u\in[U]} z_u \tag{2.43a}$$

$$\text{s.t.} \quad z_u \geq \boldsymbol{q}^\intercal(\boldsymbol{b}^u - \boldsymbol{A}^u\boldsymbol{x}) \qquad u \in [U],\ \boldsymbol{q} \in Q^u \tag{2.43b}$$

$$\boldsymbol{r}^\intercal(\boldsymbol{b}^u - \boldsymbol{A}^u\boldsymbol{x}) \leq 0 \qquad u \in [U],\ \boldsymbol{r} \in R^u. \tag{2.43c}$$

This LP may be very large, so it has to be solved by *row generation*. A Restricted Master Benders LP (RMBLP) only contains a small subset of its constraints and may include some artificial constraints to prevent unboundness. Given an optimal solution $(\boldsymbol{x}^*, \boldsymbol{z}^*)$ to a RMBLP, for each $u \in [U]$, the following subproblem LP is solved:

$$\zeta^{u*} = \max \quad \boldsymbol{\pi}^u(\boldsymbol{b}^u - \boldsymbol{A}^u\boldsymbol{x}^*) \tag{2.44a}$$

$$\text{s.t.} \quad \boldsymbol{\pi}^u\boldsymbol{D}^u = \boldsymbol{f}^u \tag{2.44b}$$

$$\boldsymbol{\pi}^u \geq \boldsymbol{0}. \tag{2.44c}$$

If (2.44) is unbounded, the LP solving method should provide $\boldsymbol{r}^* \in ExtRay(P^u)$ such that $\boldsymbol{r}^{*\intercal}(\boldsymbol{b}^u - \boldsymbol{A}^u\boldsymbol{x}^*) > 0$. So, the corresponding constraint in (2.43c) is violated and is added to the RMBLP. In Benders' context, this is called a *feasibility cut* for the following reason. The unboundedness of (2.44) proves (by weak LP duality) that $D^u\boldsymbol{y}^u \geq \boldsymbol{b}^u - \boldsymbol{A}^u\boldsymbol{x}^*$ has no solutions. Therefore, the "tentative solution" $\boldsymbol{x}^*$ would lead to infeasibility in (2.39) and should be corrected. If $z_u^* < \zeta^{u*} < \infty$ then an optimal solution $\boldsymbol{\pi}^* = \boldsymbol{q}^* \in Ext(P^u)$ finds a violated constraint in (2.43c), which is known as an *optimality cut*. In this case, the reason is that variable z_u should carry the correct cost $\zeta^{u*} = \min f^u\boldsymbol{y}^u$ s.t. $D^u\boldsymbol{y}^u \geq \boldsymbol{b}^u - \boldsymbol{A}^u\boldsymbol{x}^*$ induced by $\boldsymbol{x}^*$ on subproblem u.

Benders decomposition greatly extends the range of applications of LP decomposition. For example, it is particularly well-suited for stochastic programming where uncertainty in the parameters (such as demand, supply, costs, etc.) is modeled using scenarios. Consider a two-stage stochastic LP where vector $\boldsymbol{x}$ represents decisions that should be made now and vector $\boldsymbol{y}$ represents additional decisions that will be made in the future, after the uncertain parameters are revealed. The idea is to consider U possible scenarios for those parameters. For each scenario $u \in [U]$, let p_u be its probability and let $\boldsymbol{y}^u$ denote the corresponding second-stage decisions. One may set an LP having the following format:

$$\min \quad \boldsymbol{c}\boldsymbol{x} + \sum_{u\in[U]} p_u\boldsymbol{f}^u\boldsymbol{y}^u \tag{2.45a}$$

$$\text{s.t.} \quad \boldsymbol{A}^u\boldsymbol{x} + \boldsymbol{D}^u\boldsymbol{y}^u \geq \boldsymbol{b}^u \qquad u \in [U], \tag{2.45b}$$

where $\boldsymbol{f}^u$, $\boldsymbol{A}^u$, $\boldsymbol{D}^u$, and $\boldsymbol{b}^u$ may depend on the parameters of scenario $u \in [U]$. The objective function (2.45a) minimizes the expected total cost over all scenarios. Note that properly sampling a few dozen uncertain parameters may already require values of U in the range of thousands. So, the resulting LPs may be very large. Happily, a Benders decomposition of (2.45) (equivalent to a DW decomposition of its dual), together with computer parallelism and sophisticated sampling, may provide practical solution methods for huge two-stage stochastic LPs (see Section 6.5 of Bertsimas and Tsitsiklis [1997]).

2.11. A beginners' implementation mistake. As mentioned in Note 1.6, modern LP solvers treat bounds over individual variables in a special way. Those bounds may be already included in the variable definition instead of being given as explicit constraints. Suppose that is known that all generated variables λ will have values between 0 and 1, perhaps because there is a convexity constraint. When including newly generated variables into the RMLP, one should *not* define them as having upper bound 1 (instead, the provided bounds should be $[0\,,\,+\infty]$). The reason is that the implicit constraints in format $\lambda_j \leq 1$ have hidden dual variables (actually, those dual values may be recovered from the provided reduced costs, see Exercise E 3.3). Those hidden dual variables have value zero in most iterations, and therefore, they are easily overlooked. However, at some seemingly random iteration, one of those hidden dual variables "robs" the value of another explicit dual variable, perhaps the ν variable associated with the convexity constraint. The result is a nasty bug in a CG code.

2.12. Flow decomposition as a DW decomposition. In most cases where subproblems exhibit a network flow structure, the additional use of the decomposition of flows into paths+cycles, given in Theorem 2.11, enhances the efficiency of CGAs. A comprehensive analysis and experimental results supporting this assertion can be found in Jones et al. [1993]. It might raise a question for readers: is flow decomposition a variant of DW decomposition, or is it a separate principle altogether? The answer aligns with the former. Below, we provide an interpretation of flow decomposition as a DW decomposition.

We start by noticing that a network flow problem with multiple sources and/or sinks can be transformed into an equivalent problem with a single sink and a single source. Let S and T be the sets of source and sink vertices respectively. Define a modified graph $G' = (V', A')$, where $V' = V \cup \{s, t\}$ and $A' = A \cup \{(s, i) : i \in S\} \cup \{(i, t) : i \in T\}$. The modified demands d' are zero, except that $d'_s = \sum_{i \in S} d_i$ and $d'_t = \sum_{i \in T} d_i$. The original arc costs are not changed, new arcs have zero cost. The original arc capacities are not changed, but an arc (s, i), $i \in S$ has capacity $-d_i$ and an arc (i, t), $i \in T$ has capacity d_i.

Consider a network flow problem over a graph G' with a single source s and single sink t (with demand d'_t) and its corresponding LP (2.23). Apply a DW reformulation that keeps constraints $x_a \leq u_a$, $a \in A'$, in the master and let the remaining constraints define polyhedron P. It can be proved (left as an exercise) that $Q = Ext(P) = \{d'_t \boldsymbol{\chi}(B) \,|\, B \subseteq A'$ is an elementary $s-t$ path in $G'\}$ and that $ExtRay(P) = \{\boldsymbol{\chi}(C)/|C| \,|\, C \subseteq A'$ is an elementary cycle in $G'\}$. Consider the sets Ω and Θ defined in the enunciate of Theorem 2.11. Sets $ExtRay(P)$ and Θ correspond to the same extreme rays, so the latter can be used for defining the resulting MLP:

$$\min \quad \sum_{\boldsymbol{p} \in Q} \left(\sum_{a \in A} c_a p_a \right) \lambda_{\boldsymbol{p}} + \sum_{\boldsymbol{r} \in \Theta} \left(\sum_{a \in A} c_a r_a \right) \mu_{\boldsymbol{r}} \tag{2.46a}$$

$$\text{s.t.} \quad \sum_{\boldsymbol{p} \in Q} p_a \lambda_{\boldsymbol{p}} + \sum_{\boldsymbol{r} \in \Theta} r_a \mu_{\boldsymbol{r}} \leq u_a \qquad a \in A \tag{2.46b}$$

$$\sum_{\boldsymbol{p} \in Q} \lambda_{\boldsymbol{p}} = 1 \tag{2.46c}$$

$$(\, \boldsymbol{\lambda},\ \boldsymbol{\mu} \,) \geq 0. \tag{2.46d}$$

Scaling the λ variables we obtain an equivalent MLP defined over sets Ω and

Θ:

$$\min \quad \sum_{\boldsymbol{p}\in\Omega}\left(\sum_{a\in A} c_a p_a\right)\lambda_{\boldsymbol{p}} + \sum_{\boldsymbol{r}\in\Theta}\left(\sum_{a\in A} c_a r_a\right)\mu_{\boldsymbol{r}} \tag{2.47a}$$

$$\text{s.t.} \quad \sum_{\boldsymbol{p}\in\Omega} p_a\lambda_{\boldsymbol{p}} + \sum_{\boldsymbol{r}\in\Theta} r_a\mu_{\boldsymbol{r}} \le u_a \qquad a \in A \tag{2.47b}$$

$$\sum_{\boldsymbol{p}\in\Omega} \lambda_{\boldsymbol{p}} = d'_t \tag{2.47c}$$

$$(\,\boldsymbol{\lambda},\ \boldsymbol{\mu}\,) \ge 0. \tag{2.47d}$$

Then, the path+cyle decomposition theorem can be viewed as the translation of those MLP solutions into the original arc flow variables:

$$\boldsymbol{x} = \sum_{\boldsymbol{p}\in\Omega} \boldsymbol{p}\lambda_{\boldsymbol{p}} + \sum_{\boldsymbol{r}\in\Theta} \boldsymbol{r}\mu_{\boldsymbol{r}}.$$

2.13. Is it worthy to solve LPs by CG using an LP solver as pricer? Several methods that applied DW decomposition and CG to certain LPs with block-angular structure (but no particular structure in the subproblems, so pricing still has to be performed by an LP solver) were proposed during the 1960s and 1970s, as surveyed in Ho and Loute [1981]. Those authors asserted that the overall results so far had not been very encouraging when compared with applying the Simplex method directly to the original LP. One of the reasons was that the Simplex implementations at the time were already good at handling the sparse basis matrices corresponding to LPs with block-diagonal substructures. On the other hand, the CGA often suffered from slow convergence. Ho and Loute [1981] then proposed DECOMPSX, a sophisticated and general implementation that did better than the existing LP codes on some problems that could be decomposed into many subproblems.

The situation has not changed much in the last 40 years. While the CGA implementations improved a lot, so did the Simplex (and later, interior-point) LP implementations. We provide some examples of papers where a CGA implementation could beat the best available general LP solvers on particular LP problems.

Rosen and Maier [1990] and Entriken [1996] showed that DW decomposition

together with parallelization could outperform simplex algorithm implementations of those times. Tebboth [2001] did a comprehensive implementation of DW decomposition, in which many known and new computational strategies were used. When using a computer with seven processors, in some instances, it could obtain a speed-up of five in comparison with the Xpress-MP commercial LP solver. Rios [2013] presented another parallel implementation of the DW decomposition, which largely outperformed CPLEX solver in some instances. The standalone C code based on the GLPK LP solver is publicly available together with the paper. Rios and Ross [2010], Wei et al. [2013] discussed the successful application of parallel DW decomposition to air traffic management problems formulated as linear programs. Finally, we can cite a relatively recent review Chung [2011] on DW decomposition for linear programs, which also discusses computational issues.

To our opinion, the following quotation from Tebboth [2001] still summarizes well the place of DW decomposition as a general LP solving tool. *"Dantzig-Wolfe decomposition will not rival mainstream techniques as an optimization method for all LP problems. But it does have some niche areas of application: certain large-scale classes of primal block angular structured problems, and in particular where the context demands rapid results using parallel optimization or near-optimal solutions with guaranteed quality. For such problems, decomposition offers the capability to solve them in a much quicker time than would otherwise be possible, providing evidence that, no matter what increases in raw computing power and resources occur in the future, there will be a class of applications, perhaps of ever-increasing size and scope, for which decomposition is the solution method of choice."*

2.14. LPs that have to be solved by CG. In the probabilistic logic created by Nilsson [1986] for use in artificial intelligence expert systems, facts and clauses have probabilities of being true. Checking whether a given set of probabilities is consistent or finding the smallest changes to those probabilities to make them consistent can be done by solving a huge LP, with the number of variables growing exponentially with the number of basic facts.

Jaumard et al. [1991] proposed a very effective CGA for solving these LPs. However, the pricing subproblem is not defined by another LP. Instead, it cor-

responds to a Weighted Max-SAT, an $\mathcal{NP}$-hard combinatorial problem. This type of subproblem is typical when using CG to solve integer programs, as will be seen in Chapter 4. Despite that, this interesting application is still classified as column generation for LPs, since it is being used to solve a problem defined over continuous variables, and no branching is necessary. It is an example of a case where *an LP must be solved by CG because there is no known original compact LP*.

Exercises

E 2.1. Use the CGA for solving the MLP (2.5). RMLPs and the subproblems may be solved by any LP solver. Convert the primal solution of each RMLP into points in the x space and locate them in the cartesian plane shown in Figure 2.1. Draw a cartesian plane depicting the set of feasible dual MLP solutions. Locate the dual solution of each RMLP.

E 2.2. Consider the following LP:

$$\begin{array}{lrcrcl} \max z = & 7x_1 & + & 3x_2 & & \\ \text{s.t.} & x_1 & + & x_2 & \leq & 4 \\ & 2x_1 & & & \leq & 5 \\ & x_1 & + & 4x_2 & \geq & 4 \\ & -x_1 & + & x_2 & \leq & 3 \\ & x_1 & - & 2x_2 & \leq & 2 \\ & & & \boldsymbol{x} & \geq & \mathbf{0}. \end{array}$$

Its set of feasible solutions is depicted in dark area in Figure 2.6. Consider a DW reformulation that keeps the first three constraints in the master. The resulting unbounded polyhedron P is shown in light blue in that picture. Obtain sets $Ext(P)$ and $ExtRay(P)$ and write the resulting *explicit reformulated LP*. Solve that LP.

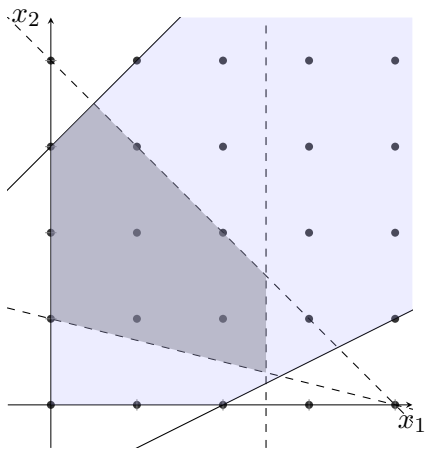


Figure 2.6: DW decomposition of the LP in E 2.2

E 2.3. Use the CGA for solving the MLP from the previous exercise. Convert the primal solution of each RMLP into points in the x space and locate them in the cartesian plane.

E 2.4. Consider the following LP:

$$
\begin{array}{lrcrcrcrcr}
\max z = & x_1 & + & 7x_2 & + & 2x_3 & + & 3x_4 & & \\
\text{s.t.} & 4x_1 & + & 2x_2 & + & 2x_3 & + & 3x_4 & \leq & 30 \\
 & x_1 & + & x_2 & + & x_3 & + & 2x_4 & = & 15 \\
 & 2x_1 & + & 3x_2 & - & 7x_3 & - & 6x_4 & = & 0 \\
 & & - & x_2 & + & 2x_3 & & & \leq & 0 \\
 & & & x_2 & - & 2x_3 & - & 2x_4 & \leq & 0 \\
 & & - & x_2 & + & 3x_3 & + & 2x_4 & \leq & 2 \\
 & & & & & & & \boldsymbol{x} & \geq & \mathbf{0}.
\end{array}
$$

Let P be the polyhedron defined by the 3rd, 4th, 5th, and 6th constraints, plus the variable non-negativities, $Ext(P) = \{ (0\ 0\ 0\ 0), (3\ 0\ 0\ 1), (1\ 4\ 2\ 0) \}$ and $ExtRay(P) = \{ (0\ 2/3\ 0\ 1/3) \}$. Write the complete MLP obtained by a DW reformulation that keeps the first two constraints in the master. Solve that MLP and then convert its primal solution into a solution of the original LP.

E 2.5. Use the DW reformulation (by keeping the first two constraints in the master) and the CGA to solve the original LP in the previous exercise. Remember that on maximization problems the artificial variables should have large negative costs. Get optimal primal and dual (see Note 2.6) solutions for the original LP.

E 2.6. Consider the following LP:

$$
\begin{array}{lrcrcrcrcrcr}
\min z = & -x_1 & + & 3x_2 & - & 6x_3 & - & 7x_4 & + & 9x_5 & & \\
\text{s.t.} & x_1 & + & 2x_2 & & & - & x_4 & + & 3x_5 & = & 36 \\
 & & - & x_2 & + & 2x_3 & & & & & \leq & 6 \\
 & x_1 & + & x_2 & + & x_3 & & & & & \leq & 10 \\
 & & & & & & & 2x_4 & + & x_5 & \leq & 22 \\
 & & & & & & & & & x_5 & \leq & 10 \\
 & & & & & & & 3x_4 & + & x_5 & \leq & 30 \\
 & & & & & & & & & \boldsymbol{x} & \geq & \mathbf{0}.
\end{array}
$$

By performing a DW reformulation keeping only the first constraint in the master, the subproblem decomposes into two subproblems, defined by polyhedra $P^1 = Conv(\{(0\ 0\ 0), (0\ 0\ 3), (10\ 0\ 0), (0\ 10\ 0), (7\ 0\ 3), (0\ 14/3\ 16/3)\})$ and $P^2 = Conv(\{(0\ 0), (10\ 0), (0\ 10), (6\ 10), (8\ 6)\})$. Write the complete MLP. Solve it and convert its solution into a solution of the original LP.

E 2.7. Use the DW reformulation (by keeping the first constraint in the master) and the CGA to solve the original LP in the previous exercise.

E 2.8. Consider the following LP:

$$\begin{array}{lrcrcrcrcrcrcl}
\min z = & -9x_1 & + & 2x_2 & + & 6x_3 & + & 7x_4 & - & 9x_5 & + & 2x_6 & & \\
\text{s.t.} & 3x_1 & + & 2x_2 & + & x_3 & + & x_4 & + & 3x_5 & + & 2x_6 & = & 50 \\
 & & & 6x_2 & + & x_3 & & & & & + & 6x_6 & \leq & 20 \\
 & 4x_1 & + & 5x_2 & & & & & & & & & = & 12 \\
 & & & & & x_3 & - & x_4 & & & & & \geq & 4 \\
 & & & & & & & & & 4x_5 & + & 5x_6 & = & 12 \\
 & & & & & & & & & & & \boldsymbol{x} & \geq & \mathbf{0}.
\end{array}$$

By performing a DW reformulation keeping only the first two constraints in the master, the subproblem decomposes into three subproblems. The first and the third subproblems are identical and can be defined by polyhedron $P^1 = Conv(\{(3\ 0), (0\ 5/12)\})$; while the second subproblem corresponds to $P^2 = Conv(\{(4\ 0)\}) + Cone(\{(1\ 0), (1/2\ 1/2)\})$. Write the complete ML and solve it. Convert its optimal solution into an optimal solution of the original LP.

E 2.9. Use the DW reformulation (by keeping the first two constraints in the master) and the CGA to solve the original LP in the previous exercise.

E 2.10. Consider the following LP:

$$\begin{array}{lrcrcrcrcrcrcl}
\max z = & & & 4x_2 & + & 5x_3 & + & x_4 & + & 4x_5 & - & x_6 & & \\
\text{s.t.} & x_1 & + & 2x_2 & & & - & x_4 & + & 3x_5 & + & x_6 & = & 32 \\
 & 3x_1 & & & + & 2x_3 & + & 2x_4 & + & 3x_5 & - & x_6 & \geq & 13 \\
 & 6x_1 & + & 8x_2 & + & 9x_3 & & & & & & & \leq & 24 \\
 & & & & & & & 10x_4 & - & 7x_5 & & & \leq & 5 \\
 & & & & & & - & x_4 & + & 2x_5 & & & \leq & 6 \\
 & & & & & & & & & & & \boldsymbol{x} & \geq & \mathbf{0}.
\end{array}$$

By performing a DW reformulation keeping only the first two constraints in the master, the subproblem decomposes into two subproblems, defined by polyhedra $P^1 = Conv(\{(0\ 0\ 0), (4\ 0\ 0), (0\ 3\ 0), (0\ 0\ 8/3)\})$ and $P^2 = Conv(\{(0\ 0), (0\ 3), (1/2\ 0), (4\ 5)\})$. Note that variable x_6 does

not appear in any subproblem. Therefore, it is an example of an original variable that should remain in the MLP (see Note 2.7). Write that MLP and solve it.

E 2.11. Use the DW reformulation (by keeping the first two constraints in the master) and the CGA to solve the original LP in the previous exercise.

E 2.12. Consider the LP from Exercise E 2.10, but modified in such a way that variable x_6 has coefficient -1 in the 3rd constraint and coefficient $1/10$ in the 4th constraint. Now, a DW reformulation keeping the first two constraints in the master would lead to a single subproblem, which is not desirable. Apply the trick described in Note 2.9, creating a copy of x_6 to obtain an original LP decomposable into two subproblems and solve that resulting LP using the CGA. Then, solve the same decomposition into two subproblems using the CGA over an Explicit Master LP of format (2.38).

E 2.13. Use the CGA to solve the LP from Exercises E 2.4 and E 2.5 again, but this time without using artificial variables. Use Farkas' pricing (Note 2.5) until a first feasible RMLP is obtained.

E 2.14. Consider the following LP:

$$
\begin{array}{lrcrcrcrcrcr}
\min z = & 4x & + & 2y_1 & + & 5y_2 & + & 8y_3 & + & 6y_4 & & \\
\text{s.t.} & 2x & - & y_1 & + & 2y_2 & & & & & \geq & 7 \\
& -x & + & y_1 & + & y_2 & & & & & = & 7 \\
& x & & & & & + & 2y_3 & + & y_4 & \geq & 9 \\
& 2x & & & & & & & + & y_4 & = & 10 \\
& & & & & & & & & (x, \boldsymbol{y}) & \geq & \mathbf{0}.
\end{array}
$$

Use the CGA to solve the MLP obtained by a DW reformulation of its *dual LP* that keeps the dual constraint corresponding to the variable x in the master (so, the subproblem decomposes into two subproblems), as shown in Note 2.10. Interpret that CG as a method that solves a Benders

decomposition of the primal LP, showing the generated feasibility/optimality cuts.

E 2.15. Consider the following two-stage stochastic problem. A factory manufactures products A and B. Each unit of A requires 3 units of resource R1 and 2 units of resource R2. Each unit of B requires 1 unit of R1 and 2 units of R2. The factory is planning for the next year, for which six parameters are uncertain: the cost of each resource, and both the selling price and demand for each product. However, it is possible to buy resources now, when R1 costs \$2/unit and R2 costs \$4/unit. The first-stage decision variables x_{R1} and x_{R2} indicate how many units are bought now. There are $U = 4$ equally probable next-year scenarios. For each scenario $u \in [U]$, the second-stage decision variables y^u_{R1}, y^u_{R2}, y^u_A, and y^u_B indicate how many units of each resource would be bought and how many units of each product would be manufactured. Write the problem of maximizing the expected profit as an LP in format (2.45), taking the uncertain parameters from the following table:

	Cost (\$)		Price (\$)		Demand	
Scen.	**R1**	**R2**	**A**	**B**	**A**	**B**
1	3	5	21	15	50	64
2	5	6	10	12	30	50
3	4	5	22	26	75	45
4	7	6	25	15	55	80

Solve that LP using Benders decomposition.

E 2.16. Generalize Theorem 2.6 for the unbounded subproblem case. In other words, prove that at any iteration of the CGA for solving (2.22), if the subproblem solution value is bounded then $z_{\mathrm{RM}} + \overline{c}^*$ is a lower bound on its optimal cost z_{M}.

E 2.17. Generalize Theorem 2.8 for the case where the polyhedra defining the subproblems may be unbounded.

E 2.18. Open exercise. Suppose that an original LP is solved by the standard RSA. In each of its intermediate iterations, is it possible to obtain a lower bound on the optimal solution value akin to those obtained when the reformulated LP is solved by the CGA? In which situations? In other words, discuss particular cases when it is possible to use the minimum reduced cost value in formulas for obtaining valid lower bounds.

E 2.19. Generalize Theorem 2.12 for the case where the DW reformulation leads to multiple subproblems and groups of identical subproblems are aggregated. In order words, prove that an optimal dual solution to original LP (2.11) can be directly obtained from the solution of MLP (2.15) by the CGA.

E 2.20. Apply the generalization of Exercise E 2.19 to obtain the dual solution of the original LP in Exercises E 2.7, E 2.9 and E 2.11.

E 2.21. Solve using the CGA by decomposition into flows the MNFP instance depicted in Figure 2.7, with the following costs:

		arc cost (c_a^k)						vertex demand (d_i^k)			
		AB	**AC**	**BC**	**BD**	**CD**	**DA**	**A**	**B**	**C**	**D**
k	**1**	2	6	4	-1	3	-3	-2	0	-2	4
	2	-3	4	5	1	5	2	-3	0	1	2

E 2.22. Solve the MNFP from the previous exercise by a GCA using decomposition into paths and cycles.

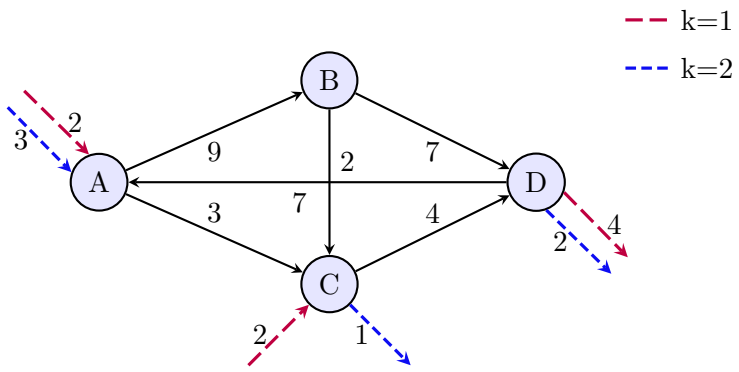


Figure 2.7: Multi-Commodity Flow Problem

E 2.23. Project Exercise. Implement (perhaps using an existing framework) a CGA for solving multi-commodity network flows. Compare the variants in which the columns correspond to (1) complete bounded flows; (2) paths + cycles. Compare with solving the original LP (2.24) using simplex and interior point methods. Include tests on big instances, in particular, instances where K is very large.

Chapter 3

Integer Programming Review

This chapter is not intended to work as a first exposition to integer programming. Very good introductory books include the classic Wolsey [1998] and its recently updated second edition [Wolsey, 2020], and also Chen et al. [2010] and Conforti et al. [2014]. More advanced books include Schrijver [1986] and Nemhauser and Wolsey [1988].

Integer programming is much more difficult than linear programming since it is an $\mathcal{NP}$-hard problem. Nevertheless, after more than 60 years of algorithmic progress, formulating other $\mathcal{NP}$-hard problems as integer programs often provides the best known way of solving them to optimality. Branch-and-bound and branch-and-cut are the most standard algorithms in integer programming.

3.1. MIPs and the Branch-and-Bound Algorithm

An Integer Program (IP) is an LP plus the additional constraint that all its variables should be integer. A general IP can be represented as:

$$z_{\text{IP}} = \min \quad \boldsymbol{cx} \tag{3.1a}$$

$$\text{s.t.} \quad \boldsymbol{x} \in P \tag{3.1b}$$

$$\boldsymbol{x} \in \mathbb{Z}^n, \tag{3.1c}$$

where P is a polyhedron defined by a set of linear constraints. The discrete set formed by the solutions of an IP can be denoted as $Int(P) = P \cap \mathbb{Z}^n$. When it is also possible to have continuous variables, we have a Mixed-Integer Program (MIP):

$$z_{\text{IP}} = \min \quad \boldsymbol{cx} + \boldsymbol{hy} \tag{3.2a}$$
$$\text{s.t.} \quad (\boldsymbol{x}, \boldsymbol{y}) \in P \tag{3.2b}$$
$$\boldsymbol{x} \in \mathbb{Z}^n, \tag{3.2c}$$

where $\boldsymbol{y} \in \mathbb{R}^p$ is the vector of variables not required to be integer. Since an IP can be viewed as the particular case of a MIP with $p = 0$, when we talk about MIPs (like in "MIP solver"), we may be also referring to IPs.

Definition 3.1: Linear relaxation. Given a MIP (3.2), its *linear relaxation* is the LP obtained by removing the integrality contraints (3.2c). Its optimal solution value is referred to as z_{LP}.

The Branch-and-Bound Algorithm (BBA), proposed by Land and Doig [1960], solves a MIP as a sequence of LPs. The first LP to be solved is its linear relaxation, which is assumed to be not unbounded (the case where the linear relaxation is unbounded is quite technical and has little practical interest; anyway, in such case, the MIP may be unbounded, infeasible or have optimal solutions). Branching is performed when an optimal LP solution is not integer. In such a case, the polyhedron defining its solution space is divided into smaller, disjoint polyhedra (each such polyhedra defines a *subproblem*) that together contain all of its integer solutions. However, bounding occurs when the LP optimization result proves that its polyhedron does not contain improving solutions. The following property will hold during the algorithm: *any improving MIP solution belongs to some active subproblem.* Of course, all MIP solutions (if they exist) should be contained in the original problem defined by P.

The BBA pseudo-code is presented in Algorithm 1. Variables $BestSol$ and UB keep the best MIP solution already found and its cost, respectively, while L is the list of active subproblems, each subproblem being defined by a set of linear constraints. List L is initialized with P and, on each iteration, a subproblem S is removed from L to be explored (while S is being explored it is still considered to be active). This means that the LP $z_S = \min \boldsymbol{cx} + \boldsymbol{hy}$ subject to $(\boldsymbol{x}, \boldsymbol{y}) \in S$ is solved and exactly one of the following cases arise:

1. If $z_S \geq UB$, it is proven that S can not contain a MIP solution with a value

better than UB (if z_{IP} is known to be integer, like in the case of a pure IP with $\boldsymbol{c}$ integer, this is true if $\lceil z_S \rceil \geq UB$). In this case, S is said to be *pruned by bound*. As an empty S yields $z_S = \infty$, infeasible subproblems are always pruned by bound.

2. If S is not pruned, one should look at its optimal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$. If $\boldsymbol{x}^*$ is integer, $BestSol$ and UB are updated. Since it is not possible that further improving MIP solutions still exist in S, that subproblem is *pruned by integrality*.

3. Otherwise, S may still contain improving MIP solutions. In order to continue searching for such solutions, we perform a *branching*. This means that an integer variable x_j such that x_j^* is fractional should be chosen. The subproblems $S \cap (x_j \leq \lfloor x_j^* \rfloor)$ and $S \cap (x_j \geq \lceil x_j^* \rceil)$, *the children of* S, are inserted into L. Note that there are no MIP solutions where $\lfloor x_j^* \rfloor < x_j < \lceil x_j^* \rceil$, so L is guaranteed to still have active subproblems containing all improving MIP solutions.

The algorithm ends when L is empty, meaning that, if $Int(P) \neq \emptyset$, $BestSol$ contains an optimal solution to the problem and $z_{\text{IP}} = UB$. The procedure in Line 11 possibly improves BBA efficiency by removing from L some children of subproblems that would have been pruned if the new UB was available when they were processed, avoiding solving unnecessary LPs. In other words, if L contains an unprocessed subproblem S' such that its parent subproblem S has $z_S \geq UB$ ($\lceil z_S \rceil \geq UB$, if z_{IP} is known to be integer), then S' can be immediately removed from L and be considered as pruned by bound. The presented BBA has two degrees of freedom, corresponding to the choice of the next active subproblem to be explored and the choice of the branching variable.

An execution of the BBA can be depicted as a rooted tree, where nodes represent subproblems and edges represent branchings. The root node corresponds to the

Algorithm 1 Branch-and-bound algorithm for solving MIP (3.2)

$UB \leftarrow \infty$
$BestSol \leftarrow$ NULL
$L \leftarrow \{P\}$
while $L \neq \emptyset$ **do**
  Remove a subproblem S from L
  Solve $z_S = \min \boldsymbol{cx} + \boldsymbol{hy}$ subject to $(\boldsymbol{x}, \boldsymbol{y}) \in S$ ▷ If infeasible, $z_S = \infty$
  if $z_S < UB$ **then** ▷ If z_{IP} integer, $\lceil z_S \rceil < UB$
    if $\boldsymbol{x}^*$ is integer **then**
      $UB \leftarrow z_S$
      $BestSol \leftarrow (\boldsymbol{x}^*, \boldsymbol{y}^*)$
      Remove from L subproblems now prunable with UB
    else
      Choose a variable x_j such that x_j^* is fractional
      Insert subproblems $S \cap (x_j \geq \lceil x_j^* \rceil)$ and $S \cap (x_j \leq \lfloor x_j^* \rfloor)$ into L
    end if
  end if
end while
return $(BestSol, UB)$

linear relaxation. For example, consider the following MIP:

$$
\begin{array}{rlrcrcrcr}
z_{\text{IP}} = & \min & & & -x_2 & + & 8y_1 & & \\
 & \text{s.t.} & x_1 & + & 2x_2 & & & \geq & 1 \\
 & & x_1 & + & 2x_2 & & & \leq & 2 \\
 & & x_1 & - & x_2 & + & 3y_1 & \geq & 0 \\
 & & x_1 & - & x_2 & - & 3y_1 & \leq & -1 \\
 & & & & & & y_1 & \geq & 0 \\
 & & & & & & \boldsymbol{x} & \in & \mathbb{Z}^2_+.
\end{array}
$$

A BBA tree solving that MIP is depicted in Figure 3.1. The choice of the next active subproblem to be explored used the *depth-first strategy* (remove from L the last inserted subproblem) and the choice of the branching variable used the *most fractional rule* (choose a fractional variable x_j such that $x_j^* - \lfloor x_j^* \rfloor$ is closer to 0.5). Subproblem $S_1 = P$ corresponds to the linear relaxation, $S_2 = S_1 \cap (x_1 \leq 0)$, $S_3 = S_2 \cap (x_2 \leq 0)$, and so on. The optimal solution with $z = 5/3$ is found in S_4, the fourth subproblem to be explored. So, S_5 with $z = 3.5$ is pruned by bound. A

BBA tree also indicates parent/child relations between its nodes. So, we may say that S_2 and S_5 are the children of S_1 or that S_2 is the parent of S_4.

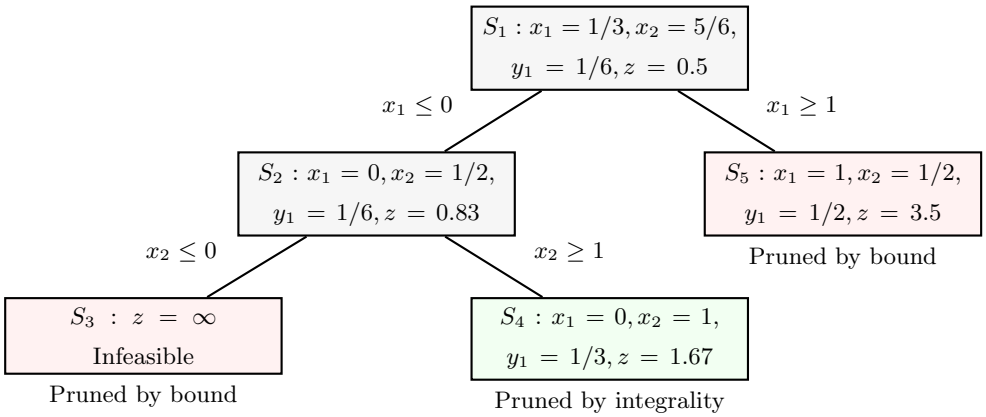


Figure 3.1: Example of a BBA tree

The practical performance of the BBA may vary widely, depending on the particular MIP being solved. In the best case, if the linear relaxation is infeasible or finds an optimal solution that is also a feasible MIP solution, the BBA only solves the root node. However, it is quite possible that a very large number of nodes have to be explored. There are several factors affecting the BBA practical performance. A key factor is the integrality gap.

Definition 3.2: Integrality gap of a MIP. Given a MIP with optimal solution value z_{IP} and its linear relaxation with optimal solution value z_{LP}, the *absolute integrality gap* is $z_{\text{IP}} - z_{\text{LP}}$, while the *relative integrality gap* (in this book often simply referred as the *gap* and expressed as a percentage) is $(z_{\text{IP}} - z_{\text{LP}})/z_{\text{IP}}$.

Grosso modo, for MIPs with the same number of integer variables, the expected number of nodes in the BBA tree grows exponentially with the gap.

There is a second definition for gap that also appears in the literature. We also state it formally in order to avoid confusion.

Definition 3.3: Optimality gap of a solution. Suppose that the best known solution to a MIP has value UB and that the best known lower bound on z_{IP} is LB. The *absolute optimality gap* and the *relative optimality gap* of *that solution* are

$UB - LB$ and $(UB - LB)/LB$, respectively.

For example, MIP solvers often provide optimality gaps during a BBA execution, calculated from the best integer solution found so far and from the current global lower bound on z_{IP} provided by the minimum z_S over the subproblems S that are parents of active subproblems. In general, optimality gaps work as guarantees of the quality of an integer solution and depend on the lower bounding mechanism used. In contrast, the integrality gap is an intrinsic feature of a MIP.

3.2. Formulating Combinatorial Optimization Problems as MIPs

An *optimization problem* can be formally defined in three parts: (1) *Instance*: the data required for fully specifying a particular case of the problem; (2) *Solutions*: the specification of what is a valid solution of an instance; and (3) *Goal*: the objective function to be minimized/maximized. When the set of solutions is discrete, we have a Combinatorial Optimization Problem (COP). In this book, we focus on a more restricted category of COPs:

Definition 3.4: Linear Combinatorial Optimization Problem. Let a problem instance contain a *base set* N with elements numbered from 1 to n and a $1 \times n$ vector $\boldsymbol{c}$, where c_j denotes the cost of the j-th element in N. Assume that a solution to the problem may be represented by a vector $\boldsymbol{x} \in \mathbb{Z}_+^n$, where x_j is the number of times that the j-th element appears in the solution (vectors $\boldsymbol{c}$ and $\boldsymbol{x}$ may also be indexed directly by the elements of N, i.e., $c_e = c_j$ and $x_e = x_j$ if e is the j-th element of N). The set of solutions for a given instance, denoted by X, is defined implicitly by a property that must be satisfied for each of its elements. Often, the elements of X are defined as the incidence vectors $\boldsymbol{\chi}(N') \in \mathbb{B}^n$ of certain sets $N' \subseteq N$. It is assumed that X is finite. Then, a *Linear Combinatorial Optimization Problem* (LCOP) is equivalent to:

$$\min\ z = \boldsymbol{cx} \tag{3.3a}$$

$$\text{s.t.} \quad \boldsymbol{x} \in X. \tag{3.3b}$$

At first sight, the above definition may seem too restrictive. Actually, the majority of the classic $\mathcal{NP}$-hard combinatorial optimization problems (like the hundreds of problems listed in the appendix of Garey and Johnson [1979]) can be cast as LCOPs in a more or less straightforward way. We provide such an example.

Definition 3.5: Vertex Cover Problem. Instance: undirected graph $G = (V, E)$, weights w_i, $i \in V$. Solutions: the vertex covers of G, i.e., the sets $V' \subseteq V$ such that for every edge $e = \{u, v\} \in E$, $e \cap V' \neq \emptyset$. Goal: Minimize $\sum_{i \in V'} w_i$.

The Vertex Cover Problem is an LCOP: by considering $N = V$, $\boldsymbol{c} = \boldsymbol{w}$, and $X = \{\boldsymbol{\chi}(V') : V' \text{ is a vertex cover of G}\}$, it fits in Definition 3.4.

We can formulate and solve many LCOPs, even if they are $\mathcal{NP}$-hard, as MIPs. In fact, this is often the best known way of solving (in the sense of finding a proven optimal solution) an $\mathcal{NP}$-hard problem.

Definition 3.6: Formulation of a Set. The IP $\min \boldsymbol{cx}$ subject to $x \in P$, $\boldsymbol{x} \in \mathbb{Z}^n$ is a *formulation for a set* $X \in \mathbb{Z}^n$ if: (i) it is infeasible if and only if $X = \emptyset$, (ii) otherwise, any optimal solution for it is also an optimal solution for (3.3).

A polyhedron P such that $X = Int(P) = P \cap \mathbb{Z}^n$ always provides a formulation for X. However, Definition 3.6 also permits formulations where $Int(P) \setminus X$ is not empty but the structure of the *possible vectors* $\boldsymbol{c}$ (for example, if the LCOP definition assumes that $\boldsymbol{c} > \boldsymbol{0}$) makes sure that solutions not in X are never optimal for the IP. Finding a formulation for an individual set X is usually not interesting.

Definition 3.7: LCOP Formulation. *Formulating an LCOP* means finding a description of a family of IPs formulating the sets X corresponding to each instance of the LCOP.

For example, a Vertex Cover Problem formulation is:

$$\min \quad \sum_{i \in V} w_i \, x_i \tag{3.4a}$$

$$\text{s.t.} \quad x_u + x_v \geq 1 \qquad \{u, v\} \in E \tag{3.4b}$$

$$\boldsymbol{x} \in \mathbb{Z}_+^{|V|}, \tag{3.4c}$$

where $G = (V, E)$ and $\boldsymbol{w}$ are given by the vertex cover instance.

Formulations over the variables used in the definition of the LCOP, are referred to as *natural formulations*. However, it is also possible to formulate an LCOP using additional auxiliary variables (see Note 3.17).

Definition 3.8: Extended Formulation. The MIP $\min \boldsymbol{cx}$ s.t. $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}) \in P$, $\boldsymbol{x} \in \mathbb{Z}^n$, $\boldsymbol{y} \in \mathbb{R}^p$, $\boldsymbol{w} \in \mathbb{Z}^q$ is an *extended formulation* for a set $X \in \mathbb{Z}^n$ if: (i) it is infeasible if and only if $X = \emptyset$, (ii) otherwise, any optimal solution for it yields (after dropping its y and w parts) an optimal solution for (3.3). An extended formulation for an LCOP is a family of MIPs providing extended formulations for the sets X corresponding to its instances.

There may be an infinite number of formulations (natural or extended) for the same LCOP. Any formulation leads to MIPs that, in principle, solve the LCOP. However, the practical performance of the branch-and-bound based algorithms varies a lot depending on which formulation is used. The differences in performance can be really dramatic. There are many cases where a "bad formulation" leads to a MIP that would require centuries of CPU time to be solved (which means that, for all practical purposes, it fails to solve the instance) and a "good formulation" leads to a MIP that solves the same instance in seconds. The following concept captures a key aspect that makes formulations better or worse and provides a theoretical way of comparing alternative formulations.

Definition 3.9: Relative formulation strength. Let IP1 $\equiv \min \boldsymbol{cx}$ subject to $x \in P_1$, $\boldsymbol{x} \in \mathbb{Z}^n$ and IP2 $\equiv \min \boldsymbol{cx}$ subject to $x \in P_2$, $\boldsymbol{x} \in \mathbb{Z}^n$ be formulations for a set $X \in \mathbb{Z}^n$. We say that IP1 is *equally strong* as IP2 if $P_1 = P_2$ and IP1 is *strictly stronger* (or only *stronger*, for short) than IP2 if $P_1 \subset P_2$.

Stronger formulations lead to smaller gaps.

Theorem 3.1: *Let z_{LP1} and z_{LP2} be optimal values of the linear relaxation of formulations IP1 and IP2, respectively. If IP1 is stronger than IP2 then $z_{LP1} \geq z_{LP2}$ for all cost vectors $\boldsymbol{c}$ and $z_{LP1} > z_{LP2}$ for some cost vectors $\boldsymbol{c}$.*

Some comments about formulation strength:

- The order provided by Definition 3.9 is only partial, in the sense that it is possible to have two not equally strong formulations IP1 and IP2 such that

one is not stronger than the other. However, in that case one has just found a new formulation IP3 $\equiv \min \boldsymbol{cx}$ subject to $x \in P_1 \cap P_2$, $\boldsymbol{x} \in \mathbb{Z}^n$ that is stronger than both IP1 and IP2.

- Definition 3.9 can be generalized to include extended formulations using the concept of projection. Formulation MIP1 $\equiv \min \boldsymbol{cx}$ subject to $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}) \in P_1$, $\boldsymbol{x} \in \mathbb{Z}^n$, $\boldsymbol{y} \in \mathbb{R}^p$, $\boldsymbol{w} \in \mathbb{Z}^q$ is *stronger* than formulation IP2 $\equiv \min \boldsymbol{cx}$ subject to $x \in P_2$, $\boldsymbol{x} \in \mathbb{Z}^n$ if $Proj_{\boldsymbol{x}}(P_1) \equiv \{\boldsymbol{x} \in \mathbb{R}^n : \exists (\boldsymbol{y}, \boldsymbol{w}) \in \mathbb{R}^{p+q}$ such that $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}) \in P_1\} \subset P_2$. It is possible to compare extended formulations defined in different spaces by comparing their projections onto $\boldsymbol{x}$. See Note 3.16 for more details.

- The strongest possible formulation for a set X, a *perfect* one, has $P = Conv(X)$. In fact, that perfect formulation always has gap zero and makes the BBA finish at the root node (at least if the linear relaxation is solved by a simplex-based method, guaranteed to find an extreme optimal solution, even if alternative non-extreme optimal solutions exist). An extended formulation that projects onto $Conv(X)$ is also said to be a perfect formulation. There are good theoretical reasons for believing that we will never have a polynomially-solvable perfect formulation for an $\mathcal{NP}$-hard problem (Note 3.14).

Given alternative formulations for the same LCOP, the number of nodes in the corresponding branch-and-bound search trees is likely to depend exponentially on the formulation gaps. This means that it can be worth going to great lengths to strengthen a formulation and reduce those gaps. Yet, the formulation size, especially in the case of extended formulations, is also very important for the overall BBA performance because it directly impacts the time for solving each LP.

3.3. Cutting Planes and the Branch-and-Cut Algorithm

A standard way of strengthening a formulation is by adding cutting planes.

Definition 3.10: Valid inequality, cutting plane. A linear inequality $\boldsymbol{\alpha x} \geq \alpha_0$, $\boldsymbol{\alpha} \in \mathbb{R}^{1 \times n}$, $\alpha_0 \in \mathbb{R}$, is *valid* for a set $X \subset \mathbb{R}^n$ if it is satisfied for all points $\boldsymbol{x} \in X$.

Let $\boldsymbol{x}^*$ be an optimal fractional solution of the linear relaxation of a formulation for a set $X \subset \mathbb{Z}^n$. A *cutting plane* is a valid inequality for X such that $\boldsymbol{\alpha x}^* < \alpha_0$. In that case, we may also say that that $\boldsymbol{\alpha x} \geq \alpha_0$ *cuts* $\boldsymbol{x}^*$. It is clear that adding $\boldsymbol{\alpha x} \geq \alpha_0$ to the formulation makes it stronger.

Valid inequalities are not equally good to be used as cutting planes. The deeper the cut, i.e., the closer to $Conv(X)$, the better. The best possible cuts are those that define facets of $Conv(X)$.

Definition 3.11: Face, proper face, facet, facet-defining inequality. Let P be a polyhedron. Polyhedron $F \subseteq P$ is *face* of P if there is an inequality $\boldsymbol{\alpha x} \geq \alpha_0$ valid for P such that $F = P \cap (\boldsymbol{\alpha x} = \alpha_0)$. In that case, we may also say that $\boldsymbol{\alpha x} \geq \alpha_0$ *defines* the face F. A face F is *proper* if $F \neq \emptyset$ and $F \neq P$. A proper face F is a *facet* if it is maximal, i.e., there is no other proper face F' such that $F \subset F'$. An inequality that defines a facet is said to be *facet-defining*.

Figure 3.2 depicts a formulation for $X = \{(0,1),(1,1),(2,1),(1,2)\}$. The points corresponding to its linear relaxation are shown in light orange, while those in $Conv(X)$ are in dark orange. The optimal fractional solution x^* can be cut by either of the two cutting planes depicted as dashed lines. The dashed lines correspond to the sets of format $\boldsymbol{\alpha x} = \alpha_0$ given by the points that satisfy the cutting planes as equalities. The red line only intersects $Conv(X)$ at $(1,2)$, so the corresponding valid inequality is a proper face, not facet-defining. However, the green line intersects $Conv(X)$ at the segment joining points $(0,1)$ and $(1,2)$, so the corresponding valid inequality (which could be $-x_1 + x_2 \geq -1$ or any of its positive scalar multiples, like $-2x_1 + 2x_2 \geq -2$) defines the facet $Conv(\{(0,1),(1,2)\})$, and therefore, is likely to be a better cutting plane. See Note 3.13 for an additional discussion on facets and their representations.

The Branch-and-Cut Algorithm (BCA) is an enhancement of the BBA where formulations can be dynamically strengthened by adding cutting planes. The BCA pseudo-code is presented in Algorithm 2. It differs from the BBA in lines 13-17. If $\boldsymbol{x}^*$ is fractional, a *separation procedure* is called to search for a cutting plane. If found, it is added to the current S and the algorithm loops back to line 6. If no cutting plane is found, branching is performed. Lines 13 and 15 indicate that, if the MIP contains variables that are not required to be integer, we may use more general *mixed-integer inequalities* of format $\boldsymbol{\alpha x} + \boldsymbol{\beta y} \geq \alpha_0$ for cutting a point $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ with

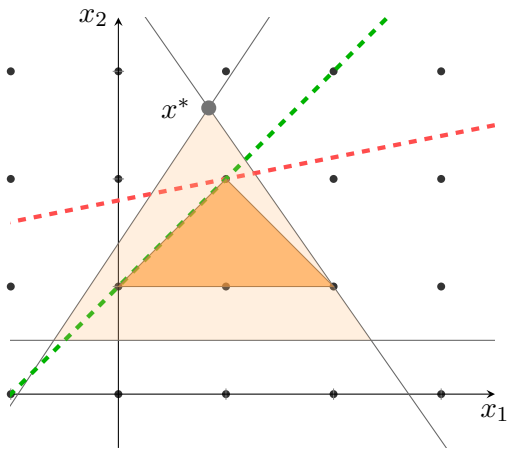


Figure 3.2: Examples of cutting planes

fractional $\boldsymbol{x}^*$. Such inequality is valid if it is satisfied for all points in $(\boldsymbol{x}, \boldsymbol{y}) \in P$ such that $\boldsymbol{x} \in \mathbb{Z}^n$. Some remarks about the BCA:

- When compared to the BBA, the BCA may spend a lot more time in each node, separating cutting planes and reoptimizing LPs, in the hope of obtaining a very substantial reduction in the total number of nodes.
- In order to improve node convergence, it is usual to try to separate multiple cutting planes for the same fractional point, adding several of them at once.
- Sometimes, after some rounds of separation in the same node, it is possible that the cutting planes start to only cut the fractional solution slightly, leading to negligible improvements in the z values. This is known as *tailing-off*. This is much more frequent when the used cuts are not deep, but it may happen even with facet-defining cuts. Anyway, if such an undesirable tailing-off situation is detected, it is better to stop the separation for that node and proceed with branching.

The BCAs implemented in general-purpose MIP solvers include sophisticated separation procedures that rely on quite general techniques for finding cutting planes directly from the MIP coefficients (Note 3.3). The cutting planes thus obtained are seldom facet-defining. Yet, they may be good enough to obtain major improvements in performance when compared to pure BBA.

Algorithm 2 Branch-and-cut algorithm for solving MIP (3.2)

$UB \leftarrow \infty$
$BestSol \leftarrow$ NULL
$L \leftarrow \{P\}$
while $L \neq \emptyset$ **do**
    Remove a subproblem S from L
    Solve $z_S = \min \boldsymbol{cx} + \boldsymbol{hy}$ subject to $(\boldsymbol{x}, \boldsymbol{y}) \in S$ ▷ If infeasible, $z_S = \infty$
    if $z_S < UB$ **then** ▷ If z_{IP} integer, $\lceil z_S \rceil < UB$
        if $\boldsymbol{x}^*$ is integer **then**
            $UB \leftarrow z_S$
            $BestSol \leftarrow (\boldsymbol{x}^*, \boldsymbol{y}^*)$
            Remove from L subproblems now prunable with UB
        else
            Try to separate $\boldsymbol{\alpha x} + \boldsymbol{\beta y} \geq \alpha_0$ cutting $(\boldsymbol{x}^*, \boldsymbol{y}^*)$
            if cutting plane found **then**
                $S \leftarrow S \cap (\boldsymbol{\alpha x} + \boldsymbol{\beta y} \geq \alpha_0)$
                goto 6
            end if
            Choose a variable x_j such that x_j^* is fractional
            Insert $S \cap (x_j \geq \lfloor x_j^* \rfloor)$ and $S \cap (x_j \leq \lceil x_j^* \rceil)$ into L
        end if
    end if
end while
return $(BestSol, UB)$

When a BCA is being used for solving the formulation of a specific LCOP, it is possible to do better. One may take its particular combinatorial structure into account and perform a theoretical investigation of the polyhedra $Conv(X)$ defined by the sets X that correspond to the LCOP instances. The goal is identifying *families of inequalities* that are facet-defining at least in some cases. Some families of inequalities, usually the simplest ones, can be separated in polynomial time. Other families lead to separation problems that are also $\mathcal{NP}$-hard. However, even in the last case, the combinatorial structure of the LCOP may help in devising reasonably effective and efficient heuristic separation procedures.

For example, consider the vertex cover instance defined by the graph $G = (V, E)$ depicted in Figure 3.3 and unitary weights. For that instance, Formulation (3.4) is $z_{\text{IP}} = \min \mathbf{1}\boldsymbol{x}$ subject to $\boldsymbol{x} \in P, \boldsymbol{x} \in \mathbb{Z}^6$, where $P = \{x_1 + x_2 \geq 1, x_1 + x_4 \geq 1, x_1 + x_5 \geq 1, x_2 + x_3 \geq 1, x_2 + x_4 \geq 1, x_2 + x_5 \geq 1, x_3 + x_6 \geq 1, x_4 + x_5 \geq 1, x_5 + x_6 \geq 1, \mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1}\}$. As its linear relaxation yields the fractional solution with value 0.5 for all variables, with $z_{\text{LP}} = 3$, the BBA would need to branch and some nodes would have to be explored in order to show that $z_{\text{IP}} = 4$.

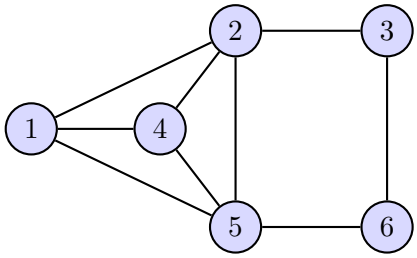


Figure 3.3: Example of vertex cover instance, unitary weights

However, suppose that after a theoretical investigation of the vertex cover polyhedra one discovers that if $C \subseteq V$ induces a maximal *clique* (complete subgraph) in G then the inequality $\sum_{i \in C} x_i \geq |C| - 1$ is valid and facet-defining. Then, one could also devise a separation procedure for that family of cuts, taking a fractional solution $\boldsymbol{x}^*$ as input and looking for a maximal clique C such that $\sum_{i \in C} x_i^* < |C| - 1$. However, as this separation problem is also $\mathcal{NP}$-hard, it is better to propose a heuristic separation procedure, not guaranteed to find a violated cut if one exists. Anyway, assume that such separation is embedded in a BCA specialized in solving vertex cover problems. If in our example that separation heuristic finds $x_1 + x_2 + x_4 + x_5 \geq 3$

for cutting the fractional solution of the linear relaxation, then the second LP would already yield an integer solution with $z = 4$ and the BCA would finish in the root node.

3.4. Successes and Limitations of the BCA: two examples

BCAs are definitely the best known way of solving general MIPs. We talk here about BCAs specialized for solving important LCOPs.

3.4.1. The Traveling Salesperson Problem

The classic Traveling Salesperson Problem (TSP) is often mentioned, even by people not in the optimization community, as the most prototypical $\mathcal{NP}$-hard combinatorial optimization problem.

Definition 3.12: Traveling salesperson problem. Instance: undirected graph $G = (V, E)$ and edge costs c_e, $e \in E$. Solutions: the tours of G, i.e., the cycles that visit each vertex in V exactly once. Goal: minimize the sum of the cost of the edges in the Hamiltonian cycle.

The TSP can be formulated as:

$$\min \quad \sum_{e \in E} c_e x_e \tag{3.5a}$$

$$\text{s.t.} \quad \sum_{e \in \delta(i)} x_e = 2 \qquad i \in V \tag{3.5b}$$

$$\sum_{e \in \delta(S)} x_e \geq 2 \qquad S \subset V \tag{3.5c}$$

$$\boldsymbol{x} \in \mathbb{B}^{|E|}. \tag{3.5d}$$

Binary variable x_e, $e \in E$, indicates whether edge e belongs to the tour. Constraints (3.5b) are known as Degree equalities, while (3.5c) are known as Subtour Elimination inequalities. Dantzig et al. [1954] used that formulation to solve a TSP instance

with 49 vertices, an amazing result considering the very limited computers available at that time.

Starting in the late 1970s, several authors performed an intense investigation of the TSP polyhedra, corresponding to the convex hull of the integer points in that formulation. Many families of facet-defining inequalities were found, including 2-Matching, Comb, Path, Clique-Tree, Bipartition, Ladder, and Domino-parity inequalities. Subtour Elimination and 2-Matching Inequalities can be separated in polynomial time; separation heuristics for the more complex families have been proposed. All that research effort went along with increasingly impressive practical results. In 1978, the largest solved instance had 120 vertices. In 2006, all the TSPLIB instances with up to 85,900 vertices could be solved to optimality (the largest instance taking 136 years of CPU time!) by a highly sophisticated branch-and-cut implementation (Note 3.15). More importantly, all the tested 463,000 random Euclidean instances ranging from 100 to 2500 vertices could be solved with default parameters. The 10,000 instances with 1000 vertices took an average time of 10 minutes, as reported in Applegate et al. [2007]. As expected, since the TSP is $\mathcal{NP}$-hard, there are some much smaller instances that may take a long time to be solved, like those constructed by Hougardy and Zhong [2021] and Vercesi et al. [2023]. However, those instances are not likely to be found in practice.

It is tempting to view the big success on the TSP (and also the significant successes on several other problems, like for example, the Independent Set Problem [Padberg, 1973, Rossi and Smriglio, 2001, Rebennack et al., 2011] or the Max Cut Problem [Barahona and Mahjoub, 1986, De Simone and Rinaldi, 1994]) of the approach "natural formulation + polyhedral investigation + separation procedures = BCA" as an indication that it may work well on every LCOP. Unfortunately, this does not appear to be true.

3.4.2. The Capacitated Vehicle Routing Problem

The Capacitated Vehicle Routing Problem (CVRP) is the most classic vehicle routing variant [Dantzig and Ramser, 1959].

Definition 3.13: Capacitated vehicle routing problem. Instance: undirected graph $G = (V, E)$, where $V = \{0\} \cup V_+$, vertex 0 represents a depot and V_+ the set of customers; edge costs c_e, $e \in E$; integer positive demands d_i, $i \in V_+$; and

integer vehicle capacity W. Solutions: sets of routes in G that, together, visit all customers exactly once. A route is a vertex-elementary cycle (a cycle that does not repeat vertices) that passes by the depot and such that the sum of the demands of the customers in it does not exceeds W. Goal: minimize the sum of the cost of the edges in the routes.

The CVRP has the following natural formulation [Laporte and Nobert, 1983]:

$$\min \quad \sum_{e \in E} c_e x_e \tag{3.6a}$$

$$\text{s.t.} \quad \sum_{e \in \delta(i)} x_e = 2 \qquad i \in V_+ \tag{3.6b}$$

$$\sum_{e \in \delta(S)} x_e \geq 2\lceil d(S)/W \rceil \qquad S \subseteq V_+ \tag{3.6c}$$

$$x_e \leq 1 \qquad e \in E \setminus \delta(0) \tag{3.6d}$$

$$\boldsymbol{x} \in \mathbb{Z}_+^{|E|}. \tag{3.6e}$$

Variable x_e, $e \in E$, indicates how many times edge e appears in the solution. Note that edges adjacent to the depot can be used twice, in case of routes visiting a single customer. Constraints (3.6b) are also known as degree equalities. Constraints (3.6c), where $d(S) = \sum_{i \in S} d_i$, are Rounded Capacity Cuts (RCCs). The quantity $\lceil d(S)/W \rceil$ is a lower bound on number of routes that should visit customers in S. RCCs make sure that routes are connected to the depot and also that they do not exceed capacity. Exact separation of RCCs is $\mathcal{NP}$-hard, but very good separation heuristics do exist [Lysgaard, 2003].

The CVRP has many similarities with the TSP (actually, it generalizes it). Given the enormous practical importance of vehicle routing, in the 1990s and the early 2000s, many researchers tried to emulate what have been done on TSP and create a very good BCA for the CVRP. So, they investigated the polyhedra defined by the convex hull of the integer points in that formulation. Several families of valid inequalities were found, including Framed Capacity, Strengthened Comb, Multistars, and Extended Hypotour, as summarized in Naddef and Rinaldi [2002]. Suitable heuristic separation procedures were devised. However, the obtained results can not be compared with those obtained in the TSP. By looking at the several BCAs for

the CVRP proposed in that period [Araque et al., 1994, Augerat et al., 1995, Blasum and Hochstättler, 2000, Ralphs et al., 2003, Achuthan et al., 2003, Wenger, 2003, Ralphs, 2003, Lysgaard et al., 2004] one can observe a "diminishing returns" effect, where substantial theoretical and implementation efforts achieve results that are only marginally better than those of previous works. In fact, some CVRPLIB instances with only 50 customers could not be solved by any of those BCAs.

Why BCAs work so much better on TSP than on CVRP?

- The starting TSP formulation (3.5) is already quite strong, usually obtaining gaps of less than 0.5% on TSPLIB instances. By also separating 2-matching inequalities, one can obtain in polynomial time (Note 3.14) a lower bound that is often less than 0.2% away from the optimal in those typical instances. This leaves a small gap to be closed by the heuristic separation of complex inequalities and by branching.

- The starting CVRP formulation (3.6) is not so strong, obtaining gaps between 2%–5% on typical CVRPLIB instances (even if exact separation of RCCs by MIP is used). By adding all known inequalities, using the best available separation procedures, typical gaps decrease to around 1%–4%. Those gaps may look small, but (at least for the sake of solving instances to optimality) they are not, truly leading to very large branch-and-bound trees.

Branch-and-cut algorithms are the best known way of solving general MIPs and have found spectacular successes in solving some important $\mathcal{NP}$-hard combinatorial problems. Yet, quite modest results obtained in other very important problems show its limitations. So, other alternatives for obtaining stronger formulations should be considered. The most important such alternative is perhaps the use of column generation.

Notes

3.1. Gomory's cutting plane algorithm. The first algorithm for solving IPs [Gomory, 1958] is based on a procedure that is guaranteed to separate a cutting plane (now known as a Gomory Cut) for any optimal fractional solution of a linear relaxation. He proved that, by iterating that procedure, an integer solution would be obtained with a finite number of cuts. That algorithm was generalized for MIPs in Gomory [1960]. Unfortunately, Gomory's cutting plane algorithm does not work well in practice. As the cuts are not likely to be deep, convergence is slow and the size of the LPs grows a lot. Moreover, the method is prone to severe numerical issues.

3.2. Late recognition for Land and Doig. The BBA was proposed in Land and Doig [1960] (the name branch-and-bound was coined in Little et al. [1963]; the variant shown as Algorithm 1, where the tree is necessarily binary, appears in Dakin [1965]). It proved to be much better in practice than Gomory's cutting plane algorithm and was adopted in the mid-1960s as *the* algorithm for solving MIPs, a situation that lasted until the 1990s [Bixby, 2012]. Even today, the BBA is still at the core of all widely used approaches for solving MIPs. More advanced algorithms, like BCA, Branch-and-Price Algorithm (BPA), or Branch-Cut-and-Price Algorithm (BCPA) — the last two algorithms will be defined in Chapter 4 — are built on top of the basic BBA. Finally, there are also many BBAs for particular COPs that do not use LPs, the bounds used for pruning nodes being obtained by combinatorial procedures or by Lagrangian Relaxation (Chapter 5).

The delay in conferring major awards and honors for Ailsa Land and Alison Harcourt (née Doig), the creators of such an impactful and seminal algorithm, is conspicuous. In fact, before 2018 (when the movement for better recognizing the contributions of women to science got momentum) the only such honors were that the Canadian OR Society awarded Land with the Harold Larnder prize in 1994 and that Land and Doig [1960] was chosen as one of the ten classic papers in integer programming to be reprinted in Jünger et al. [2010]. Land was posthumously awarded the EURO Gold Medal, the highest distinction

within OR in Europe, in 2021. In contrast, most other pioneers mentioned in this book received honors not so long after their seminal contributions.

Besides gender and even geographical issues (Harcourt made her career in Australia), we speculate about another additional possible explanation for that relative lack of recognition: the BBA was too simple and too obviously exponential to be viewed in the 1960s as a great piece of mathematics! In fact, many researchers of that time were looking for a "mathematically deep" and efficient algorithm for integer programming, something comparable to the simplex algorithm for linear programming. The hopes of finding such a holy grail algorithm would only subside in the mid-1970s, when the $\mathcal{NP}$-completeness theory was established.

3.3. Modern MIP Solvers. Branch-and-Cut is now the dominating approach for solving general MIPs. Actually, the modern MIP solvers are highly complex and sophisticated codes that incorporate many advanced elements, for example:

- Preprocessing – There are techniques for strengthening the coefficients of the constraints in a MIP. By probing and propagating logical implications it may be possible to fix some variables. Preprocessing may also be performed in nodes down the tree: as branching constraints and fixing by reduced costs restrict the subproblems, it may be worth preprocessing them again. In fact, as mentioned in Note 3.6, fixing by reduced costs and logical implications can be combined to provide powerful reduction tests. Many specific preprocessing techniques are described in Achterberg et al. [2020].
- Primal heuristics – Finding good feasible solutions as early as possible is crucial to BCA performance, allowing effective pruning by bound and also effective fixing by reduced costs. MIP solvers apply many heuristic techniques for finding improving integer solutions, from simply rounding of the current fractional solution to complex local search based methods.
- Cut generation – Many families of cutting planes that were proposed in the literature have been incorporated into the MIP solvers. The families range from improved versions of the original Gomory's Mixed-Integer

Cuts to families of cuts that are derived from combinatorial structures that are automatically detected in the MIP constraints. For example, it is possible to build a conflict graph having two vertices for each binary variable, one corresponding to the variable itself and one for its complement, and edges between pairs of vertices that can be proven to be incompatible (can not have both value one). Then, well-known cuts for the Independent Set Problem (ISP), like cliques and odd holes, can be separated from that graph. Of course, a cut that is facet-defining for a MIP substructure is not likely to be facet-defining for the complete MIP, but it may still be very useful.

- Cut selection – The goal is to find a relatively small set of cutting planes that can significantly reduce the gaps without burdening the LPs too much. As indicated by Achterberg [2007], the cut selection in MIP solvers is usually based on efficacy and orthogonality. The efficacy is the Euclidean distance of the cut hyperplane to the current LP solution, and an orthogonality bound makes sure that the cuts added to the LP form an almost pairwise orthogonal set of hyperplanes.
- Symmetry handling – The solvers try to identify groups of symmetric variables and either try to branch on their aggregations or introduce symmetry-breaking constraints.
- BB tree management – There are sophisticated methods for choosing the branching variable (Note 3.10) and for choosing the next active subproblem to be explored.

In some cases, those advanced elements can be so effective that they may "fix", up to a point, a weak formulation provided by the user. More details about general MIP solvers can be found in Lodi [2010]. According to Bixby [2022], the best MIP solvers of today, running in the same hardware, are 4 or 5 orders of magnitude faster than the MIP solvers of 1990 (which were essentially pure BBAs) on typical benchmark sets. By also considering the hardware progress, including the widespread availability of multiple cores, we have 7 to 8 orders of speed-up. Yet, given the exponential complexity of all BB-based algorithms, the size of the instances that can be solved in a given amount of time increased a lot, but by a much smaller factor. Koch et al. [2022] also provides an assessment of the progress in MIP solving, covering

the period from 2001 to 2020.

The current most advanced commercial MIP solvers, COPT, Cplex, Gurobi, and Xpress

3.4. Constraint Programming and SAT solvers. The main "rivals" of MIP solvers as an off-the-shelf tool for the exact solution of $\mathcal{NP}$-hard COPs are Constraint Programming (CP) solvers and SAT solvers.

- A CP model represents a COP as a set of logical constraints over a vector of discrete variables $\boldsymbol{x}$. In many cases, since higher-level constructs are available, such a model can be more intuitive than a MIP model. For example, the *all_different* constraint type may be used for saying that a subset of the variables should have distinct values. CP solvers use advanced techniques such as constraint propagation and backtracking to explore the solution space [Rossi et al., 2006]. In principle, CP is more suited for solving decision problems, where the goal is finding a feasible solution or proving that none exists. However, optimization problems are also handled via objective function constraints of format $c(\boldsymbol{x}) \leq \overline{z}$ and $c(\boldsymbol{x}) \geq \underline{z}$, where $\overline{z}$ is an upper bound that is updated as improving solutions are found and $\underline{z}$ is a lower bound that is updated when it is proven that no solutions with a smaller cost exist. CP has proven to be very effective for highly constrained problems (like those often found in scheduling, project planning, and timetabling) where the fixing of a single variable may trigger significant reductions in the search space.
- SAT solvers address the Boolean Satisfiability Problem, which asks whether there exists a solution that satisfies a given Boolean formula. SAT solving (which may be seen as a particular case of CP) has seen big advancements over the years, becoming highly efficient for some problems, especially in verifying circuit designs and software verification. Advanced techniques like conflict-driven clause learning played a crucial role in that. A comprehensive reference on the topic is Biere et al. [2021].

There is some degree of cross-fertilization and integration between those two COP-solving paradigms and MIP solving. For example, the open-source Solving Constraint Integer Programs [Bolusani et al., 2024] SCIP optimization

suite was created with that integration idea in mind. Another example is Yunes et al. [2010]. There is even the CPAIOR (International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research) Conference Series dedicated to the topic. Indeed, most modern MIP solvers borrow several CP techniques, for instance, to better propagate the effect of fixing variables by branching or by reduced costs.

3.5. Cut callback routines. BCA algorithms for specific LCOPs can be implemented on top of general-purpose MIP solvers. User-written separation procedures can be defined as *cut callback routines*. Those routines then will be called by the MIP solver for each fractional solution found along the BCA; the cutting planes that may be found are added to the formulation. This is very convenient because the BCA coder will profit from several advanced algorithmic elements already implemented in those MIP solvers.

We remark that, at the time of writing, general-purpose MIP solvers do not offer an equivalently easy way of implementing a Branch-and-Price Algorithm.

3.6. Fixing by reduced costs. The fixing of variables by reduced costs is a simple technique for improving the performance of BBAs and BCAs.

Theorem 3.2: *Let UB be the value of the best known solution for a MIP in general format (3.2). After solving its linear relaxation, define $gap = UB - z_{LP}$ (if z_{IP} is known to be integer, like in the case of a pure IP with $\boldsymbol{c}$ integer, $gap = UB - \lceil z_{LP} \rceil$). Let $\bar{\boldsymbol{c}}^*$ be the vector with the reduced cost values. Then, every integer variable x_j, $j \in [n]$, such that $\bar{c}^*_j \geq gap$ should have value zero in any improving MIP solution.*

Proof. We provide a proof for the particular case of an IP in format $z_{\text{IP}} = \min \boldsymbol{cx}$ subject to $\boldsymbol{Ax} \geq \boldsymbol{b}$, $\boldsymbol{x} \in \mathbb{Z}^n_+$ (the proof for the general case is left as an exercise). The dual of its linear relaxation is $\max \boldsymbol{\pi b}$ subject to $\boldsymbol{\pi A} \leq \boldsymbol{c}$, $\boldsymbol{\pi} \in \mathbb{R}^{1\times m}_+$. Let $\boldsymbol{\pi}^*$ be the optimal dual solution found, with cost $\boldsymbol{\pi}^*\boldsymbol{b} = z_{\text{LP}}$. By definition, $\bar{\boldsymbol{c}}^* = \boldsymbol{c} - \boldsymbol{\pi}^*\boldsymbol{A}$. Let x_j be a variable with $\bar{c}^*_j \geq gap$ and with positive value in some solution $\boldsymbol{x}'$. This means that $\boldsymbol{x}'$ should also be a solution for the modified IP $\min \boldsymbol{cx}$ subject to $\boldsymbol{Ax} \geq \boldsymbol{b}$, $x_j \geq 1$, $\boldsymbol{x} \in \mathbb{Z}^n_+$. The dual of its linear

relaxation is max $\boldsymbol{\pi b}+\alpha$ subject to $\boldsymbol{\pi A}+\alpha \boldsymbol{e}_j \leq \boldsymbol{c}$, $\boldsymbol{\pi} \in \mathbb{R}_+^{1\times m}, \alpha \in \mathbb{R}_+$. Solution $(\boldsymbol{\pi}, \alpha) = (\boldsymbol{\pi}^*, \bar{c}_j^*)$ is feasible for it and costs $z_{\text{LP}} + \bar{c}_j^*$. Therefore, $\boldsymbol{cx'} \geq UB$ and $\boldsymbol{x'}$ can not be an improving solution for the original IP. □

Variables fixed to zero using the above result can be eliminated, leading to lighter LPs during the BBA tree search. As mentioned in Exercise E 3.3, it may also be possible to fix variables to their upper bounds. In fact, the fixing by reduced costs can be tried in all nodes, after every LP is solved, using the local lower bounds z_S and the current global upper bound UB for computing *gap*. Variables fixed at a node are eliminated from all its descendent nodes. Fixing by reduced cost is effective when the absolute gaps are small, which may happen already in the root node in the case of strong formulations (and if some good heuristic UB is available). However, even for not-so-strong formulations, at some level down the tree, the gaps will be small enough to allow an effective fixing.

The reasoning in the proof of Theorem 3.2 can be used for deriving more complex and powerful reduction tests, combining reduced costs with logical implications. For example, suppose that an analysis of the structure of the constraints of an IP shows that if a certain variable x_i has a positive value then either x_j or x_k should have a positive value. Then, if $\bar{c}_i^* + \min\{\bar{c}_j^*, \bar{c}_k^*\} \geq gap$, x_i can be fixed to zero. Those tests can be particularly effective for MIPs modeling some specific LCOPS, as happens in the Steiner Problem in Graphs (Note 3.17).

3.7. Chvátal-Gomory cuts. The Chvátal-Gomory procedure [Chvátal, 1973] for deriving valid cuts is the following. Consider a set $X = \{\boldsymbol{x} \in \mathbb{Z}_+^n : \boldsymbol{Ax} \leq \boldsymbol{b}\}$, where $\boldsymbol{A}$ has dimension $m \times n$. Let $\boldsymbol{\rho} \in \mathbb{R}_+^{1\times m}$ be a vector of non-negative multipliers. Then, inequality $\boldsymbol{\rho Ax} \leq \boldsymbol{\rho b}$ is also valid for X. As $\boldsymbol{x} \geq \boldsymbol{0}$, by rounding down every component of $\boldsymbol{\rho A}$, we get the weakened valid inequality $\lfloor \boldsymbol{\rho A} \rfloor \boldsymbol{x} \leq \boldsymbol{\rho b}$. By noticing that its LHS is integer for any $\boldsymbol{x} \in X$, it is also possible to round down the RHS, producing the stronger valid inequality $\lfloor \boldsymbol{\rho A} \rfloor \boldsymbol{x} \leq \lfloor \boldsymbol{\rho b} \rfloor$. The separation procedure proposed in [Gomory, 1958] can be viewed as a particular case of the Chvátal-Gomory procedure (which explains its name).

Chvátal proved that any valid inequality for X, including those defining the facets of $Conv(X)$, can be derived by the recursive application of that procedure. In fact, he defined a hierarchy of valid inequalities. The original inequalities in $\boldsymbol{Ax} \leq \boldsymbol{b}$ have rank 0. The inequalities that need one application of the Chvátal-Gomory rounding procedure over $\boldsymbol{Ax} \leq \boldsymbol{b}$ to be obtained have rank 1. Then, rank 2 inequalities are those obtainable by the rounding procedure over inequalities with rank ≤ 1, and so on. The exact separation of rank 1 cuts (finding multipliers that lead to a violated cut, if such multipliers do exist) is already $\mathcal{NP}$-hard [Eisenbrand, 1999]. However, "offline" computation experiments with rank 1 cut exact separation by MIP (too slow to be used in practice during an actual BCA run) indicated that those cuts have a large potential for significantly reducing gaps [Fischetti and Lodi, 2007].

3.8. Objective value cuts. Suppose an IP in format (3.1) and such that z_{IP} is known to be integer. If a fractional solution x^* leads to a fractional $z^* = \boldsymbol{cx}^*$ it is very tempting to add an *objective value cut* $\boldsymbol{cx} \geq \lceil z^* \rceil$. Many people have actually tried to do that. As the computational results are seldom good, those negative experiences are not published and instead become part of the unwritten knowledge or "community folklore".

But why does adding objective value cuts usually turn out to be ineffective, or even harmful? The first effect of such a cut within a BCA is to increase the node value to $\lceil z^* \rceil$. Unhappily, this is not really useful, as that node lower bound was already known and utilized both for pruning and fixing by reduced costs. On the other hand, the side effect of an objective value cut is creating nodes with many alternative optimal fractional solutions. The z value in those nodes and in their children will only move if all solutions with value $\lceil z^* \rceil$ are removed either by cutting or by branching. The problem is that some important decisions within the BCA are based on objective function improvement, like for example, the choice of the branching variable using strong branching (Note 3.10). *This means that parts of the algorithm will become blind while the z values are stuck to $\lceil z^* \rceil$.* Moreover, an objective value cut leads to an extremely degenerate dual solution: the dual variable of the cut has value 1 and the remaining dual variables have value zero (so, all variables have reduced cost zero). While this happens, dual variables and

reduced costs also can not be used for guiding decisions.

3.9. Set Covering, Set Partitioning, and Set Packing. There are three subfamilies of IPs that deserve particular consideration. In all those subfamilies the coefficient matrix $\boldsymbol{A}$ belongs to $\mathbb{B}^{m\times n}$. A Set Covering Problem (SCP) has the format $\min \boldsymbol{cx}$ subject to $\boldsymbol{Ax} \geq \mathbf{1}$, $\boldsymbol{x} \in \mathbb{Z}^n_+$. A Set Partitioning Problem (SPP) has format $\min \boldsymbol{cx}$ subject to $\boldsymbol{Ax} = \mathbf{1}$, $\boldsymbol{x} \in \mathbb{Z}^n_+$. Finally, a Set Packing Problem (SPcP) has format $\max \boldsymbol{cx}$ subject to $\boldsymbol{Ax} \leq \mathbf{1}$, $\boldsymbol{x} \in \mathbb{Z}^n_+$. In a small abuse of nomenclature, IPs where most constraints fit into one of those formats are often still referred to as SCPs, SPPs, or SPcPs.

As will be seen in Chapter 4, there are many situations when a Dantzig-Wolfe reformulation of an IP produces either an SCP, an SPP, or an SPcP, but with a very large number of variables, requiring the use of Column Generation techniques. However, the interest in those three families of IPs started in the early 1960s, when it was realized that they could be very useful for modeling some complex situations and still keeping that complexity "hidden". Consider for example the following problem.

Definition 3.14: Airline Crew Scheduling Problem. Instance: Set A of airports and set of F flights that will occur during a planning period of T consecutive days. A flight $f \in F$ is characterized by its origin airport $oa_f \in A$ (say, New York JFK), origin time of departure ot_f (like, 3/12/2023 at 11:35 AM), final destination airport $da_f \in A$ (say, Miami MIA), and final destination time of arrival dt_f (like, 3/12/2023 at 4:15 PM). Flights may have intermediate stops, but this is not explicitly shown in the instance data. Assume that an unlimited number of crews are available, all of them based at the same airport $b \in A$ (more complex variants consider multiple bases and limits on the number of available crews), and that any crew can work on any flight. The same crew that worked on flight f_1 can also work on flight f_2, provided that $da_{f_1} = oa_{f_2}$ and $dt_{f_1} + \Delta_{f_1} \leq ot_{f_2}$, where Δ_{f_1} is the minimum rest time after flight f_1. Sets $\mathcal{R}$ and $\mathcal{C}$ of work rules and cost rules, respectively, are also given. Solutions: sets of rosters that, together, include each flight in F exactly once. A *roster* is a sequence of flights that start and end at the base airport and can be done by the same crew, respecting the work rules

$\mathcal{R}$. The work rules may include complex safety regulations, union agreements, and company policies. Days without flights in a roster, if they are spent at the base airport, are interpreted as rest days. Goal: minimize the sum of the costs of the rosters. Those costs are calculated using potentially complex cost rules $\mathcal{C}$.

The Airline Crew Scheduling Problem was one of the first industry problems to be routinely solved using IP formulations (as surveyed in Arabeyre et al. [1969]). The problem can be formulated as an SPP as follows. Let Q be the set of all possible rosters. Let c_q be the cost of roster $q \in Q$ and let p_{qf}, $f \in F$, be a binary constant indicating whether flight f is included in it. The formulation is:

$$z_{\text{IP}} = \min \quad \sum_{q \in Q} c_q x_q \tag{3.7a}$$

$$\text{s.t.} \quad \sum_{q \in Q} p_{qf} x_q = 1 \qquad f \in F \tag{3.7b}$$

$$\boldsymbol{x} \in \mathbb{Z}_+^{|Q|}. \tag{3.7c}$$

A noteworthy aspect of the above SPP formulation is that the potentially intricate rules $\mathcal{R}$ are not explicitly expressed within it. Rosters that comply with all the rules in $\mathcal{R}$ are present in the SPP, while those that do not are simply absent. Similarly, the cost rules $\mathcal{C}$ are concealed within the procedure that calculates the cost of each roster before solving the SPP. To reduce the size of the formulation, flights may be aggregated into *rotations*, pre-defined "good" two-or-three-day sequences of flights starting and ending at the base airport. In that case, a roster should be a sequence of compatible rotations and the SPP model would have one row for each rotation. Even with that aggregation, real-world instances may lead to SPPs with a very large number of variables. In practice, it is frequent to solve the restricted problems only containing a subset $S \subset Q$ of pre-defined "good" rosters, $|S|$ being limited to keep the resulting SPP tractable. However, significantly better solutions can be obtained by also applying Column Generation, even if heuristically (see Chapter 6).

3.10. Pseudo-cost branching, strong branching. The choice of the branching variable is a key decision within a BB-based algorithm. Consider a hypothetical BBA run where every branching cuts 5% of the gap in both children nodes (the difference between their z values and their common parent z value corresponds to that proportion of the total integrality gap) and where an upper bound UB with value equal to z_{IP} is already known from the beginning. The BBA would explore $2^{21} - 1$ nodes. Now, suppose a similar situation except that each branching cuts 10% of the gap. The BBA would only explore $2^{11} - 1$ nodes. While that example is artificial, there are numerous real cases where an advanced branching strategy can indeed improve upon naive branching by orders of magnitude, especially on difficult problems where the search trees may be very large.

Many older BB-based algorithms used relatively simple rules for selecting the branching variable. This is not likely to be really good. Extensive computational experiments in Achterberg et al. [2005] showed that the popular rule of branching on the most fractional variable is not better than branching over a random fractional variable!

The strategy known as *pseudo-cost branching* was introduced in Bénichou et al. [1971]. The idea is to keep information from the branchings already performed in order to guide future choices. For every variable x_j that was already branched, a pseudo-cost is the average observed increase in the objective function value per unit of change in x_j. For example, assume that the *upward pseudo-cost* $\xi_j^+ = 8$ and that the *downward pseudo-cost* $\xi_j^- = 5$. If in the current node being explored $x_j^* = 0.7$ then the pseudo-cost estimate is that forcing $x_j \geq 1$ would increase the z value by 2.4 units, while $x_j \leq 0$ would increase the z value by 3.5 units. Pseudo-cost branching works but has limitations. In particular, it performs poorly at the top of the search tree because there is little or no past branching information. Note that those are exactly the most critical branching decisions. While a bad choice near the bottom of the tree only has a local impact, bad choices at the top of the tree may have a global impact on overall performance.

A potentially better strategy was motivated by the following observations:

- Consider a computationally expensive experiment where all candidates (integer variables with a fractional value in the LP solution) of a certain

node are tested and the resulting actual improvements in the z values are available for ranking and classifying the candidates. Typically, most candidates would be classified as "poor", some as "regular", and only relatively few as "good". In fact, it is not so rare that only one or two candidates would stand out as "excellent". *It would be also observed that rules or even pseudo-costs are not able to consistently identify the best candidates*. Even the very complex rules implicit in the trained neural networks obtained by recent Machine Learning approaches have limitations [Bengio et al., 2021, Scavuzzo et al., 2024].

- Consider a hypothetical perfectly balanced BB tree. The level at depth l (depth 0 corresponds to the root node) has 2^l nodes. So, *as the size of the search tree grows, the proportion of the total BBA running time spent at the top of the tree becomes very small, almost negligible*. Actual BB trees are not balanced, but that general observation is still true.

Strong branching is a computationally expensive procedure that tests many candidates before choosing one of them. Extensive experiments have shown that it most often pays, especially on nodes on the top of the tree. This strategy was originally proposed by Applegate, Bixby, Chvátal, and Cook around 1995 in their Concorde TSP solver (see Note 3.15) and was soon included in general MIP solvers. As will be fully discussed in Chapter 8, there are several tricks for making strong branching more efficient. We now mention the following:

- The effort in testing many candidates that are not chosen should not be wasted. The produced information can be used to significantly improve the pseudo-costs. In fact, the best overall strategy is often to combine strong branching on the top of the tree with the resulting improved pseudo-cost branching on the bottom [Achterberg et al., 2005].
- The re-optimization of an LP after a branching constraint is introduced is done using the dual simplex algorithm, hot-started with the parent LP optimal basis (Note 1.7). When testing one of the children corresponding to a candidate, if a fixed number of dual simplex iterations obtains a very small increase in the z value, the algorithm is stopped, and that unpromising candidate is dropped.

3.11. Branching over linear expressions. The BBA and the BCA for solving MIP (3.2) may only perform branching over individual x variables, as presented in Algorithm 1 and Algorithm 2. However, in some situations, it may be advantageous to also perform branching over linear expressions. A *linear expression* $\boldsymbol{\alpha x}$, $\boldsymbol{\alpha} \in \mathbb{R}^{1\times n}$, is suitable for branching if its value should be integer for any optimal MIP solution. If a fractional solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ is such that the value $\boldsymbol{\alpha x}^*$ is fractional, then one can branch on the disjunction $\boldsymbol{\alpha x} \leq \lfloor \boldsymbol{\alpha x}^* \rfloor$ or $\boldsymbol{\alpha x} \geq \lceil \boldsymbol{\alpha x}^* \rceil$. We present two classic examples of branching over linear expressions.

- Consider the SPP $\min \boldsymbol{cx}$ subject to $\boldsymbol{Ax} = \mathbf{1}$, $\boldsymbol{x} \in \mathbb{Z}^n_+$, where $\boldsymbol{A} \in \mathbb{B}^{m\times n}$. In many practical situations (like the Airline Crew Scheduling Problem shown in Note 3.9) n can be fairly large. Consider a branching over a certain individual variable x_j. Fixing x_j to one is likely to be effective, in the sense of moving the z value of the child node significantly. This happens because all the other variables that conflict with x_j will be also fixed to zero. In fact, a preprocessing step would eliminate all rows $i \in [m]$ such that $a_{ij} = 1$ from the corresponding child node. On the other hand, fixing x_j to zero barely changes the problem and the child node z value probably will not move significantly. The overall effect would be a very unbalanced and large search tree.

  Ryan and Foster [1981] introduced a better branching scheme for the SPP. The idea is to select two rows l and k and branch on the expression $\sum_{j\in[m]:a_{lj}+a_{kj}=2} x_j$. If that expression is set to 1, then rows l and k should be covered by the same variable. This is equivalent to eliminating all variables x_j such that $a_{lj} + a_{kj} = 1$. If the expression is set to 0, then rows l and k should be covered by distinct variables, which is equivalent to eliminating all variables x_j such that $a_{lj} + a_{kj} = 2$. The Ryan-Foster Branching (RFB) is proved to be complete: if the solution of the SPP linear relaxation is fractional then there should be rows l and k such that the corresponding RF expression has a fractional value.

  RFB is sometimes used when the Dantzig-Wolfe reformulation of an IP produces an SPP. However, its use in a Column Generation context leads to potential problems, as will be discussed in Chapter 4 and in Chapter 8.

- Consider the TSP definition and its formulation (3.5). Any route should contain an even number of edges in $\delta(S)$, for any $S \subset V$. So, it is possible to branch over linear expressions $\sum_{e \in \delta(S)} x_e/2$, as first suggested in Clochard and Naddef [1993]. This scheme, which is sometimes referred to as branching on cutsets (in contrast with branching on edges), is complete. This is proven by simply realizing that branching over variable x_e, $e = \{u, v\}$, is equivalent to branching over the expression defined by the set $S = \{u, v\}$.

  Branching on cutsets can also be effective on VRPs. For example, consider the CVRP formulation (3.6). BCAs over it, like Lysgaard et al. [2004], may branch on expressions corresponding to sets of customers $S \subseteq V_+$. Silva et al. [2024] contains an extensive experimental evaluation of branching schemes for VRPs, including RFB and branching on cutsets.

3.12. Big-M constraints. After a number of formulation tricks are learned (the classic reference is Williams [1978], now in its 5th edition [Williams, 2013], see also Chen et al. [2010]), it is not so difficult to cast most LCOPs as MIPs. Yet, there are some types of formulations that are notoriously weak and should be avoided if possible. This includes those containing the so-called *Big-M constraints*, which are typically used to propagate the implications of a binary variable to a continuous variable (or to an integer variable that can assume large values). In those constraints, the binary variable appears multiplied by a large coefficient, in some cases by a really large coefficient (the Big-M).

The most classic example of a formulation having Big-M constraints is the Miller-Tucker-Zemlim (MTZ) formulation [Miller et al., 1960] proposed for the

Asymmetric TSP, defined over a directed graph $G = (V, A)$, where $V = [n]$:

$$\begin{aligned}
\min \quad & \sum_{e \in A} c_a y_a & & & (3.8a)\\
\text{s.t.} \quad & \sum_{a \in \delta^-(i)} y_a = 1 & & i \in V & (3.8b)\\
& \sum_{a \in \delta^+(i)} y_a = 1 & & i \in V & (3.8c)\\
& w_1 = 1 & & & (3.8d)\\
& 2 \le w_i \le n & & i \in V, i \neq 1 & (3.8e)\\
& w_i - w_j + (n-1) y_{ij} \le n-2 & & (i,j) \in A, j \neq 1 & (3.8f)\\
& (\boldsymbol{y}, \boldsymbol{w}) \in \mathbb{Z}_+^{|A|} \times \mathbb{Z}_+^n, & & & (3.8g)
\end{aligned}$$

where binary variable y_a indicates that arc a is used and w_i is the position of vertex i in the route with respect to vertex 1. Big-M constraints (3.8f) eliminate subtours in the following way. Consider the constraint corresponding to a certain $(i, j) \in A$, such that $j \neq 1$. If $y_{ij} = 0$, the constraint is nullified, since it only implies that $w_i - w_j \le n-2$, a redundant inequality. On the other hand, if $y_{ij} = 1$, the constraint implies that $w_j \ge w_i + 1$, which is its desired effect. In fact, for any cycle C formed by arcs that do not pass by vertex 1, it is impossible that $y_a = 1$ for all $a \in C$. MTZ formulation is much weaker than (3.6). As the binary variables are multiplied by the large coefficient $(n-1)$ in Constraints (3.8f), it is quite easy for a fractional solution to "fool" them. For example, if $i, j \neq 1$, fractional solutions containing $y_{ij} = y_{ji} = (n-2)/(n-1)$ (a near-integer subtour of size 2) may be feasible.

We remark that Big-M formulations, like the MTZ, may work reasonably well, up to a certain instance size, if the formulation is light, so each node in the BB tree can be quickly solved. However, sometimes such a formulation only works well because advanced MIP solvers can improve it a lot, both by preprocessing and by separating strong cutting planes.

3.13. Multiple representations of the same facet. In the example of Figure 3.2, all inequalities that are positive scalar multiples of $-x_1 + x_2 \ge -1$ define the facet $Conv(\{(0,1),(1,2)\})$ of $Conv(X)$. However, in general, it may

be harder to determine if two distinct inequalities define the same facet of $Conv(X)$ or not. In order to understand that we need some additional concepts.

Definition 3.15: Affine combination, affine independence. Given a finite set $X = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_t\}$ of points in an n-dimensional space, $\boldsymbol{x}$ is said to be an *affine combination* of the points in X if $\boldsymbol{x} = \sum_{j\in[t]} \boldsymbol{p}_j \lambda_j$ for some value of $\boldsymbol{\lambda} \in \mathbb{R}^t$ such that $\sum_{j\in[t]} \lambda_j = 1$. A set of points is said to be *affinely independent* if no point in it can be obtained as an affine combination of other points in that set.

Definition 3.16: Polyhedral dimension, full dimension. The *dimension of a polyhedron* P, denoted as $dim(P)$, is the maximum cardinality of a set of affinely independent points in P minus 1. A polyhedron $P \subset \mathbb{R}^n$ is said to be *full-dimensional* if $dim(P) = n$.

In the example of Figure 3.2, $dim(Conv(X)) = 2$. A possible maximum set of affinely independent points in $Conv(X)$ is $\{(0,1),(1,2),(2,1)\}$. The proof of the following results can be found in [Nemhauser and Wolsey, 1988].

Theorem 3.3: *The 0-dimensional faces of a polyhedron P correspond to the points in $Ext(P)$.*

Theorem 3.4: *A non-empty face F of polyhedron P is a facet if and only if $dim(F) = dim(P) - 1$.*

Theorem 3.5: *A polyhedron $P \subset \mathbb{R}^n$ can be minimally represented as $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{D}\boldsymbol{x} \geq \boldsymbol{d}\}$, where $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is any set of $n - dim(P)$ linearly independent equalities containing P and $\boldsymbol{D}\boldsymbol{x} \geq \boldsymbol{d}$ has exactly one inequality defining each facet of P.*

Theorem 3.6: *Suppose that $\boldsymbol{\alpha}\boldsymbol{x} \geq \alpha_0$ defines a face F of P and let $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ be a set of $n - dim(P)$ linearly independent equalities containing P. Then, all inequalities $(\theta\boldsymbol{\alpha} + \boldsymbol{u}\boldsymbol{A})\boldsymbol{x} \geq \theta\alpha_0 + \boldsymbol{u}\boldsymbol{b}$, where $\boldsymbol{u} \in \mathbb{R}^{1\times n-dim(P)}$ and $\theta \geq 0$ are equivalent, in the sense of defining the same face F.*

Theorem 3.6 indicates that the theoretical investigation of the polyhedra $Conv(X)$ associated with an LCOP can become considerably more complex if they are not full-dimensional. Specifically, it is necessary to prove that any newly discovered family of facet-defining inequalities does indeed define new facets, i.e., that they are not equivalent to other already known inequalities. For example, a TSP Subtour Elimination inequality $\sum_{e \in \delta(S)} x_e \geq 2$ in (3.5c) is equivalent to $\sum_{e \in E(S)} x_e \leq |S| - 1$, as the latter can be obtained from the former by subtracting the Degree constraints in (3.5b) corresponding to the vertices in S.

3.14. Existence of perfect formulations. The conjecture that $\mathcal{P} \neq \mathcal{NP}$ is widely believed to be true. In this paragraph, we assume that it is indeed true. So, it is not possible to find a *polynomially-solvable* perfect formulation (natural or extended) for any $\mathcal{NP}$-hard LCOP. This rules out the existence of two kinds of perfect formulations for those problems: (i) those that are polynomially-sized; and (ii) those that have an exponentially large number of constraints, but those constraints can be separated in polynomial time. The first impossibility is known to be true since the Ellipsoid method proved that linear programming belongs to $\mathcal{P}$ [Khachiyan, 1979]. The second impossibility is a less direct consequence of the Ellipsoid method, a result known as *the equivalence of separation and optimization* [Grötschel et al., 1981]: it is possible to solve an LP with exponentially many constraints in polynomial time, provided that its constraints can be separated in polynomial time. By the way, by duality, this also proves that it is possible to solve an LP with exponentially many variables in polynomial time, provided that its pricing subproblem can be solved in polynomial time.

Most people also believe that if an LCOP belongs to $\mathcal{P}$ then it is possible to find a polynomially-solvable perfect formulation for it. Interestingly, it is known that some problems in $\mathcal{P}$ do not admit perfect formulations (natural or extended) with polynomial size. For example, the matching problem has a perfect formulation [Edmonds, 1965] that includes an exponential family of inequalities (blossom inequalities) that can be separated in polynomial time. However, no perfect formulation with polynomial size is possible for that

problem [Yannakakis, 1991, Rothvoß, 2017].

3.15. Concorde TSP solver. The exceptional 600-pages book Applegate et al. [2007] is fully devoted to the TSP, in particular to its computational aspects. Besides covering the history of the problem and the contributions by hundreds of researchers, several of its chapters are focused on the BCA implemented in Concorde [2020], a code by the book authors that has been producing the best exact results since 1992. In particular, in 2006, it solved *pla85700*, the last open instance in the original TSPLIB [Reinelt, 1991]. Concorde is perhaps the most sophisticated code ever written for solving a particular COP (130,000 lines in language C at that time). Truly, besides solving very large instances of the most famous $\mathcal{NP}$-hard COP, several ideas first introduced in Concorde were eventually adopted in the algorithms for solving other COPs or even on general MIP solvers. For example, we can mention the concept of strong branching (Note 3.10).

By the way, Concorde could be classified as a branch-cut-and-price algorithm because only a small part of the $O(|V|^2)$ edge variables in Formulation (3.5), those that are more likely to appear in an optimal fractional solution, are included in the initial LPs. As the algorithm progresses, pricing over the remaining edge variables is performed for adding variables with negative reduced cost. The pricing is also very useful for fixing most of the variables (usually those that correspond to longer edges) to zero by reduced costs and eliminating them definitely. Yet, that pricing can be performed by inspection, i.e., by explicitly evaluating the reduced costs of each variable (actually, as described in Section 12.3 of Applegate et al. [2007], Concorde uses some tricks for efficiently pricing blocks of variables). Pricing by inspection does not fit in the definition of CG-based algorithm given in the Introduction of this book, which requires that the pricing should be solved as a subproblem by an optimization algorithm. So, we prefer classifying Concorde as a BCA. We believe that such a stricter definition makes sense. Otherwise, even the RSA would have to be considered a CG-based algorithm! Moreover, the "true" BCPAs that will be described in the next chapter have to face the crucial issue of how the new dual variables from the added cuts interfere with the algorithm used in the pricing. That issue does not exist when the pricing is performed by inspection.

3.16. Projection of extended formulations. Extended formulations can be compared against natural formulations through the concept of projection. The comparison is frequently carried out using ad hoc reasoning to demonstrate that specific families of inequalities in the natural space are indeed implied by the extended formulation. However, there are some methods that can help with that.

Given an extended formulation for a fixed set X (not a general LCOP formulation), the Fourier-Motzkin elimination method (see Schrijver [1986]) can be used to compute all inequalities defining its projection onto the natural space by eliminating one extended variable at a time. The method is implemented in packages like PORTA and Parma Polyhedra Library and can be used for analyzing small examples (the size of the output grows exponentially with the size of the input), providing useful insights for a more theoretical polyhedral investigation.

However, there is another useful mathematical tool. Consider an extended formulation in the following format: min $\boldsymbol{cx}$ subject to $(\boldsymbol{x}, \boldsymbol{y}) \in P$, $\boldsymbol{x} \in \mathbb{Z}^n$, $\boldsymbol{y} \in \mathbb{R}^p$, where $P = \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^n \times \mathbb{R}^p : \boldsymbol{Ax} + \boldsymbol{Dy} \geq \boldsymbol{b}\}$, matrices $\boldsymbol{A}$ and $\boldsymbol{D}$ having m rows. The *projection cone of* P *onto* x is defined as $C_x(P) = \{\boldsymbol{u} \in \mathbb{R}_+^{1\times m} : \boldsymbol{uD} = \boldsymbol{0}\}$. For each $\boldsymbol{u} \in C_x(Q)$, it is clear that $(\boldsymbol{uA})x \geq (\boldsymbol{ub})$ is a valid inequality in the space of the natural x variables. The following result states that all inequalities that define the polyhedron $Proj_{\boldsymbol{x}}(P) \equiv \{\boldsymbol{x} \in \mathbb{R}^n : \exists\, \boldsymbol{y} \in \mathbb{R}^p, \boldsymbol{Ax} + \boldsymbol{Dy} \geq \boldsymbol{b}\}$ can be obtained by $\boldsymbol{u}$ multipliers that cancel the extended y variables:

Theorem 3.7: *If* $P = \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^n \times \mathbb{R}^p : \boldsymbol{Ax} + \boldsymbol{Dy} \geq \boldsymbol{b}\}$, *then* $Proj_{\boldsymbol{x}}(P) = \{\boldsymbol{x} \in \mathbb{R}^n : (\boldsymbol{uA})x \geq (\boldsymbol{ub}), \forall \boldsymbol{u} \in C_x(P)\}$.

Theorem 3.7 (see Conforti et al. [2010] for its proof) can be applied in different ways. One can use it "offline" during a polyhedral investigation: giving a suitable set of multiplies in $C_x(P)$ it is possible to prove that a certain family of inequalities in the x space (already known or not) is implied by the extended formulation. In principle, one could also use it "on-the-fly", as a separation routine in a BCA. Suppose that we want to find a cutting plane $(\boldsymbol{uA})x \geq (\boldsymbol{ub})$ separating a given point $\boldsymbol{x}^* \in \mathbb{R}^n$ from $Proj_{\boldsymbol{x}}(P)$. This can be done by solving

the following LP:

$$z^* = \min \quad \boldsymbol{u}(\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b}) \tag{3.9a}$$

$$\text{s.t.} \quad \boldsymbol{u}\boldsymbol{D} = \boldsymbol{0} \tag{3.9b}$$

$$\sum_{i\in[m]} u_i = 1 \tag{3.9c}$$

$$\boldsymbol{u} \geq \boldsymbol{0}. \tag{3.9d}$$

If $z^* \geq 0$ then $\boldsymbol{x}^* \in Proj_{\boldsymbol{x}}(P)$, otherwise a violated cut is obtained. Constraint (3.9c) is a possible normalization of the u multipliers, to avoid LP unboundedness. This LP-based separation method can be seen as a particular case of the Benders decomposition (Note 2.10) where only feasibility cuts are separated because there are no costs in the extended variables.

3.17. The power of extended formulations There are cases where extended formulations can be much stronger than natural formulations. For example, consider the following problem:

Definition 3.17: Steiner Problem in Graphs (SPG). Instance: undirected graph $G = (V, E)$, positive edge costs c_e, $e \in E$; and a set of terminal vertices $T \subseteq V$. Solutions: edge-set $E' \subseteq E$ such that the subgraph induced by E' has a path connecting any pair of terminals in T. Goal: Minimize the sum of the costs in E'.

As the costs are positive, the optimal solutions necessarily define trees, known as *Steiner trees*. The SPG has the following natural formulation [Aneja, 1980]:

$$min \quad \sum_{e\in E} c_e x_e \tag{3.10a}$$

$$\text{s.t.} \quad \sum_{e\in\delta(S)} x_e \geq 1 \qquad \forall S \subset V, S \cap T \neq \emptyset, T \setminus S \neq \emptyset \tag{3.10b}$$

$$x_e \in \mathbb{Z}_+ \qquad \forall e \in E. \tag{3.10c}$$

Constraints (3.10b) are known as undirected Steiner Cut inequalities. They

are facet-defining and can be separated in polynomial time. Several other families of facet-defining inequalities were found, including Partition, Odd-Hole, Anti-Hole and Wheel inequalities [Chopra and Rao, 1988a,b], for which heuristic separation procedures were provided. In spite of those efforts, the performance of BCAs using those inequalities is not good. The problem is that the starting formulation (3.10) is too weak. Even the tiny instance where $V = T = \{1, 2, 3\}$, $E = \{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$ and unitary costs already has a positive gap: $z_{\text{LP}} = 1.5$, while $z_{\text{IP}} = 2$. Linear relaxation bounds more than 10% away from the optimal are typical on SteinLIB instances. This leaves a large gap to be closed by the heuristic separation of complex cuts and by branching.

A much stronger SPG formulation is obtained by using an additional set of variables. Define a directed graph $G_D = (V, A)$ where A contains a pair of opposite arcs (i, j) and (j, i) for each edge $e = \{i, j\} \in E$. Choose a vertex $r \in T$. It can be seen that there is a one-to-one correspondence between Steiner trees in G and Steiner arborescences (directed trees) in G_D rooted at r. Define a binary variable y_a for each $a \in A$. The extended formulation by Wong [1984] (equally strong as a multi-commodity flow formulation independently proposed in Claus and Maculan [1983] and also equally strong as another extended formulation proposed in Lucena [1992]) is:

$$\min \quad \sum_{e \in E} c_e x_e \tag{3.11a}$$

$$\text{s.t.} \quad \sum_{e \in \delta^-(S)} y_a \geq 1 \qquad \forall S \subset V, r \notin S, T \cap S \neq \emptyset \tag{3.11b}$$

$$y_a \geq 0 \qquad \forall a \in A \tag{3.11c}$$

$$y_{ij} + y_{ji} = x_e \qquad \forall e = \{i, j\} \in E \tag{3.11d}$$

$$x_e \in \mathbb{Z} \qquad \forall e \in E. \tag{3.11e}$$

Directed Steiner Cut inequalities (3.11b) are much stronger than their undirected counterparts (3.10b). The integrality gaps formulation (3.11) are seldom more than 0.1% on SteinLIB instances (except on artificial instances created with the intent of being hard). In fact, the highly effective SPG codes of today, which are capable of quickly solving many instances with tenths of thousands of vertices, often do not need to separate cuts other than (3.11b);

their algorithmic effort is focused on devising sophisticated dual ascent (fast combinatorial algorithms for obtaining good dual feasible solutions) and graph reductions (based on the propagation of logical implications combined with reduced cost arguments) procedures in order to speed up the solution of that very strong linear relaxation [Koch and Martin, 1998, Polzin and Daneshmand, 2001, Poggi de Aragão et al., 2001, Uchoa et al., 2002, Polzin and Daneshmand, 2009, Rehfeldt and Koch, 2021], see Ljubić [2021] for a recent survey.

The relation between the natural undirected formulation and the extended directed formulation was studied in Goemans [1994]. It was shown that the directed formulation implies (by projection, as discussed in Note 3.16) not only Partition and Odd-Holes inequalities but also a whole zoo of previously unknown facet-defining inequalities for the Steiner polyhedron. Those new families of inequalities can be very complex. It is possible to obtain facet-defining inequalities where coefficients take all integral values between 1 and a chosen odd number. It is remarkable that a simple family of inequalities defined in the extended space, which can even be separated in polynomial time, implies so many facets in the natural space.

The SPG case illustrates the fact that it may be easier to find strong extended formulations than strong natural formulations. A possible intuitive explanation for that is the following: a single variable in an extended formulation may be "richer" in the sense of conveying more information than a single natural variable. In the SPG example, $x_e = 1$ implies that edge $e = \{i, j\}$ belongs to an optimal Steiner tree. However, $y_{ij} = 1$ not only also implies that but also gives the additional information that if e is removed from that tree, vertices r and i will remain in the same subtree. So, edge e may be used for connecting r with vertices in the subtree containing j but not with vertices in the subtree containing i. That information is necessary for deriving the strengthened family of cuts (3.11b).

The existence of a much stronger extended formulation with only two times more variables (the x variables in (3.11) may be eliminated), as happens in the SPG, is very exceptional. Typically, obtaining a significantly stronger extended formulation requires using many times more variables. Yet, there are numerous examples of very large strong extended formulations [Gouveia et al.,

2019] and even pseudo-polynomially large extended formulations (see for examples Martin [1987], Valério de Carvalho [2002], Uchoa [2011], Sadykov and Vanderbeck [2013], de Lima et al. [2022, 2023]) that perform quite well in practice. By the way, the DW reformulation for IP, which will be described in the next chapter, can be viewed as a technique for obtaining stronger extended formulations with exponentially many additional variables.

Exercises

E 3.1. Solve the following IP with the BBA, showing the enumeration tree. Choose the branching variable by the most fractional rule (break ties by largest cost) and explore the tree using the depth-first strategy. Use a package to solve the LPs in each node.

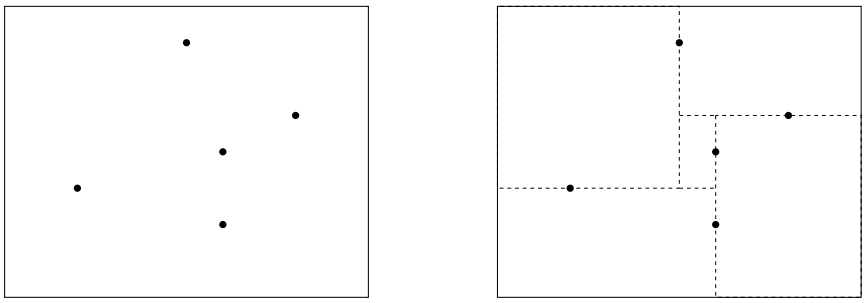
$$
\begin{array}{lrcrcrcrcr}
\max z = & 4x_1 & + & 9x_2 & + & 7x_3 & + & 3x_4 & & \\
\text{s.t.} & 2x_1 & + & 3x_2 & - & x_3 & + & 3x_4 & = & 13 \\
& x_1 & + & 4x_2 & & & & & \leq & 5 \\
& 2x_1 & + & 3x_2 & & & & & \leq & 7 \\
& & & & & 4x_3 & + & 3x_4 & \leq & 20 \\
& & & & & 2x_3 & + & 5x_4 & \leq & 25 \\
& & & & & & & \boldsymbol{x} & \in & \mathbb{Z}^4_+.
\end{array}
$$

E 3.2. Fixing variables with dual feasible solutions. Let UB be the value of the best known solution for an IP in format $\min \boldsymbol{cx}$ subject to $\boldsymbol{Ax} \geq \boldsymbol{b}$, $\boldsymbol{x} \in \mathbb{Z}^n_+$ and let $\boldsymbol{\rho}' \in \mathbb{R}^{1\times m}_+$ be a feasible solution (not necessarily optimal) to the dual of its linear relaxation. Generalize Theorem 3.2 by proving that for every variable x_j, $j \in [n]$, such that $\boldsymbol{\rho}'\boldsymbol{b} + c_j - \boldsymbol{\rho}'\boldsymbol{a}_j \geq UB$ should have value zero in any improving IP solution.

E 3.3. Fixing variables to upper bounds. Consider an IP in format $\min \boldsymbol{cx}$ subject to $\boldsymbol{Ax} \geq \boldsymbol{b}$, $\boldsymbol{x} \leq \boldsymbol{u}$, $\boldsymbol{x} \in \mathbb{Z}^n_+$, where $\boldsymbol{u}$ is an integer vector of variable upper bounds. Let UB be the value of its best known solution. After solving

its linear relaxation, define $gap = UB - z_{\text{LP}}$ (if z_{IP} is known to be integer, $gap = UB - \lceil z_{\text{LP}} \rceil$). Let $\boldsymbol{\theta}^* \in \mathbb{R}_-^{1\times n}$ be the vector with the optimal dual variables corresponding to constraints $\boldsymbol{x} \leq \boldsymbol{u}$. Prove that every variable x_j, $j \in [n]$, such that $-\theta_j^* \geq gap$ should have value u_j in any improving MIP solution. (Modern LP solvers treat variable upper bound constraints implicitly (Note 1.6), so their hidden dual variables θ are obtained from the provided reduced costs as $\theta_j^* = \min\{0, \bar{c}_j^*\}$, $j \in [n]$. It is only possible that $\bar{c}_j^* < 0$ if $x_j^* = u_j$.)

E 3.4. A Rectangular Partition problem. Given a rectangle R in the plane defined by its integer corner points (0 0), (L 0), (L W), and (0 W), and a set P of n integer points in the interior of R, the solutions are partitions of R into subrectangles such that no points in P lie in the strict interior of any subrectangle. The goal is to minimize the total length of the cuts (not necessarily guillotine) required to obtain the rectangular partition. For example, if $L = 10$, $W = 8$, and $P = \{(2\ 3), (5\ 7), (6\ 2), (6\ 4), (8\ 5)\}$, the optimal solution (shown in Figure 3.4) has length 21. Formulate the problem as a SPP having $O(\min\{n^2, LW\})$ rows and $O(\min\{n^4, (LW)^2\})$ variables. If in trouble, check de Meneses and de Souza [2000].



(a) Instance (b) Optimal solution

Figure 3.4: Rectangular Partition instance and its optimal solution

Chapter 4

Dantzig-Wolfe Decomposition and Column Generation for Integer Programming

Dantzig-Wolfe reformulation and column generation play an important role in integer programming, by offering a practical way of obtaining stronger formulations for some very important combinatorial optimization problems. We present branch-and-price and branch-cut-and-price, the enhanced algorithms that arise when combining column generation with standard branch-and-bound and branch-and-cut algorithms.

4.1. Dantzig-Wolfe Reformulation for IP

Consider the following IP:

$$
\begin{aligned}
z_{\mathrm{IP}} \;=\; \min \quad & \boldsymbol{c}\boldsymbol{x} && \text{(4.1a)}\\
\text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && \text{(4.1b)}\\
& \boldsymbol{x} \in P && \text{(4.1c)}\\
& \boldsymbol{x} \in \mathbb{Z}^n, && \text{(4.1d)}
\end{aligned}
$$

where polyhedron P is defined by a set of linear constraints. It is assumed that P is bounded, so the set $Int(P) = P \cap \mathbb{Z}^n$ is finite. Any $\boldsymbol{x} \in Int(P)$ can be described as an integer convex combination of those points: $\boldsymbol{x} = \sum_{\boldsymbol{q} \in Q} \boldsymbol{q}\lambda_{\boldsymbol{q}}$, $\sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} = 1$, $\boldsymbol{\lambda} \in \mathbb{Z}_+^{|Q|}$, where $Q = Int(P)$. That combination is trivial, in the sense that $\lambda_{\boldsymbol{x}}$ should

be 1 and $\lambda_{\boldsymbol{q}}$ should be 0 for any $\boldsymbol{q} \in Q \setminus \{\boldsymbol{x}\}$. Anyway, the IP can be rewritten as:

$$
\begin{aligned}
z_{\text{IP}} \;=\; \min \quad & \boldsymbol{c}\boldsymbol{x} && \text{(4.2a)}\\
\text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && \text{(4.2b)}\\
& \boldsymbol{x} = \sum_{\boldsymbol{q} \in Q} \boldsymbol{q}\lambda_{\boldsymbol{q}} && \text{(4.2c)}\\
& \sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} = 1 && \text{(4.2d)}\\
& \boldsymbol{\lambda} \in \mathbb{Z}_{+}^{|Q|}. && \text{(4.2e)}
\end{aligned}
$$

By eliminating the x variables using (4.2c), the final Dantzig-Wolfe reformulated IP is:

$$
\begin{aligned}
z_{\text{IP}} \;=\; \min \quad & \sum_{\boldsymbol{q} \in Q} (\boldsymbol{c}\boldsymbol{q})\lambda_{\boldsymbol{q}} && \text{(4.3a)}\\
\text{s.t.} \quad & \sum_{\boldsymbol{q} \in Q} (\boldsymbol{A}\boldsymbol{q})\lambda_{\boldsymbol{q}} = \boldsymbol{b} && \text{(4.3b)}\\
& \sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} = 1 && \text{(4.3c)}\\
& \boldsymbol{\lambda} \in \mathbb{Z}_{+}^{|Q|}. && \text{(4.3d)}
\end{aligned}
$$

The Dantzig-Wolfe reformulation for linear programming yields an alternative equivalent LP. However, the Dantzig-Wolfe reformulation for integer programming can produce something different. The following result deserves to be called *the fundamental theorem of column generation for integer programming*:

Theorem 4.1: *The reformulated IP (4.3) is equally strong as:*

$$
\begin{aligned}
z_{IP} \;=\; \min \quad & \boldsymbol{c}\boldsymbol{x} && \text{(4.4a)}\\
s.t. \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && \text{(4.4b)}\\
& \boldsymbol{x} \in Conv(Int(P)) && \text{(4.4c)}\\
& \boldsymbol{x} \in \mathbb{Z}^{n}, && \text{(4.4d)}
\end{aligned}
$$

which can be stronger than the original IP (4.1) if P does not have the integrality property, i.e., if P has fractional extreme points.

Proof. Reformulated IP (4.3) is equivalent to the explicit reformulated IP (4.2). Consider the linear relaxation of the latter. By Definition 2.2, the set $\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} = \sum_{\boldsymbol{q}\in Q} \boldsymbol{q}\lambda_{\boldsymbol{q}}, \sum_{\boldsymbol{q}\in Q} \lambda_{\boldsymbol{q}} = 1, \boldsymbol{\lambda} \geq \boldsymbol{0}\} = Conv(Int(P))$, where $Q = Int(P)$, which proves that (4.2) and (4.3) are equally strong as (4.4). If $Ext(P) \setminus \mathbb{Z}^n = \emptyset$ then $P = Conv(Int(P))$, so (4.3) is equally strong as (4.1). Otherwise, $P \supset Conv(Int(P))$, so (4.3) can be stronger than (4.1). □

Theorem 4.1 indicates that the Dantzig-Wolfe reformulation can be used for strengthening an original formulation, as it implicitly obtains all inequalities defining $Conv(Int(P))$, an operation sometimes referred to as a *partial convexification*, or, more precisely, as the *convexification of* P. As will be seen in some examples in this chapter, sometimes the resulting formulations are a lot stronger. However, this is not for free. The reformulated IP (4.1) usually has a huge number of variables. This in itself is not a problem, since we can use the Column Generation Algorithm for solving its linear relaxation, which defines the following Master LP:

$$
\begin{aligned}
z_{\mathrm{M}} \;=\; \min \quad & \sum_{\boldsymbol{q}\in Q} (\boldsymbol{cq})\lambda_{\boldsymbol{q}} && \text{(4.5a)}\\
\text{s.t.} \quad & \sum_{\boldsymbol{q}\in Q} (\boldsymbol{Aq})\lambda_{\boldsymbol{q}} = \boldsymbol{b} && \text{(4.5b)}\\
& \sum_{\boldsymbol{q}\in Q} \lambda_{\boldsymbol{q}} = 1 && \text{(4.5c)}\\
& \boldsymbol{\lambda} \geq \boldsymbol{0}. && \text{(4.5d)}
\end{aligned}
$$

The pricing subproblem is:

$$
\begin{aligned}
\bar{c}^* \;=\; \min \quad & (\boldsymbol{c} - \boldsymbol{\pi}^*\boldsymbol{A})\boldsymbol{x} - \nu^* && \text{(4.6a)}\\
\text{s.t.} \quad & \boldsymbol{x} \in P && \text{(4.6b)}\\
& \boldsymbol{x} \in \mathbb{Z}^n, && \text{(4.6c)}
\end{aligned}
$$

where $(\boldsymbol{\pi}^*, \nu^*)$ are optimal RMLP solutions ($\boldsymbol{\pi}$ and ν are the dual variables corresponding to (4.5b) and (4.5c), respectively). Here is the potential problem: the pricing subproblem is an IP, which in general is an $\mathcal{NP}$-hard problem.

It is usual to say that Constraints (4.1b) are the ones chosen to be *kept in the master*, while Constraints (4.1c) are the ones chosen to *go to the subproblem* or *to be convexified*. Actually, (4.1b) do not need to only contain linear equations. If there

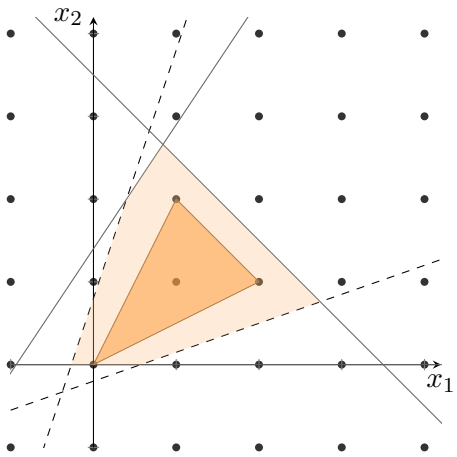
are also linear inequalities in it, the corresponding constraints in (4.3b) and (4.5b) will keep the same sense.

We provide an example of IP reformulation. Consider the following original IP:

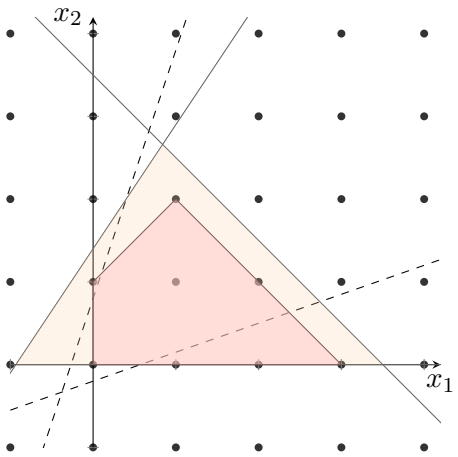
$$\begin{array}{rlrcrcl} z_{\text{IP}} = & \min & -6x_1 & + & 5x_2 & & \\ & \text{s.t.} & -15x_1 & + & 5x_2 & \leq & 4 \\ & & 7x_1 & - & 20x_2 & \leq & 4 \\ & & -15x_1 & + & 10x_2 & \leq & 14 \\ & & 2x_1 & + & 2x_2 & \leq & 7 \\ & & & & x_2 & \geq & 0 \\ & & & & \boldsymbol{x} & \in & \mathbb{Z}^2. \end{array} \tag{4.7}$$

The set of solutions of that IP is $X = \{(0,0),(1,1),(2,1),(1,2)\}$. The optimal integer solution is $\boldsymbol{x} = (2,1)$, with $z_{\text{IP}} = -7$; while its linear relaxation yields $\boldsymbol{x} = (74/27, 41/54) = (2.74, 0.76)$ and $z_{\text{LP}} = -683/54 = -12.65$. Consider a Dantzig-Wolfe reformulation of that IP that keeps the first two inequalities in the master and uses the remaining three inequalities for defining P. It can be seen that $Int(P) = \{(0,0),(1,0),(2,0),(3,0),(0,1),(1,1),(2,1),\ (1,2)\}$ and that $Conv(Int(P)) = \{\boldsymbol{x} \in \mathbb{R}^2 \ : \ -x_1 + x_2 \leq 1,\ x_1 + x_2 \leq 3,\ x_1 \geq 0,\ x_2 \geq 0\}$. By Theorem 4.1, the resulting Master LP will be equivalent to the linear relaxation of the stronger formulation gained by replacing the constraints defining P by the constraints defining $Conv(Int(P))$. By doing that and solving the corresponding LP, we would get $\boldsymbol{x} = (64/27, 17/27) = (2.37, 0.63)$ with $z_{\text{LP}} = -299/27 = -11.07$. So, the reformulation was successful on reducing the gap.

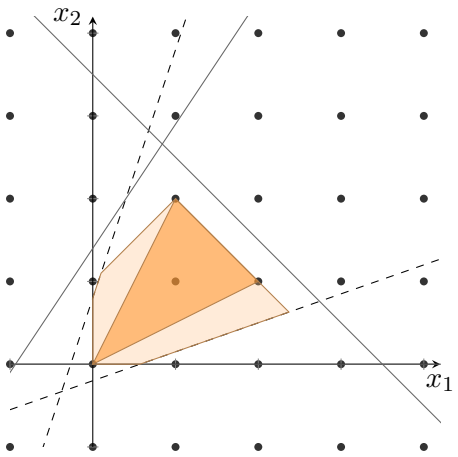
As that example has only two variables, the reformulation procedure can be depicted graphically. Figure 4.1a shows in pale orange the linear relaxation of the original IP, the constraints kept in the Master are dashed. The set $Conv(X)$ is shown in orange. Then, Figure 4.1b shows in pale pink P and in pink $Conv(Int(P))$. Finally, Figure 4.1c shows the strengthened linear relaxation that is obtained when the constraints that define P are replaced by the constraints defining $Conv(Int(P))$. In more realistic cases, it is not possible to perform the formulation strengthening by explicitly computing the linear constraints defining $Conv(Int(P))$. However, the strengthening can be obtained by solving the MLP using the CGA, as will be



(a) Original Formulation



(b) Convexification of $Int(P)$



(c) Strengthened Formulation

Figure 4.1: Dantzig-Wolfe reformulation of an IP

illustrated in our example. The subproblem has the following format:

$$\begin{aligned}
\overline{c}^* = \min \quad & \left((-6 \;\; 5) - \boldsymbol{\pi}^* \begin{pmatrix} -15 & 5 \\ 7 & -20 \end{pmatrix} \right) \boldsymbol{x} - \nu^* \\
& = (-6 + 15\pi_1^* - 7\pi_2^*)x_1 + (5 - 7\pi_1^* + 20\pi_2^*)x_2 - \nu^* \\
\text{s.t.} \quad & -15x_1 + 10x_2 \leq 14 \\
& 2x_1 + 2x_2 \leq 7 \\
& x_2 \geq 0 \\
& x_1 \;\; , \;\; x_2 \in \mathbb{Z}.
\end{aligned}$$

The constraints in that IP are those that define the polyhedron P, plus the integrality constraint. Suppose that one realizes that point $(0\ 0) \in Int(P)$, so the convexity constraint can be relaxed to ≤ 1. There is no need for artificial variables in the first RMLP, that is:

$$\begin{aligned}
z_{\text{RM}} = \quad \min \quad & 0 \\
\text{s.t.} \quad & \leq 4 \\
& \leq 4 \\
& \leq 1.
\end{aligned}$$

By trivially solving this void LP, one gets $z_{\text{RM}} = 0$, $\boldsymbol{\pi}^* = (0\ 0)$ and $\nu^* = 0$. So, the first subproblem IP is:

$$\begin{aligned}
\overline{c}^* = \min \quad & -6x_1 + 5x_2 \\
\text{s.t.} \quad & \boldsymbol{x} \in Int(P).
\end{aligned}$$

Its solution is $\boldsymbol{x}^* = (3\ 0)$, with $\overline{c}^* = -18$. The corresponding column is inserted into the RMLP, which becomes:

$$\begin{aligned}
z_{\text{RM}} = \quad \min \quad & -18\lambda_1 \\
\text{s.t.} \quad & -45\lambda_1 \leq 4 \\
& 21\lambda_1 \leq 4 \\
& \lambda_1 \leq 1 \\
& \lambda_1 \geq 0.
\end{aligned}$$

By optimizing it, one gets $z_{\text{RM}} = -24/7 = -3.42$, $\boldsymbol{\pi}^* = (0\ -6/7)$ and $\nu^* = 0$. The

second subproblem IP is $\overline{c}^* = \min 0x_1 - 85/7x_2$, subject to $\boldsymbol{x} \in Int(P)$. Its solution yields $\boldsymbol{x}^* = (\,1\;\;2\,)$, with $\overline{c}^* = -170/7$. The corresponding column is inserted into the RMLP, which becomes:

$$\begin{array}{rlrcrcrcl} z_{\mathrm{RM}} = & \min & - & 18\lambda_1 & + & 4\lambda_2 & & \\ & \text{s.t.} & - & 45\lambda_1 & - & 5\lambda_2 & \leq & 4 \\ & & & 21\lambda_1 & - & 33\lambda_2 & \leq & 4 \\ & & & \lambda_1 & + & \lambda_2 & \leq & 1 \\ & & & \lambda_1 & , & \lambda_2 & \geq & 0. \end{array}$$

By reoptimizing the RMLP, $z_{\mathrm{RM}} = -299/27 = -11.07$, $\boldsymbol{\pi}^* = (\,0\;\;-11/27\,)$ and $\nu^* = -255/27$. The third subproblem IP is $\overline{c}^* = \min -85/27x_1 - 85/27x_2 + 255/27$, subject to $\boldsymbol{x} \in Int(P)$. One of its optimal solutions is $\boldsymbol{x}^* = (\,3\;\;0\,)$, with $\overline{c}^* = 0$. This proves that the current RMLP is optimal, so $z_{\mathrm{RM}} = z_{\mathrm{M}}$ is a valid lower bound on z_{IP}. Now, the primal solution of the RMLP ($\lambda_1 = 37/54$, and $\lambda_2 = 17/54$) can be used for computing the fractional solution of the improved linear relaxation: $\boldsymbol{x} = 37/54\,(\,3\;0\,) + 17/54\,(\,1\;2\,) = (\,2.37\;0.63\,)$.

Some observations about DW reformulation for integer programming:

- Suppose the following MIP:

$$\begin{array}{rlll} z_{\mathrm{IP}} = \min & \boldsymbol{cx} + \boldsymbol{hy} & & (4.8a) \\ \text{s.t.} & \boldsymbol{Ax} + \boldsymbol{Dy} = \boldsymbol{b} & & (4.8b) \\ & (\boldsymbol{x}, \boldsymbol{y}) \in P & & (4.8c) \\ & \boldsymbol{x} \in \mathbb{Z}^n, & & (4.8d) \end{array}$$

  where $\boldsymbol{y} \in \mathbb{R}^p$ is the vector of variables not required to be integer. A DW reformulation of it keeping (4.8b) in the master (see Exercise E 4.6) would produce a linear relaxation that is equivalent to replacing (4.8c) with $(\boldsymbol{x}, \boldsymbol{y}) \in Conv(\{(\boldsymbol{x}, \boldsymbol{y}) \in P, \boldsymbol{x} \in \mathbb{Z}^n\})$. The pricing subproblems would be MIPs in the following format:

$$\begin{array}{rlll} \overline{c}^* = \min & (\boldsymbol{c} - \boldsymbol{\pi}^*\boldsymbol{A})\boldsymbol{x} + (\boldsymbol{h} - \boldsymbol{\pi}^*\boldsymbol{D})\boldsymbol{y} - \nu^* & & (4.9a) \\ \text{s.t.} & (\boldsymbol{x}, \boldsymbol{y}) \in P & & (4.9b) \\ & \boldsymbol{x} \in \mathbb{Z}^n. & & (4.9c) \end{array}$$

An optimal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ with negative reduced cost generates column $\begin{pmatrix} \boldsymbol{A}\boldsymbol{x}^* + \boldsymbol{D}\boldsymbol{y}^* \\ 1 \end{pmatrix}$ with cost $\boldsymbol{c}\boldsymbol{x}^* + \boldsymbol{h}\boldsymbol{y}^*$ to the MLP.

- The case where an unbounded polyhedron P leads to a set $Int(P)$ with infinity cardinality is too technical and has very little practical interest, so it will not be addressed. As mentioned in Section 2.4, it is usually easy to avoid unbounded subproblems by adding suitable lower and upper bounds $l_j \leq x_j \leq u_j$, for each individual variable x_j, in the subproblem that it appears. This is particularly true for the kind of IPs that we are mostly interested in this book, those that formulate LCOPs. In that context, the variables are non-negative (see Definition 3.4) and are almost always restricted to fairly small ranges.

- As happens with LPs, the most important structure that can be exploited is the case where the pricing decomposes into multiple independent subproblems (a.k.a. the block-diagonal case, see Note 2.8). In that case, either if the subproblems are distinct or identical, the techniques for solving the linear relaxation of the resulting reformulated IPs are similar to those presented in Section 2.3, excepting that each of the subproblems defines an IP.

  But now the structure of the decomposed subproblems has crucial importance. Solving those IPs using general MIP solvers is often too time-consuming. The column generation can work much better if the subproblems define well-structured and relatively simple LCOPs for which specially tailored algorithms do exist. Yet, having LCOPs in $\mathcal{P}$ as subproblems may be less interesting because those problems are likely to admit perfect formulations (see Note 3.14). For example, we know perfect formulations for minimum spanning trees, minimum cost flows (including its particular cases, like the shortest path problem), or minimum cost matchings. This means that if P is not already a perfect formulation for that LCOP, i.e. if P does not have the integrality property, it is possible to replace P with a perfect formulation and get the same improved bounds that would be obtained by the DW reformulation, possibly taking less computational time.

  Indeed, the DW reformulation is often most interesting when the resulting subproblems are LCOPs that are $\mathcal{NP}$-hard but well-solved in practice. In those cases, the implicit convexification provided by the DW reformulation can achieve the improved bounds that could not be practically obtained by

the explicit convexification of $Int(P)$. A typically favorable situation is when the subproblem LCOP is weakly $\mathcal{NP}$-hard, i.e., it can be solved by a pseudo-polynomial algorithm. Yet, there are many examples of successful applications that require solving strongly $\mathcal{NP}$-hard subproblems. The balance "$\mathcal{NP}$-hard yet well-solved" is often subtle and will be a central theme in this book.

We remark that having an $\mathcal{NP}$-hard pricing subproblem in a BPA *is not* similar to having an $\mathcal{NP}$-hard separation subproblem in a BCA. In the latter situation, if the separation subproblem for some families of cuts happens to be intractable, heuristics may be used. When the heuristics fail, some violated cuts may have been missed, but one certainly has a valid bound. There is no need to ever call an exact separation (except perhaps over integer solutions if the constraints are needed to ensure their feasibility; however separation over integer solutions is usually easy, as discussed in Note 5.3). On the other hand, in the CG situation, even with the best possible pricing heuristics, at least one call to the exact pricing is necessary to establish a valid bound (Theorem 2.6 is only valid if $\overline{c}^*$ is indeed the optimum subproblem value). If the call to the exact pricing takes too much time, the whole algorithm fails. As a consequence of that, the pricing subproblem should be *consistently well-solved* for the size of instances that one desires to handle.

As will be seen in the next sections, a very important concern in designing a column generation based algorithm is how the possible branching and cutting operations can affect the potentially delicate structure of the pricing subproblem.

4.2. The Branch-and-Price Algorithm

The Column Generation Algorithm can solve the linear relaxation of a DW reformulated IP. However, in order to find an optimal integer solution, the CGA should be combined with the BBA, in the so-called Branch-and-Price Algorithms (BPAs). The BPAs can be classified as being either robust or non-robust.

Definition 4.1: Robust branching, robust BPA. A branching in a BPA is *robust* if the structure of the pricing subproblems in the resulting children nodes remains unchanged. A branching that forces changes in the pricing structure of any

children node is *non-robust*. A BPA that only performs robust branchings is said to be *robust*, otherwise it is *non-robust*

4.2.1. Branching on the generated variables (non-robust)

The first idea for creating a BPA would be just using the BBA for solving the reformulated IP over the λ variables, with the only difference being that the CGA would solve the LPs in each node of the tree. In fact, the linear relaxation to be solved at the root node corresponds to the Master LP, which the CGA can indeed solve. If its optimal solution $\boldsymbol{\lambda}^*$ is integer, the problem is finished. Otherwise, a variable $\lambda_{\boldsymbol{q}'}$ such that $\lambda^*_{\boldsymbol{q}'}$ is fractional should be selected for branching. Consider the two resulting children nodes:

- If there is a single subproblem, so that MLP has format (4.5), the $\geq$ branching constraint is necessarily $\lambda_{\boldsymbol{q}'} \geq 1$. This means that the resulting MLP can be easily solved by inspection. Either the solution $\lambda_{\boldsymbol{q}'} = 1$ and the remaining variables zero is the only feasible solution (and therefore optimal) or the problem is infeasible.

  However, the case where the reformulation leads to multiple subproblems is not that trivial. Consider a Master LP in format (2.15). In that case, a branching $\lambda^{k'}_{\boldsymbol{q}'} \geq \lceil \lambda^{k'*}_{\boldsymbol{q}'} \rceil$ changes the Master LP to:

$$\min\ z = \sum_{k\in[K]} \sum_{\boldsymbol{q}\in Q^k} (\boldsymbol{c}^k \boldsymbol{q})\lambda^k_{\boldsymbol{q}} + (\boldsymbol{c}^{k'} \boldsymbol{q}')\lceil \lambda^{k'*}_{\boldsymbol{q}'} \rceil \tag{4.10a}$$

$$\text{s.t.} \quad \sum_{k\in[K]} \sum_{\boldsymbol{q}\in Q^k} (\boldsymbol{A}^k \boldsymbol{q})\lambda^k_{\boldsymbol{q}} = \boldsymbol{b} - (\boldsymbol{A}^{k'} \boldsymbol{q}')\lceil \lambda^{k'*}_{\boldsymbol{q}'} \rceil \tag{4.10b}$$

$$\sum_{\boldsymbol{q}\in Q^k} \lambda^k_{\boldsymbol{q}} = U^k \qquad k \in [K],\ k \neq k' \tag{4.10c}$$

$$\sum_{\boldsymbol{q}\in Q^{k'}} \lambda^{k'}_{\boldsymbol{q}} = U^{k'} - \lceil \lambda^{k'*}_{\boldsymbol{q}'} \rceil \tag{4.10d}$$

$$\boldsymbol{\lambda} \geq \boldsymbol{0}. \tag{4.10e}$$

  The CGA algorithm should also be used for solving that MLP. The columns in the parent node RMLP may be used for hot-starting it, but additional

artificial variables may also be needed for initialization. Then, after the CGA finishes, a post-processing step should add the value $\lceil \lambda^{k*}_{\boldsymbol{q}'} \rceil$ to the value of the variable $\lambda^{k*}_{\boldsymbol{q}'}$. The modified MLP (4.10), sometimes referred to as a *residual MLP*, has a structure that is very similar to MLP (2.15), only the RHS of the constraints was changed and a constant term was added to the objective function. This means that the same CGA used in the parent node can solve it, without any change in the pricing subproblem. If the RHS of (4.10d) becomes zero, this means that subproblem k' will simply not be invoked by the CGA for solving the corresponding child node.

Imposing lower bounds to the generated variables in a Master LP can be done without changing the structure of the pricing. By the way, as will be seen in Chapter 6, this fact can be exploited for creating efficient *diving heuristics*.

- If there is a single subproblem, so that MLP has format (4.5), the $\leq$ branching constraint is necessarily $\lambda_{\boldsymbol{q}'} \leq 0$. This means that variable $\lambda_{\boldsymbol{q}'}$ should be removed from the RMLP in the child node. But this is not enough. The suppressed variable $\lambda_{\boldsymbol{q}'}$ will have a negative reduced cost and will probably be generated again by the CGA. This may only not happen if some other generated variables make z_{RM} drop to exactly the same z value of the parent node. In any case, the effect of that branching will be null. The only way of avoiding this is to include the constraint $\boldsymbol{x} \neq \boldsymbol{q'}$ in pricing subproblem (2.17). Here we have a potentially serious issue. As said before, many effective column generation based algorithms have pricing subproblems that correspond to $\mathcal{NP}$-hard LCOPs that can be reasonably well-solved by specific algorithms, like, for example, a pseudo-polynomial Dynamic Programming Algorithm. The additional constraints for enforcing that $\boldsymbol{x} \neq \boldsymbol{q'}$ break the structure of the LCOP and require algorithmic adaptations that may make the pricing significantly less efficient (not to mention the effort of having to understand and adapt complex codes that may have been produced by third-parties). The problem is often cumulative. A few such "bad branchings" may still be well-handled by the adapted pricing, but too many of them may make it so slow that the whole BPA may fail.

  Consider the case with multiple subproblems and a Master LP in format

(2.15). In that case, a branching $\lambda^{k'}_{\boldsymbol{q}'} \leq \lfloor \lambda^{k'*}_{\boldsymbol{q}'} \rfloor$ changes the Master LP to:

$$\min\ z = \sum_{k\in[K]} \sum_{\substack{\boldsymbol{q}\in Q^k \\ \text{if } k=k', \\ \boldsymbol{q}'\neq\boldsymbol{q}'}} (\boldsymbol{c}^k\boldsymbol{q})\lambda^k_{\boldsymbol{q}} + (\boldsymbol{c}^{k'}\boldsymbol{q}') \qquad \lambda^{k'}_{\boldsymbol{q}'} \tag{4.11a}$$

$$\text{s.t.} \quad \sum_{k\in[K]} \sum_{\substack{\boldsymbol{q}\in Q^k \\ \text{if } k=k', \\ \boldsymbol{q}'\neq\boldsymbol{q}'}} (\boldsymbol{A}^k\boldsymbol{q})\lambda^k_{\boldsymbol{q}} + (\boldsymbol{A}^k\boldsymbol{q}')\ \lambda^{k'}_{\boldsymbol{q}'} = \boldsymbol{b} \tag{4.11b}$$

$$\sum_{\boldsymbol{q}\in Q^k} \lambda^k_{\boldsymbol{q}} = U^k \qquad k \in [K], k \neq k' \tag{4.11c}$$

$$\sum_{\substack{\boldsymbol{q}\in Q^{k'} \\ \boldsymbol{q}'\neq\boldsymbol{q}'}} \lambda^{k'}_{\boldsymbol{q}} + \lambda^{k'}_{\boldsymbol{q}'} = U^{k'} \tag{4.11d}$$

$$\lambda^{k'}_{\boldsymbol{q}'} \leq \lfloor \lambda^{k'*}_{\boldsymbol{q}'} \rfloor \tag{4.11e}$$

$$\boldsymbol{\lambda} \geq \boldsymbol{0}. \tag{4.11f}$$

This means that, unless $\lfloor \lambda^{k'*}_{\boldsymbol{q}'} \rfloor = 0$, variable $\lambda^{k'}_{\boldsymbol{q}'}$ should be kept in the RMLP in the child node, but with upper bound (4.11e). The pricing subproblem k' should be changed for including the constraint $\boldsymbol{x}^{k'} \neq \boldsymbol{q}'$. If this is not done, a second copy of the variable $\lambda^{k'}_{\boldsymbol{q}'}$ will be generated and the branching will be nullified. Note that the dual variable of (4.11e) does not appear in any pricing subproblem, as all possible generated variables have coefficient zero in it.

Imposing upper bounds to the generated variables in a Master LP changes the structure of the pricing. Therefore, branching schemes that do that are non-robust.

Branching over individual λ variables has a second drawback: it usually leads to very unbalanced BBA trees. Consider for example a branching of format $\lambda^{k'}_{\boldsymbol{q}'} \geq 1$ or $\lambda^{k'}_{\boldsymbol{q}'} \leq 0$. The first branching (besides not changing the pricing) is likely to be good, in the sense of moving z values significantly. On the other hand, the second branching (besides making the pricing harder) is not likely to be good. This happens because in realistic problems there is a huge number of λ variables. Fixing a single such variable to zero barely changes the problem and the z values probably will not move significantly. Therefore, as will be seen later in this chapter and also in Chapter 8, other more balanced schemes that branch over linear expressions over the generated λ variables were proposed. Yet, those schemes are still non-robust:

the pricing is changed in at least one of the children nodes.

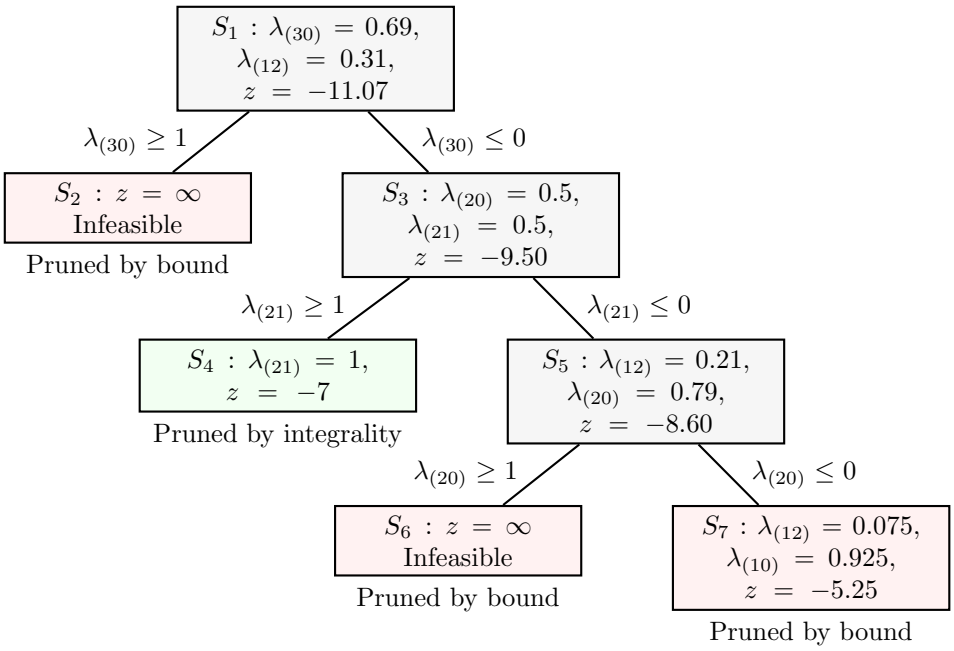


Figure 4.2: BPA tree with branching over λ variables (non-robust)

A BPA tree solving the example in page 114 is shown in Figure 4.2. When branching on λ variables it is recommended to explore the $\geq$ child node before the $\leq$ child. Not only the CGA in the $\geq$ child is likely to be easier, but the chances of quickly finding an integer solution are much larger. Node S_2 is easily solved by inspection. As the point $(\,3\;0\,)$ does not satisfy the constraints kept in the master, the node is infeasible. In order to solve S_3, the constraint $\boldsymbol{x} \neq (\,3\;0\,)$ should be added to the subproblem IP. When $Int(P)$ only contains binary vectors, a constraint like $\boldsymbol{x} \neq \boldsymbol{q}'$ can be enforced by the linear inequality $\sum_{j\in[n]} q'_j x_j + \sum_{j\in[n]}(1-q'_j)(1-x_j) \leq n-1$, where q'_j is the j-th element of $\boldsymbol{q}'$. If this is not the case, the linear inequality should be produced by considering the particular structure of $Int(P)$. In the case of S_3, the constraint $x_1 \leq 2$ cuts $(\,3\;0\,)$ but is valid for all other points in $Int(P)$. Node S_4 is also solved by inspection and finds the integer solution $\lambda_{(21)} = 1$, with value $z = -7$. Node S_5 corresponds to S_3 plus $\boldsymbol{x} \neq (\,2\;1\,)$, the linear inequality $2x_1 + x_2 \leq 4$ can enforce that. Node S_6 is infeasible. In S_7, the additional constraint $\boldsymbol{x} \neq (\,2\;0\,)$ can be enforced by $x_1 \leq 1$. As the CGA in that node finishes with

$z = -5.25$, S_7 is pruned by bound.

4.2.2. Branching on the original variables (robust)

As seen at the beginning of this chapter, the original IP (4.1) is equivalent to the explicit reformulated IP (4.2). However, it is also equivalent to

$$
\begin{aligned}
z_{\text{IP}} \;=\; \min \quad & \boldsymbol{cx} && \text{(4.12a)}\\
\text{s.t.} \quad & \boldsymbol{Ax} = \boldsymbol{b} && \text{(4.12b)}\\
& \boldsymbol{x} = \sum_{\boldsymbol{q}\in Q} \boldsymbol{q}\lambda_{\boldsymbol{q}} && \text{(4.12c)}\\
& \sum_{\boldsymbol{q}\in Q} \lambda_{\boldsymbol{q}} = 1 && \text{(4.12d)}\\
& \boldsymbol{\lambda} \geq \boldsymbol{0} && \text{(4.12e)}\\
& \boldsymbol{x} \in \mathbb{Z}^n. && \text{(4.12f)}
\end{aligned}
$$

This means that it is possible to find the optimal solution for the original IP by only enforcing that the original variables x are integer. So, given a solution $\boldsymbol{\lambda}^*$ to the Master LP (4.5), we may use (4.12c) to convert it to a solution $\boldsymbol{x}^*$. If $\boldsymbol{x}^*$ is integer then it is also the optimal solution for (4.1) and the problem is finished. Otherwise, a variable x_j such that x_j^* is fractional should be selected for branching. Consider the branching constraints $x_j \geq \lceil x_j^* \rceil$ and $x_j \leq \lfloor x_j^* \rfloor$. Using (4.12c) again, they can be translated to equivalent constraints $\sum_{\boldsymbol{q}\in Q} q_j \lambda_{\boldsymbol{q}} \geq \lceil x_j^* \rceil$ and $\sum_{\boldsymbol{q}\in Q} q_j \lambda_{\boldsymbol{q}} \leq \lfloor x_j^* \rfloor$, respectively, over the λ variables. Those new branching constraints will introduce new dual variables to the MLPs. However, those dual variables do no harm to the pricing subproblem, since they only affect its objective function, not its structure.

The procedure will be illustrated in our example. A BPA tree using branching over the x variables is shown in Figure 4.3. In Node S_1, the converted solution is $\boldsymbol{x}^* = 37/54\,(\,3\;0\,) + 17/54\,(\,1\;2\,) = (\,2.37\;\;0.63\,)$. Suppose that x_2 is chosen for the first branching. Node S_2, corresponding to $x_2 \geq 1$ would start with the following

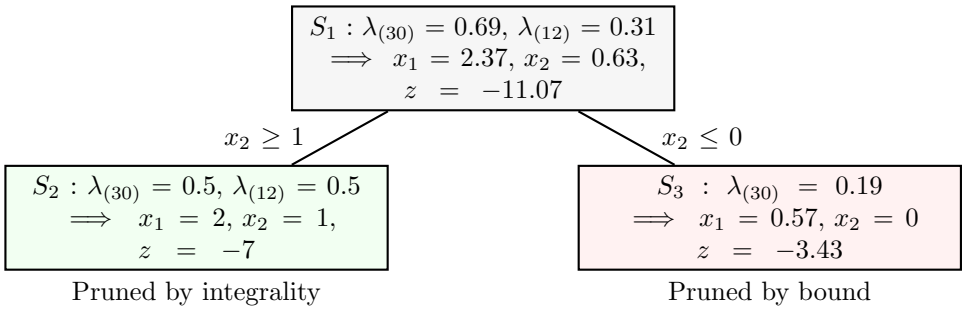


Figure 4.3: BPA tree with branching over x variables (robust)

RMLP:

$$\begin{array}{rlrrrrrrrl} z_{\mathrm{RM}} = & \min & 99a_1 & - & 18\lambda_1 & + & 4\lambda_2 & & \\ & \text{s.t.} & & - & 45\lambda_1 & - & 5\lambda_2 & \leq & 4 \\ & & & & 21\lambda_1 & - & 33\lambda_2 & \leq & 4 \\ x_2 \geq 1 \quad \triangleright & & a_1 & & & + & 2\lambda_2 & \geq & 1 \\ & & & & \lambda_1 & + & \lambda_2 & \leq & 1 \\ & & a_1, & & \lambda_1, & & \lambda_2 & \geq & 0. \end{array}$$

Variables λ_1 and λ_2 are inherited from the parent node. The third constraint is the branching constraint, as indicated in the remark. In that particular case, it is not really necessary, but it is recommended to include an artificial variable in any newly added constraint to make sure that the resulting RMLP is still feasible. By solving it, we get $z_{\mathrm{RM}} = -7$. The primal solution is $\lambda_1 = \lambda_2 = 0.5$ and the dual solution is $\boldsymbol{\pi}^* = (\,0 \;\; 0 \;\; 11\,)$ and $\nu^* = -18$. Now we have a new dual variable π_3, corresponding to the branching constraint. How to deal with that dual variable in the pricing subproblem? Exactly as if $x_2 \geq 1$ was part of the original IP and kept in the master. So, the subproblem becomes:

$$\begin{aligned} \overline{c}^* = \min \quad & \left((\,-6 \;\; 5\,) - \boldsymbol{\pi}^* \begin{pmatrix} -15 & 5 \\ 7 & -20 \\ 0 & 1 \end{pmatrix} \right) \boldsymbol{x} - \nu^* \\ & = \; (-6 + 15\pi_1^* - 7\pi_2^*)x_1 + (5 - 7\pi_1^* + 20\pi_2^* - \pi_3^*)x_2 - \nu^* \\ \text{s.t.} \quad & \boldsymbol{x} \in Int(P). \end{aligned}$$

This means that a branching constraint over the x variables only affects the objective

function of the subproblem, not its structure. In our example, the first pricing IP on node S_2 is $\overline{c}^* = \min -6x_1 - 6x_2 + 18$, subject to $\boldsymbol{x} \in Int(P)$. One of its optimal solutions is $\boldsymbol{x}^* = (\,1\;2\,)$, with $\overline{c}^* = 0$. Therefore, the current RMLP solves the node. The solution $\boldsymbol{x}^* = 0.5\,(\,3\;0\,) + 0.5\,(\,1\;2\,) = (\,2\;1\,)$ is integer, so the node is pruned by integrality. The node S_3 corresponding to $x_2 \leq 0$ is solved in a similar way. The first RMLP in that node is:

$$
\begin{array}{rrrrrrrrrrr}
z_{\mathrm{RM}} = & \min & & 99a_1 & - & 18\lambda_1 & + & 4\lambda_2 & & \\
& \text{s.t.} & & & - & 45\lambda_1 & - & 5\lambda_2 & \leq & 4 \\
& & & & & 21\lambda_1 & - & 33\lambda_2 & \leq & 4 \\
x_2 \leq 0 \quad \triangleright & & - & a_1 & & & + & 2\lambda_2 & \leq & 0 \\
& & & & & \lambda_1 & + & \lambda_2 & \leq & 1 \\
& & & a_1, & & \lambda_1, & & \lambda_2 & \geq & 0.
\end{array}
$$

Note that the artificial variable has coefficient -1 in the new constraint, making sure that the RMLP is feasible. By solving it, $z_{\mathrm{RM}} = -3.43$. The primal solution is $\lambda_1 = 52/273 = 0.19$, $\lambda_2 = 0$ and the dual solution is $\boldsymbol{\pi}^* = (\,0 \;\; -6/7 \;\; -85/7\,)$ and $\nu^* = 0$. The pricing IP is $\overline{c}^* = \min 0x_1 + 0x_2$, subject to $\boldsymbol{x} \in Int(P)$, which obviously leads to $\overline{c}^* = 0$. Therefore, the current RMLP is optimal, and $z = -3.43$ is the node value. So, S_3 is pruned by bound.

In any case, branching over the original x variables (or over their linear expressions, see Note 3.11) can be done without changing the structure of the pricing. Therefore, a BPA that only uses such kind of branching is robust.

If the branching over the x variables has such a nice property, the reader may be wondering why one would ever want to design a BPA algorithm that performs non-robust branchings over the λ variables. The issue is actually more complex, as will be seen next:

- On some important LCOPs, as in the Cutting Stock Problem discussed in Section 4.4.2, the original IP formulation is highly symmetrical and leads to the case where the DW reformulation produces a large number U of identical subproblems. As discussed on page 32, that kind of symmetry means that the same $\boldsymbol{\lambda}^*$ solution can be converted into many possible alternative $\boldsymbol{x}^* = (\boldsymbol{x}^{1*} \;\ldots\; \boldsymbol{x}^{U*})$ solutions, by only permuting indices. This means that branching over a particular x_j^u variable is almost useless (moreover, such a branching constraint would make subproblem u distinct from the other subproblems

in its group, creating the need for solving additional pricing subproblems). Therefore, in those cases, it may be necessary to perform non-robust branching over the λ variables.

Yet, effective robust branching may be possible in *some* cases where the DW reformulation produces identical subproblems. This may happen if it is possible to branch on the aggregated original y variables defined in (2.19c). As the conversion of $\boldsymbol{\lambda}^*$ to $\boldsymbol{y}^*$ given in (2.20) is unique, there are no problems with symmetric alternative solutions. Moreover, the subproblems remain identical and a single subproblem needs to be solved. Such a good situation happens in the CVRP, as discussed in Section 4.4.3.

- There are cases where it is possible to perform effective robust branching over the x variables, but non-robust branching over *linear expressions over* λ *variables* can be even more effective and lead to smaller BPA trees. In those cases, the algorithm designer should be careful, avoiding the modified pricing subproblems becoming too hard for the size of instances that one desires to handle.

As discussed in Note 4.3, sometimes it is possible to robustly include branching constraints over the original variables in the subproblems.

4.3. The Branch-Cut-and-Price Algorithm

Given the potential of DW reformulation and column generation for strengthening formulations, it is natural to look for ways of obtaining even stronger formulations by also adding cutting planes, in the so-called Branch-Cut-and-Price Algorithms (BCPAs). As happens with branchings, there are two kinds of cuts, robust and non-robust ones.

Definition 4.2: Robust Cut, robust BCPA. A cutting plane in a BCPA is *robust* if it does not require any change in the structure of the pricing subproblems. A cutting plane that does force changes in the pricing structure is *non-robust*. A BCPA that only performs robust branchings and only separates robust cuts is said to be *robust*, otherwise, it is *non-robust*.

4.3.1. Cuts on the generated variables (non-robust)

Given a solution $\boldsymbol{\lambda}^*$ to the Master LP (4.5), one may separate a cutting plane having the following general format:

$$\sum_{\boldsymbol{q}\in Q} \alpha(\boldsymbol{q})\lambda_{\boldsymbol{q}} \geq \alpha_0, \tag{4.13}$$

where coefficients $\alpha(\boldsymbol{q})$ are given by an arbitrary function that maps points in $Int(P)$ to $\mathbb{R}$. Adding such inequality to the MLP creates a new dual variable that will be denoted by σ. Now, the new MLP should be solved by a CGA in which the pricing subproblem has format:

$$\begin{aligned} \overline{c}^* \;=\; \min \quad & (\boldsymbol{c} - \boldsymbol{\pi}^*\boldsymbol{A})\boldsymbol{x} - \sigma^*\alpha(\boldsymbol{x}) - \nu^* && \text{(4.14a)} \\ \text{s.t.} \quad & \boldsymbol{x} \in P && \text{(4.14b)} \\ & \boldsymbol{x} \in \mathbb{Z}^n, && \text{(4.14c)} \end{aligned}$$

where $\boldsymbol{\pi}$ and ν are the dual variables corresponding to (4.5b) and (4.5c), respectively. This means that a newly added cutting plane defined over the λ variables introduces a non-linear term in the objective function of the pricing, breaking its original structure and forcing algorithmic adaptations that may make it much less efficient. The problem is actually cumulative. Each non-robust cut will add a non-linear term in (4.14a). Some non-robust cuts may still be well-handled by the adapted pricing, but too many of them may make it so slow that the whole BCPA may fail. The introduction of many complex non-linear terms in the objective function has all the potential for turning a quite well-solved (even if $\mathcal{NP}$-hard) LCOP into a nightmare COP.

Let us illustrate the use of such kind of cut in our example. Consider the final RMLP shown on page 117. Using the Chvátal-Gomory procedure (Note 3.7) with multipliers $\rho = (0\ 0.09\ 0.6)$ (the second constraint and the convexity constraint have positive multipliers), the violated inequality $2\lambda_1 - 3\lambda_2 \leq 0$ is obtained. In general, a violated cut can be separated from the RMLP structure. However, it can only be used in a BCPA if it has well-defined coefficients $\alpha(\boldsymbol{x})$ for any point in $\boldsymbol{x} \in Int(P)$. This always happens for cuts obtained from the Chvátal-Gomory procedure. In our example, a point (x_1, x_2) has coefficient $\lfloor 0.09(7x_1 - 20x_2) + 0.6 \rfloor$. The new RMLP

becomes:

$$\begin{array}{rlrrrrrrrrl}
z_{\text{RM}} = & \min & + & 99a_1 & - & 18\lambda_1 & + & 4\lambda_2 & & \\
& \text{s.t.} & & & - & 45\lambda_1 & - & 5\lambda_2 & \leq & 4 \\
& & & & & 21\lambda_1 & - & 33\lambda_2 & \leq & 4 \\
\text{CG cut} \quad \triangleright & & & a_1 & + & 2\lambda_1 & - & 3\lambda_2 & \leq & 0 \\
& & & & & \lambda_1 & + & \lambda_2 & \leq & 1 \\
& & & a_1, & & \lambda_1, & & \lambda_2 & \geq & 0.
\end{array}$$

Whenever a cut is added to an RMLP it is advisable to also add an artificial variable (not actually needed in this case) to make sure that the resulting RMLP is still feasible. The new optimal solution is $\lambda_1 = 0.6$, $\lambda_2 = 0.4$, and $z_{\text{RM}} = -9.2$, with dual solution is $\boldsymbol{\pi}^* = (\,0\;0\,)$, $\sigma^* = -4.4$, and $\nu^* = -9.2$. The new subproblem should take into account the non-linear (due to the floor function) effect of the new variable σ and becomes: $\overline{c}^* = \min -6x_1 + 5x_2 + 4.4\lfloor 0.63x_1 - 1.8x_2 + 0.6 \rfloor + 9.2$ s.t. $\boldsymbol{x} \in Int(P)$. This is equivalent to the following IP:

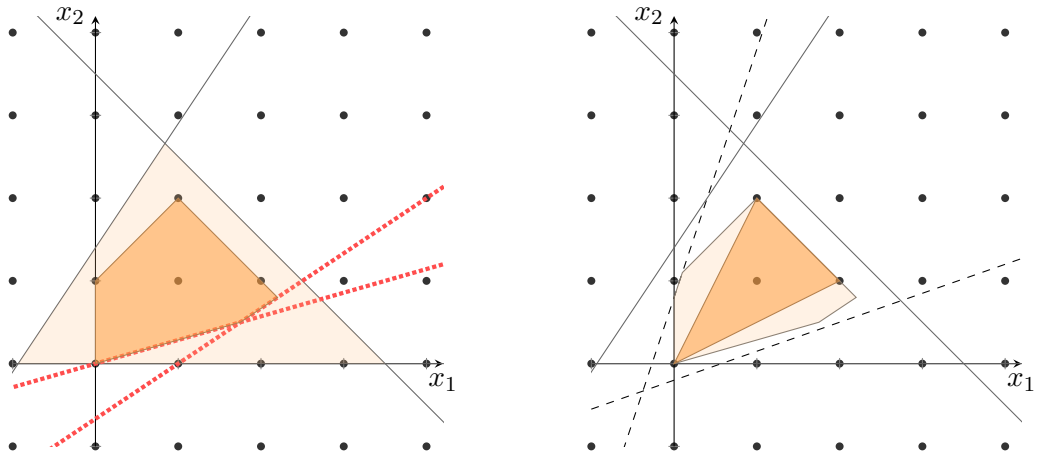
$$\begin{array}{rlr}
\overline{c}^* = & \min & -6x_1 + 5x_2 + 4.4w + 9.2 \\
& \text{s.t.} & \boldsymbol{x} \in Int(P) \\
& & w \geq (0.63x_1 - 1.8x_2 + 0.6) - 1 + \epsilon \\
& & w \in \mathbb{Z},
\end{array}$$

where ϵ is a small constant and w is an auxiliary integer variable devised to assume the coefficient of the cut in the point (x_1, x_2) (the linearization trick works because w has a positive cost in the objective function). By solving it, one of the optimal solutions is $\boldsymbol{x}^* = (\,3\;\;0\,)$ and $w^* = 2$, with $\overline{c}^* = 0$. So, the column generation stops. The MLP solution corresponds in the original variables to the point $\boldsymbol{x} = 0.6\,(\,3\;0\,) + 0.4\,(\,1\;2\,) = (\,2.2\;0.8\,)$.

It is interesting to analyze how a cut defined over the generated variables can strengthen a formulation. They do that by restricting the possible convex combinations of the points in $Int(P) = Q$. Consider our example. The set $P' = \{\boldsymbol{x} \in \mathbb{R}^2 : \boldsymbol{x} = \sum_{\boldsymbol{q} \in Q} \boldsymbol{q}\lambda_{\boldsymbol{q}}, \sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} = 1, \boldsymbol{\lambda} \geq \boldsymbol{0}, \lambda_{(10)} + \lambda_{(20)} + 2\lambda_{(30)} - 2\lambda_{(01)} - \lambda_{(11)} - 3\lambda_{(12)} \leq 0\} \subset Conv(Int(P))$, which can be computed by the Fourier-Motzkin elimination method (see Note 3.16), is depicted in Figure 4.4a (compare with Figure 4.1b). The strengthened formulation, shown in Figure 4.4b (compare with Figure 4.1c), is the

intersection of P' with the constraints kept in the Master. Some remarks:

- As always happens with any bounded polyhedron, $Ext(Conv(Int(P))) \subseteq Int(P)$. However, $Ext(P') = \{(0\ 0), (0\ 1), (1\ 2), (2.2\ 0.8), (1.75\ 0.5)\}$ has fractional points.
- The added cut implies two new inequalities, $2x_1 - 7x_2 \leq 0$ and $2x_1 - 3x_2 \leq 2$, that define distinct facets of P' (therefore, one can not dominate the other). In general, a single cut over the generated variables λ can imply a set of non-dominated cuts in the original space x (see Note 4.5). This is an indication of the potential for that kind of cut.



(a) Convexification of $Int(P)$ restricted by a single cut over the λ variables. Two implied inequalities are shown as dotted red lines

(b) Resulting Strengthened Formulation

Figure 4.4: Effect of a cut over the generated variables

It would be possible to branch at that point of the algorithm. However, we will finish the non-robust BCPA by adding a second cut. Using again the Chvátal-Gomory procedure with multipliers $\rho = (0\ 0\ 0.5\ 0.5)$ (the previous cut and the convexity constraint have positive multipliers), a violated inequality $\lambda_1 - \lambda_2 \leq 0$ for the RMLP is obtained. Again, it is necessary to determine its coefficients $\alpha(\boldsymbol{x})$ for any point in $\boldsymbol{x} \in Int(P)$. A Chvátal-Gomory cut of rank 2 has "more non-linear" coefficients since the floor function is applied twice. In this example, a point (x_1, x_2)

has coefficient $\lfloor 0.5 \lfloor 0.09(7x_1 - 20x_2) + 0.6 \rfloor + 0.5 \rfloor$. The new RMLP becomes:

$$
\begin{array}{rlrrrrrrrrl}
z_{\mathrm{RM}} = & \min & + & 99a_1 & - & 18\lambda_1 & + & 4\lambda_2 & & \\
& \text{s.t.} & & & - & 45\lambda_1 & - & 5\lambda_2 & \leq & 4 \\
& & & & & 21\lambda_1 & - & 33\lambda_2 & \leq & 4 \\
& & & & & 2\lambda_1 & - & 3\lambda_2 & \leq & 0 \\
\text{rank-2 CG cut} \quad \triangleright & & & a_1 & + & \lambda_1 & - & \lambda_2 & \leq & 0 \\
& & & & & \lambda_1 & + & \lambda_2 & \leq & 1 \\
& & & a_1, & & \lambda_1, & & \lambda_2 & \geq & 0.
\end{array}
$$

The new optimal solution is $\lambda_1 = 0.5$, $\lambda_2 = 0.5$, and $z_{\mathrm{RM}} = -7$, with dual solution is $\boldsymbol{\pi}^* = (\,0\ 0\,)$, $\boldsymbol{\sigma}^* = (\,0\ \ -11\,)$, and $\nu^* = -7$. The changed subproblem becomes: $\overline{c}^* = \min -6x_1 + 5x_2 + 11\lfloor 0.5 \lfloor 0.09(7x_1 - 20x_2) + 0.6 \rfloor + 0.5 \rfloor + 7$ s.t. $\boldsymbol{x} \in Int(P)$. This is equivalent to the following IP:

$$
\begin{array}{rll}
\overline{c}^* = & \min & -6x_1 + 5x_2 + 11w_2 + 7 \\
& \text{s.t.} & \boldsymbol{x} \in Int(P) \\
& & w_1 \geq (0.63x_1 - 1.8x_2 + 0.6) - 1 + \epsilon \\
& & w_2 \geq (0.5w_1 + 0.5) - 1 + \epsilon \\
& & \boldsymbol{w} \in \mathbb{Z}^2.
\end{array}
$$

By solving it, one of its three optimal solutions is $\boldsymbol{x}^* = (\,3\ 0\,)$ and $\boldsymbol{w}^* = (\,2\ 1\,)$, with $\overline{c}^* = 0$. So, the column generation stops. The optimal IP solution is $\boldsymbol{x} = 0.5\,(\,3\ 0\,) + 0.5\,(\,1\ 2\,) = (\,2\ 1\,)$. By the way, the second cut dominates the first cut and implies both $x_1 + 3x_2 \leq 0$ and $x_1 - x_2 \leq 1$ in the original space. The BCPA tree is shown in Figure 4.5. The single node S_1 is depicted in more detail: $S_{1.1}$ is the original linear relaxation; $S_{1.2}$ is the improved linear relaxation after the first cut is added, and $S_{1.3}$ is the further improved linear relaxation after the second cut is added.

4.3.2. Cuts on the original variables (robust)

Given a solution $\boldsymbol{\lambda}^*$ to the Master LP (4.5), we may use (4.12c) to convert it to a solution $\boldsymbol{x}^*$. If $\boldsymbol{x}^*$ is fractional we may separate a valid inequality $\boldsymbol{\alpha x} \geq \alpha_0$ cutting

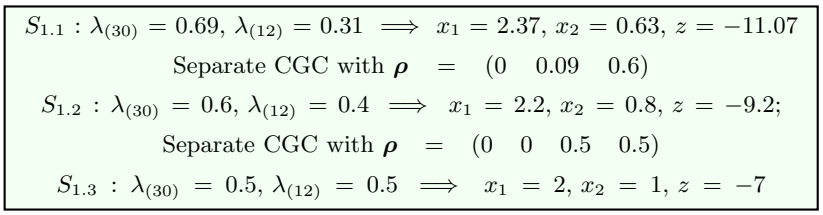


Figure 4.5: Non-robust BCPA tree

that point. Using (4.12c) again, it can be translated to an equivalent inequality

$$\sum_{\boldsymbol{q}\in Q}\left(\sum_{j\in[n]}\alpha_j q_j\right)\lambda_{\boldsymbol{q}} \geq \alpha_0 \tag{4.15}$$

over the λ variables. As also happens with constraints directly defined over the λ variables, this new constraint will also introduce a new dual variable to the MLP. However, we have here a very particular case. By comparing (4.15) and (4.13), we see that $\alpha(\boldsymbol{q})$ is given by the linear function $\sum_{j\in[n]}\alpha_j q_j$. This means that the additional term in (4.14a) will be linear and can be easily incorporated into the first term (the dual variable of the new cut is included in the $\boldsymbol{\pi}$ vector and the $\boldsymbol{\alpha}$ coefficients are included as an additional row in matrix $\boldsymbol{A}$). Everything happens as if $\boldsymbol{\alpha x} \geq \alpha_0$ was part of the original IP and kept in the master. So, there is no change in the pricing structure and the cut is robust.

The procedure will be also illustrated in our example. Let $\boldsymbol{x}^* = 37/54\,(3\ 0) + 17/54\,(1\ 2) = (2.37\ 0.63)$ be the fractional solution after the first CGA convergence. Suppose that the cutting plane $2x_1 - 3x_2 \leq 1$ is somehow (perhaps from the structure of the original IP) separated. The corresponding RMLP would become:

$$\begin{array}{rrrrrrrrrrl}
z_{\mathrm{RM}} = & \min & + & 99a_1 & - & 18\lambda_1 & + & 4\lambda_2 & & \\
& \text{s.t.} & & & - & 45\lambda_1 & - & 5\lambda_2 & \leq & 4 \\
& & & & & 21\lambda_1 & - & 33\lambda_2 & \leq & 4 \\
2x_1 - 3x_2 \leq 1 \quad \triangleright & & & a_1 & + & 6\,\lambda_1 & - & 4\lambda_2 & \geq & 1 \\
& & & & & \lambda_1 & + & \lambda_2 & \leq & 1 \\
& & & a_1, & & \lambda_1, & & \lambda_2 & \geq & 0.
\end{array}$$

By solving it, we get $z_{\mathrm{RM}} = -7$. The primal solution is $\lambda_1 = \lambda_2 = 0.5$ and the dual

solution is $\boldsymbol{\pi}^* = (\,0\;0\;\;-2.2\,)$ and $\nu^* = -4.8$. Now we have a new dual variable π_3, corresponding to the added cutting plane. The subproblem becomes:

$$\begin{aligned}\overline{c} = \quad \min \quad & \left((\,-6\;\;5\,) - \boldsymbol{\pi}^* \begin{pmatrix} -15 & 5 \\ 7 & -20 \\ 2 & -3 \end{pmatrix} \right) \boldsymbol{x} - \nu^* \\ & = (-6 + 15\pi_1^* - 7\pi_2^* - 2\pi_3^*)x_1 + (5 - 7\pi_1^* + 20\pi_2^* + 3\pi_3^*)x_2 - \nu \\ \text{s.t.} \quad & \boldsymbol{x} \in Int(P).\end{aligned}$$

So, the first pricing IP after the cut becomes $\overline{c}^* = \min -1.6x_1 - 1.6x_2 + 4.8$, subject to $\boldsymbol{x} \in Int(P)$. One of its optimal solutions is $\boldsymbol{x}^* = (\,1\;2\,)$, with $\overline{c}^* = 0$. Therefore, the current RMLP solves the Master LP. The solution $\boldsymbol{x}^* = 0.5\,(\,3\;0\,) + 0.5\,(\,1\;2\,) = (\,2\;1\,)$ is integer, so the problem is solved at the root node, without need for branching. We depict this BCPA tree in Figure 4.6. There is a single node S_1, but $S_{1.1}$ is the original linear relaxation, solved by the CGA, and $S_{1.2}$ is the improved linear relaxation after cutting plane $2x_1 - 3x_2 \leq 1$ is added, also solved by the CGA.

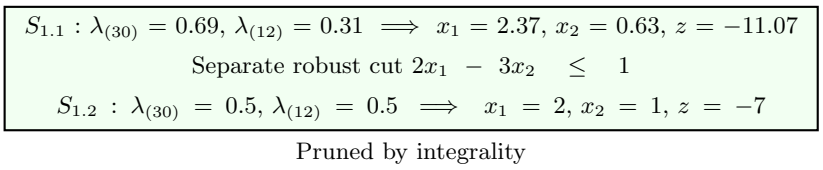


Figure 4.6: Robust BCPA tree

Cutting over the original variables can be done without changing the structure of the pricing. Therefore, a BCPA that only uses such kind of cuts and only performs robust branching is robust. Again, given the big potential problems with the pricing caused by non-robust cuts, the reader may be wondering why one would ever want to design a BCPA algorithm that uses non-robust cuts.

- When the original IP formulation is highly symmetrical and, in particular, the case where the DW reformulation produces U identical subproblems, cutting planes over the original x variables are essentially useless. Such a cutting plane will only make a fractional solution $\boldsymbol{x}^* = (\boldsymbol{x}^{1*} \;\ldots\; \boldsymbol{x}^{U*})$ shift (by only permuting some of its u indices) to another fractional solution with the same

cost. A very large number of cuts may be needed to cut all symmetric fractional solutions. There are cases where robust cuts defined over asymmetric aggregated original variables can be effective. However, there are also cases where it is not possible or effective to cut over those aggregate variables.

- There are cases where robust cuts over the original variables (or their asymmetric aggregations) are effective and lead to small gaps. However, non-robust cuts over the generated λ variables can be even more effective and, if the changes in the pricing can be somehow handled, may lead to really smaller gaps. In fact, the most advanced state-of-the-art BCPAs are exactly those where the non-robust cuts and the pricing algorithm are jointly and symbiotically designed, in such a way that the pricing can handle a large number of very tailored non-robust cuts without becoming too inefficient.

4.4. Three Examples

We provide three examples of classic LCOPs where BPAs or BCPAs can work very well.

4.4.1. The Generalized Assignment Problem

Definition 4.3: Generalized Assignment Problem (GAP). Instance: J jobs and K machines; integer capacities W_k, $k \in [K]$; assignment costs c_j^k and integer loads w_j^k, $j \in [J]$, $k \in [K]$. Solutions: assignments of jobs to machines such that the total load in each machine does not exceed its capacity. Goal: minimize total assignment cost.

The natural formulation for GAP uses binary variables x_j^k to indicate that job

j is assigned to machine k:

$$
\begin{aligned}
z_{\text{IP}} = \quad \min \quad & \sum_{k\in[K]}\sum_{j\in[J]} c_j^k x_j^k && && (4.16a)\\
\text{s.t.} \quad & \sum_{k\in[K]} x_j^k = 1 && j\in[J] && (4.16b)\\
& \sum_{j\in[J]} w_j^k x_j^k \le W_k && k\in[K] && (4.16c)\\
& 0 \le x_j^k \le 1 && j\in[J], k\in[K] && (4.16d)\\
& \boldsymbol{x} \in \mathbb{Z}^{JK}. && && (4.16e)
\end{aligned}
$$

A stronger formulation may be produced by applying the Dantzig-Wolfe reformulation. By keeping (4.16b) in the Master, the remaining constraints decompose into K independent sets of constraints. For each $k \in [K]$, P^k is:

$$
\begin{aligned}
& \sum_{j\in[J]} w_j^k x_j^k \le W_k && (4.17a)\\
& \mathbf{0} \le \boldsymbol{x}^k \le \mathbf{1}. && (4.17b)
\end{aligned}
$$

Let q_j be the j-th component of a point $\boldsymbol{q} \in Int(P^k) = Q^k$. The linear relaxation of the reformulated IP (which is an SPP, see Note 3.9) yields the following Master LP:

$$
\begin{aligned}
z_{\text{M}} = \quad \min \quad & \sum_{k\in[K]}\sum_{\boldsymbol{q}\in Q^k} \left(\sum_{j\in[J]} c_j^k q_j\right) \lambda_{\boldsymbol{q}}^k && && (4.18a)\\
\text{s.t.} \quad & \sum_{k\in[K]}\sum_{\boldsymbol{q}\in Q^k} q_j \lambda_{\boldsymbol{q}}^k = 1 && j\in[J] && (4.18b)\\
& \sum_{\boldsymbol{q}\in Q^k} \lambda_{\boldsymbol{q}}^k = 1 && k\in[K] && (4.18c)\\
& \boldsymbol{\lambda} \ge \mathbf{0}. && && (4.18d)
\end{aligned}
$$

The convexity constraints (4.18c) can be relaxed to $\le$ because $\mathbf{0}$ belongs to all sets $Int(P^k)$, $k \in [K]$. Let $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_J)$ be the vector with the dual variables associated to (4.18b) and $\boldsymbol{\nu} = (\nu_1, \ldots, \nu_K)$ be the vector with the dual variables associated to (4.18c). The pricing is performed by solving K subproblems, where

subproblem k, $k \in [K]$, is defined as:

$$\overline{c}^{k*} = \min \sum_{j \in [J]} (c_j^k - \pi_j^*) x_j^k - \nu_k^* \tag{4.19a}$$

$$\text{s.t.} \quad \boldsymbol{x}^k \in Int(P^k). \tag{4.19b}$$

The pricing subproblems are instances of the classic binary knapsack problem, which is (weakly) $\mathcal{NP}$-hard but very well solved in practice (Note 4.6). That fact makes the DW reformulation much interesting to GAP because it provides a practical way of implicitly obtaining all the facets of the knapsack polyhedra corresponding to $Conv(Int(P^k))$.

For example, consider the following GAP instance:

		Cost (c_j^k)				Load (w_j^k)				W_k
Jobs		**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	
Machines	**1**	8	3	2	9	2	3	3	1	5
	2	1	7	5	2	5	1	1	3	8

Formulation (4.16) becomes:

$$\begin{aligned}
z_{\text{IP}} = \min \quad & 8x_1^1 + 3x_2^1 + 2x_3^1 + 9x_4^1 + x_1^2 + 7x_2^2 + 5x_3^2 + 2x_4^2 & \\
\text{s.t.} \quad & x_1^1 + x_1^2 = 1 \\
& x_2^1 + x_2^2 = 1 \\
& x_3^1 + x_3^2 = 1 \\
& x_4^1 + x_4^2 = 1 \\
& 2x_1^1 + 3x_2^1 + 3x_3^1 + x_4^1 \leq 5 \\
& 5x_1^2 + x_2^2 + x_3^2 + 3x_4^2 \leq 8 \\
& \boldsymbol{x} \in \mathbb{B}^8.
\end{aligned}$$

An optimal integer solution has $x_1^1 = x_2^1 = x_3^2 = x_4^2 = 1$ and the other variables at 0, with $z_{\text{IP}} = 18$. However, its linear relaxation yields $z_{\text{LP}} = 9.69$. Such a big gap on a tiny instance indicates the standard BBA may need to explore many nodes for solving some larger GAP instances.

We now show how this instance can be solved by a robust BCPA. Figure 4.7 shows the resolution process in detail. $S_{1.1.1}$ corresponds to the first CGA iteration, where the RMLP is initialized with four artificial variables having the value 99. The

RMLP dual solution and the two subproblems are shown, both of them generate columns with negative reduced cost. The CGA converges at iteration $S_{1.1.6}$, so MLP $S_{1.1}$ yields the valid lower bound of 15 (a very significant gap reduction with respect to the linear relaxation of the original IP). Still at the root node, the robust cutting plane $x_1^2 + 2x_2^1 + 2x_3^1 + x_4^2 \leq 3$ is separated (this facet-defining cut does not seem to be part of a known family of GAP cuts, it was found computationally only for the sake of illustrating the robust BCPA). The subsequent CGA (note that now $\boldsymbol{\pi}$ has dimension 1×5) converges already in the first iteration $S_{1.2.1}$, so the new MLP $S_{1.2}$ yields the valid lower bound of 16.4. No other violated cut is found, so node S_1 ends with $z = 16.4$. The optimal RMLP at the end of that root node is:

$$
\begin{array}{lll}
z_{\text{RM}} = \min & & 10\lambda_1 + 13\lambda_2 + 11\lambda_3 + 7\lambda_4 + 11\lambda_5 + 3\lambda_6 + 3\lambda_7 + 14\lambda_8 + 2\lambda_9 \\
\text{s.t.} & & \lambda_1 + \lambda_2 + \lambda_5 + \lambda_6 = 1 \\
 & & \lambda_2 + \lambda_5 + \lambda_7 + \lambda_8 = 1 \\
 & & \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_8 + \lambda_9 = 1 \\
 & & \lambda_3 + \lambda_4 + \lambda_6 + \lambda_8 = 1 \\
\text{Cut} & \triangleright & 2\lambda_1 \quad \lambda_2 + 2\lambda_3 + \lambda_4 + 2\lambda_5 + 2\lambda_6 + 2\lambda_7 + \lambda_8 + 2\lambda_9 \leq 3 \\
 & & \lambda_1 + \lambda_3 + \lambda_5 + \lambda_7 + \lambda_9 \leq 1 \\
 & & \lambda_2 + \lambda_4 + \lambda_6 + \lambda_8 \leq 1 \\
 & & \boldsymbol{\lambda} \geq 0.
\end{array}
$$

Now, a branching over variable x_3^1 is performed. In Node S_2, corresponding to branching $x_3^1 \leq 0$, the CGA (note that now $\boldsymbol{\pi}$ has dimension 1×6) converges in two iterations to an integer solution with value 18. So, that node is pruned by integrality. Assuming that some variables with larger positive reduced costs (λ_2 and λ_8) were *cleaned*, i.e., removed from the RMLP, the optimal RMLP at the end of S_2 is:

$$
\begin{array}{lll}
z_{\text{RM}} = \min & & 10\lambda_1 + 11\lambda_3 + 7\lambda_4 + 11\lambda_5 + 3\lambda_6 + 3\lambda_7 + 2\lambda_9 + 6\lambda_{10} \\
\text{s.t.} & & \lambda_1 + \lambda_5 + \lambda_6 + \lambda_{10} = 1 \\
 & & \lambda_5 + \lambda_7 = 1 \\
 & & \lambda_1 + \lambda_3 + \lambda_4 + \lambda_9 + \lambda_{10} = 1 \\
 & & \lambda_3 + \lambda_4 + \lambda_6 = 1 \\
\text{Cut} & \triangleright & 2\lambda_1 + 2\lambda_3 + \lambda_4 + 2\lambda_5 + 2\lambda_6 + 2\lambda_7 + 2\lambda_9 + \lambda_{10} \leq 3 \\
x_3^1 \leq 0 & \triangleright & \lambda_1 + \lambda_3 + \lambda_9 \leq 0 \\
 & & \lambda_1 + \lambda_3 + \lambda_5 + \lambda_7 + \lambda_9 \leq 1 \\
 & & \lambda_4 + \lambda_6 + \lambda_{10} \leq 1 \\
 & & \boldsymbol{\lambda} \geq 0.
\end{array}
$$

In Node S_3, corresponding to branching $x_3^1 \geq 1$, the first iteration of the CGA

starts by solving the following RMLP:

$$
\begin{array}{rllcl}
z_{\mathrm{RM}} = & \min & 10\lambda_1 + 13\lambda_2 + 11\lambda_3 + 7\lambda_4 + 11\lambda_5 + 3\lambda_6 + 3\lambda_7 + 14\lambda_8 + 2\lambda_9 & & \\
 & \text{s.t.} & \lambda_1 + \lambda_2 + \lambda_5 + \lambda_6 & = & 1 \\
 & & \lambda_2 + \lambda_5 + \lambda_7 + \lambda_8 & = & 1 \\
 & & \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_8 + \lambda_9 & = & 1 \\
 & & \lambda_3 + \lambda_4 + \lambda_6 + \lambda_8 & = & 1 \\
\text{Cut} & \triangleright & 2\lambda_1 + \lambda_2 + 2\lambda_3 + \lambda_4 + 2\lambda_5 + 2\lambda_6 + 2\lambda_7 + \lambda_8 + 2\lambda_9 & \leq & 3 \\
x_3^1 \geq 1 & \triangleright & \lambda_1 + \lambda_3 + \lambda_9 & \geq & 1 \\
 & & \lambda_1 + \lambda_3 + \lambda_5 + \lambda_7 + \lambda_9 & \leq & 1 \\
 & & \lambda_2 + \lambda_4 + \lambda_6 + \lambda_8 & \leq & 1 \\
 & & \boldsymbol{\lambda} & \geq & 0,
\end{array}
$$

obtaining $z_{\mathrm{RM}} = 24$. The first subproblem obtains $\bar{c}^{1*} = 0$ and the second $\bar{c}^{2*} = -5$. Using Theorem 2.8, we obtain a lower bound of 19 for the z value of S_3. Therefore, the CGA can be stopped and the node is pruned by bound.

We will now show how the GAP could be solved by a possible *non-robust* BCPA, using only rank 1 Chvátal-Gomory Cuts (CGCs). We will apply the procedure to Constraints (4.18b), relaxed to $\leq$. Let $\boldsymbol{\rho} \in \mathbb{R}_+^{1\times J}$ be a vector of non-negative multipliers. The resulting CGC is:

$$\sum_{k\in[K]} \sum_{\boldsymbol{q}\in Q^k} \left\lfloor \sum_{j\in[J]} \rho_j q_j \right\rfloor \lambda_{\boldsymbol{q}}^k \leq \left\lfloor \sum_{j\in[J]} \rho_j \right\rfloor . \tag{4.21}$$

The cut is non-robust because the floor function in its definition is non-linear. This means that the pricing subproblems will change. Assume that at a given moment during the BCPA there are L such rank 1 cuts, CGC l, $l \in [L]$, having multipliers $\boldsymbol{\rho}^l$ and dual variable σ_l. The pricing subproblem k, $k \in [K]$, becomes:

$$
\begin{aligned}
\bar{c}^{k*} = \min \quad & \sum_{j\in[J]} (c_j^k - \pi_j^*) x_j^k - \sum_{l\in[L]} \sigma_l^* w_l \quad - \nu_k^* && (4.22a) \\
\text{s.t.} \quad & \boldsymbol{x}^k \in Int(P^k) && (4.22b) \\
& w_l \geq \sum_{j\in[J]} \rho_j^l x_j^k - 1 + \epsilon \qquad l \in [L] && (4.22c) \\
& w \in \mathbb{Z}_+^L, && (4.22d)
\end{aligned}
$$

where ϵ is a small constant and w_l, $l \in [L]$, is an auxiliary integer variable devised to

Figure 4.7: Robust BCPA tree for the GAP instance

assume the value $\lfloor \sum_{j \in [J]} \rho_j^l x_j^k \rfloor$. The linearization trick works because it is known that $\boldsymbol{\sigma} \leq 0$ (the dual variable of a $\leq$ constraint in a minimization problem is always non-positive). Anyway, those pricing subproblems can not be solved by highly efficient black-box knapsack codes. While it may be possible to adapt one of those codes for handling the new pricing subproblems, a more practical approach would be to solve them using a much less efficient general MIP solver. Anyway, even with

the best possible adapted codes, each added CGC is likely to make those pricing subproblems harder. So, the algorithm designer should be very careful in separating such cuts, in order to avoid intractable pricing.

In our example, the non-robust BCPA would work as depicted in Figure 4.8. The first CGA convergence is identical to the robust BCPA, so $S_{1.1}$ yields the valid lower bound of 15. The multipliers of a CGC for cutting the fractional solution of $S_{1.1}$ are $\boldsymbol{\rho}^1 = (\,2/3 \;\; 1/3 \;\; 1/3 \;\; 1/3\,)$. It can be seen that variables $\lambda^1_{(1010)}$, $\lambda^2_{(1001)}$, and $\lambda^2_{(0111)}$ will have coefficient 1 in (4.21), while the RHS is also 1. So, the cut is violated by 0.5 units. $S_{1.2}$ gets another fractional solution with value 15. Then a second CGC with $\boldsymbol{\rho}^2 = (\,1/3 \;\; 1/3 \;\; 1/3 \;\; 2/3\,)$ can be separated. Now, $S_{1.3}$ finds an integer solution with value 18 and the BCPA finishes at the root node. The last RMLP is:

$$
\begin{array}{lllllllllll}
z_{\mathrm{RM}} = \min & 10\lambda_1 \;+ & 13\lambda_2 \;+ & 11\lambda_3 \;+ & 7\lambda_4 \;+ & 11\lambda_5 \;+ & 3\lambda_6 \;+ & 3\lambda_7 \;+ & 14\lambda_8 \;+ & 2\lambda_9 & \\
\quad\text{s.t.} & \lambda_1 \;+ & \lambda_2 & & \;+ & \lambda_5 \;+ & \lambda_6 & & & & = 1\\
 & & \lambda_2 & & \;+ & \lambda_5 & & \lambda_7 \;+ & \lambda_8 & & = 1\\
 & \lambda_1 \;+ & \lambda_2 \;+ & \lambda_3 \;+ & \lambda_4 & & & \;+ & \lambda_8 \;+ & \lambda_9 & = 1\\
 & & & \lambda_3 \;+ & \lambda_4 & \;+ & \lambda_6 & \;+ & \lambda_8 & & = 1\\
\text{CG cut 1} \;\triangleright & \lambda_1 \;+ & \lambda_2 & & \;+ & \lambda_5 \;+ & \lambda_6 & \;+ & \lambda_8 & & \leq 1\\
\text{CG cut 2} \;\triangleright & & \lambda_2 \;+ & \lambda_3 \;+ & \lambda_4 & \;+ & \lambda_6 & \;+ & \lambda_8 & & \leq 1\\
 & \lambda_1 & \;+ & \lambda_3 \;+ & \;+ & \lambda_5 & \;+ & \lambda_7 & \;+ & \lambda_9 & \leq 1\\
 & & \lambda_2 & \;+ & \lambda_4 & \;+ & \lambda_6 & \;+ & \lambda_8 \;+ & & \leq 1\\
 & & & & & & & & & \boldsymbol{\lambda} & \geq 0.
\end{array}
$$

4.4.2. The Cutting Stock Problem

Definition 4.4: Cutting Stock Problem (CSP). Instance: J items, integer lengths w_j and demands (required number of copies) d_j, $j \in [J]$; integer stock length W. Solutions: a way of cutting the stocks that produces the demand for each item. Goal: minimize the number of used stocks.

The CSP, including the following very important particular case, is one of the most widely studied COPs.

Definition 4.5: Bin Packing Problem (BPP). The BPP is the particular case of the CSP where all demands are unitary.

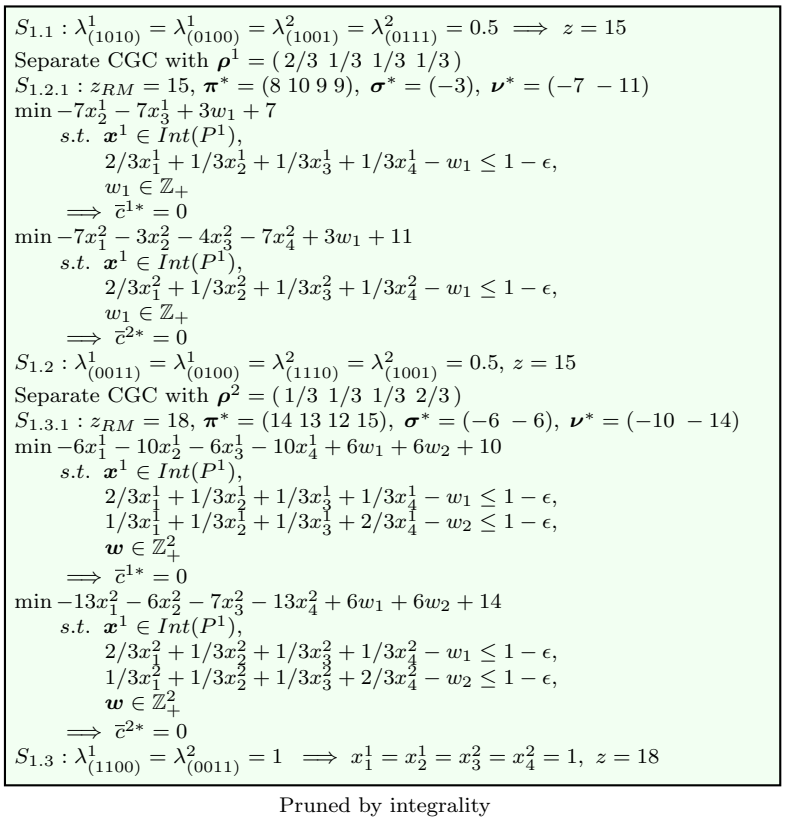


Figure 4.8: Non Robust BCPA tree for the GAP instance

The CSP can be formulated as follows. Let U be an upper bound on the number of stocks that need to be cut. This bound can be produced by any heuristic method for the CSP. Then, let binary variable y^u, $u \in [U]$, indicate whether stock u is indeed used. Integer variables x^u_j, $j \in [J]$, $u \in [U]$, represent the number of copies of item

j that will be cut from stock u. The formulation is:

$$
\begin{aligned}
z_{\text{IP}} = \quad \min \quad & \sum_{u\in[U]} y^u && && (4.23a)\\
\text{s.t.} \quad & \sum_{u\in[U]} x^u_j = d_j && j \in [J] && (4.23b)\\
& \sum_{j\in[J]} w_j x^u_j \le W y^u && u \in [U] && (4.23c)\\
& x^u_j \ge 0 && j \in [J], u \in [U] && (4.23d)\\
& 0 \le y^u \le 1 && u \in [U] && (4.23e)\\
& (\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{Z}^{J(U+1)}. && && (4.23f)
\end{aligned}
$$

This formulation (wrongly attributed to Kantorovich [1939] in the last decades, see Note 4.9) is really bad. First, its linear relaxation always yields $z_{\text{LP}} = \sum_{j\in[J]} w_j d_j / W$, which is an obvious lower bound for the CSP. Second, its extreme symmetry makes branching and cutting over it extremely ineffective. Third, it is not even polynomially-sized, since there are classes of instances where U grows exponentially with the instance size. For example, it suffices to increase the item demands while keeping the remaining data fixed in order to obtain such a class. However, a much better formulation may be obtained by applying the Dantzig-Wolfe reformulation to it. By keeping (4.23b) in the Master, the remaining constraints decompose into U independent sets. As those sets define identical subproblems, all P^u, $u \in [U]$, are equivalent to the following polyhedron P:

$$
\begin{aligned}
& \sum_{j\in[J]} w_j x_j \le W y && (4.24a)\\
& \boldsymbol{x} \ge \mathbf{0} && (4.24b)\\
& 0 \le y \le 1 && (4.24c)
\end{aligned}
$$

The point $\mathbf{0}$ belongs to $Int(P)$ and corresponds to $(\boldsymbol{x} = \mathbf{0}, y = 0)$. Any other point $\boldsymbol{q} \in Int(P)$ has component $q_{J+1} = 1$, corresponding to solutions with $y = 1$. Let $Q = Int(P) \setminus \mathbf{0}$. As we are in the case that all subproblems are identical ($K = 1$, see Section 2.3.2), the linear relaxation of the reformulated IP yields the following

Master LP:

$$
\begin{aligned}
z_{\mathrm{M}} = \quad & \min \quad && \sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} && && (4.25a)\\
& \text{s.t.} \quad && \sum_{\boldsymbol{q} \in Q} q_j \lambda_{\boldsymbol{q}} = d_j && j \in [J] && (4.25b)\\
& && \sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} + \lambda_{\boldsymbol{0}} = U && && (4.25c)\\
& && \boldsymbol{\lambda} \geq \boldsymbol{0}. && && (4.25d)
\end{aligned}
$$

Note that variable $\lambda_{\boldsymbol{0}}$, which only appears in (4.25c), can be viewed as the slack of an equivalent constraint $\sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} \leq U$. As U is an upper bound on the quantity $\sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}}$ that is being minimized, that convexity constraint is redundant and *can be removed*. So, let $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_J)$ be the vector with the dual variables associated to (4.25b). By noticing that all the generated variables correspond to points in $Int(P)$ such that $y = 1$, the pricing step in the CGA is performed by solving the following subproblem:

$$
\begin{aligned}
\bar{c}^* \; = \; & \min \quad && 1 - \sum_{j \in [J]} \pi_j^* x_j && && (4.26a)\\
& \text{s.t.} \quad && \sum_{j \in [J]} w_j x_j \leq W && && (4.26b)\\
& && \boldsymbol{x} \in \mathbb{Z}_+^J. && && (4.26c)
\end{aligned}
$$

The pricing subproblems are instances of the classic integer knapsack problem, which is also (weakly) $\mathcal{NP}$-hard and also very well solved in practice (Note 4.6). The reformulated IP for CSP corresponds to its classic formulation by Kantorovich [1939] (see Note 4.9) and by Gilmore and Gomory [1961]. The solutions of (4.26) correspond to the *cutting patterns*, i.e., the possible ways of cutting a stock. As the same cutting pattern can be used more than once, λ variables can assume values greater than 1.

For example, consider the following CSP instance: $J = 4$, $w = (\,40\ 35\ 31\ 13\,)$, $d = (\,4\ 5\ 5\ 8\,)$, and $W = 100$. Formulation (4.23), no matter which heuristic upper bound U is used, would get $z_{\mathrm{LP}} = 594/100$. As z_{IP} is known to be integer, rounding z_{LP} up obtains the valid lower bound of 6. In order to solve (4.25), we initialize the RMLP with the 4 cutting patterns produced by cutting the maximum number of

identical items from a stock. The CGA sequence is depicted in Figure 4.9. The final optimal RMLP is:

$$
\begin{array}{lrrrrrrrrl}
z_{\mathrm{RM}} = \min & \lambda_1 & +\lambda_2 & +\lambda_3 & +\lambda_4 & +\lambda_5 & +\lambda_6 & +\lambda_7 & +\lambda_8 & \\
\text{s.t.} & 2\lambda_1 & & & & & +2\lambda_6 & & & = 4 \\
& & 2\lambda_2 & & & +2\lambda_5 & & +\lambda_7 & +\lambda_8 & = 5 \\
& & & 3\lambda_3 & & & & & +2\lambda_8 & = 5 \\
& & & & 7\lambda_4 & +2\lambda_5 & +\lambda_6 & +5\lambda_7 & & = 8 \\
& & & & & & & & & \boldsymbol{\lambda} \geq \mathbf{0},
\end{array}
$$

with $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 0$, $\lambda_5 = 0.81$, $\lambda_6 = 2$, $\lambda_7 = 0.88$, $\lambda_8 = 2.5$, and $z_{\mathrm{RM}} = z_{\mathrm{M}} = 6.19$. By rounding up the MLP value, an improved lower bound of 7 is got. As proposed in Gilmore and Gomory [1961], a good heuristic solution for the CSP can be found by simply rounding up the fractional solution itself. In our example, we would make $\lambda_5 = 1$, $\lambda_6 = 2$, $\lambda_7 = 1$, $\lambda_8 = 3$, and then trim the copies of some items that exceed their demands. The resulting proven optimal integer solution with value 7 is depicted in Figure 4.10. By incorporating that primal heuristic procedure, the BPA finishes at the root node in that example. Two potentially better ways of "rounding" the final RMLP are: (1) relaxing its demand constraints to $\geq$ (allowing overproducing some items) and solving it as an IP; (2) iteratively fixing a single fractional variable $\lambda_{\boldsymbol{q}'}$ to $\lceil \lambda_{\boldsymbol{q}'} \rceil$ and resolving the residual problem (as in (4.10)) by the CGA (the diving heuristic fully described in Chapter 6). It is also convenient to relax the demand constraints to $\geq$ in order to avoid the risk of infeasibility.

The Kantorovich-Gilmore-Gomory formulation for the CSP is remarkably strong. The existence of Non Integer Round Up (NIRUP) instances, those where its linear relaxation rounded up does not match the optimal integer solution, was only proved in Marcotte [1985]. Even today, the following conjecture is still open:

Conjecture 4.1: The lower bound from the linear relaxation of the Kantorovich-Gilmore-Gomory formulation rounded up is at most one unit away from the CSP optimal integer solution.

NIRUP instances are rare [Kartak et al., 2015]. So, a BPA for the CSP is often reduced to a diving heuristic (Chapter 6) for finding one of the (usually many) integer solutions that match the optimal lower bound already found in the root

$S_{1.1.1} : z_{RM} = 7.3095$, $\boldsymbol{\pi}^* = (0.5\ 0.5\ 0.3333\ 0.1429)$
$\min 1 - 0.5x_1 - 0.5x_2 - 0.3333x_3 - 0.1429x_4\ s.t.\ \boldsymbol{x} \in X \implies \boldsymbol{x}^* = (0\ 2\ 0\ 2)$, $\overline{c}^* = -0.2857$
$S_{1.1.2} : z_{RM} = 6.5952$, $\boldsymbol{\pi}^* = (0.5\ 0.3571\ 0.3333\ 0.1429)$
$\min 1 - 0.5x_1 - 0.3571x_2 - 0.3333x_3 - 0.1429x_4\ s.t.\ \boldsymbol{x} \in X \implies \boldsymbol{x}^* = (2\ 0\ 0\ 1)$, $\overline{c}^* = -0.1429$
$S_{1.1.3} : z_{RM} = 6.3095$, $\boldsymbol{\pi}^* = (0.4286\ 0.3571\ 0.3333\ 0.1429)$
$\min 1 - 0.4286x_1 - 0.3571x_2 - 0.3333x_3 - 0.1429x_4\ s.t.\ \boldsymbol{x} \in X \implies \boldsymbol{x}^* = (0\ 1\ 0\ 5)$, $\overline{c}^* = -0.0714$
$S_{1.1.4} : z_{RM} = 6.2917$, $\boldsymbol{\pi}^* = (0.4375\ 0.375\ 0.3333\ 0.125)$
$\min 1 - 0.4375x_1 - 0.375x_2 - 0.3333x_3 - 0.125x_4\ s.t.\ \boldsymbol{x} \in X \implies \boldsymbol{x}^* = (0\ 1\ 2\ 0)$, $\overline{c}^* = -0.0417$
$S_{1.1.5} : z_{RM} = 6.1875$, $\boldsymbol{\pi}^* = (0.4375\ 0.375\ 0.3125\ 0.125)$
$\min 1 - 0.4375x_1 - 0.375x_2 - 0.3125x_3 - 0.125x_4\ s.t.\ \boldsymbol{x} \in X \implies \overline{c}^* = 0$
$S_{1.1} : \lambda_{(0202)} = 0.8125$, $\lambda_{(2001)} = 2$, $\lambda_{(0105)} = 0.875$, $\lambda_{(0120)} = 2.5$, $z = 6.1875$
Round up fractional solution, obtaining $UB = 7$

Pruned by bound

Figure 4.9: BPA tree for the CSP example, $X = \{\boldsymbol{x} \in \mathbb{Z}^4_+ : 40x1 + 35x_2 + 31x_3 + 13x_4 \leq 100\}$

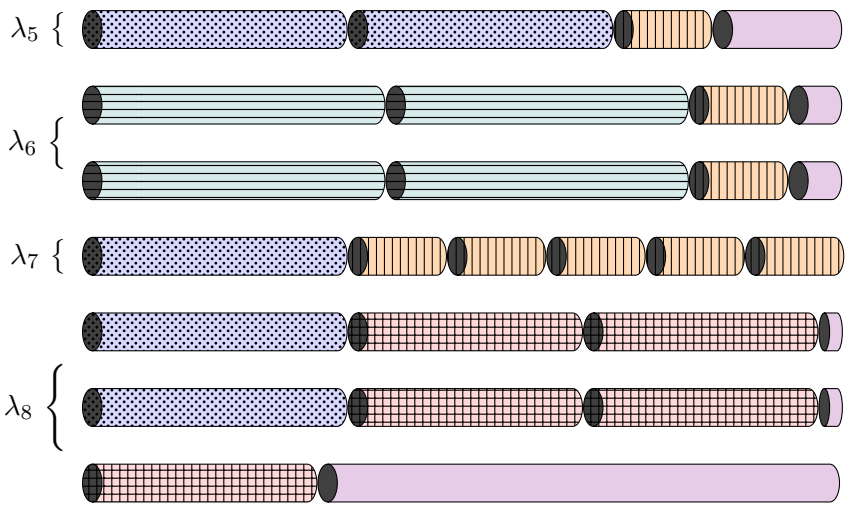


Figure 4.10: Optimal CSP solution

node. However, there are instances where this may not work, either because they are NIRUP or because finding an optimal integer solution is difficult. So, a BPA algorithm should include a complete branching scheme. However, branching in a BPA or BCPA for CSP (and for several other problems with similar characteristics) is a quite complex matter, that will be presented in depth in Chapter 8. We will mention now the following:

- Robust branching over the original variables $(\boldsymbol{x}, \boldsymbol{y})$ is useless because of their

extreme symmetry. Moreover, it is also not possible to branch over the asymmetric aggregated original variables $\sum_{k\in[K]} x_j^k$, $j \in [J]$, because they are always equal to the integer values d_j. Branching over the asymmetric aggregated original variable $\sum_{k\in[K]} y^k$, if it has a fractional value z^*, is possible but not recommended. The node S_2 corresponding to $\sum_{k\in[K]} y^k \leq \lfloor z^* \rfloor$ is certainly infeasible and can be already pruned. The node S_3 corresponding to $\sum_{k\in[K]} y^k \geq \lceil z^* \rceil = \sum_{\boldsymbol{q}\in Q} \lambda_{\boldsymbol{q}} \geq \lceil z^* \rceil$ (an objective value cut) would have $z = \lceil z^* \rceil$, a bound that was already known. The problem is that no robust branching is now available for S_3. Anyway, the objective value cut can have a negative blinding effect on a BCPA (see Note 3.8).

4.4.3. The Capacitated Vehicle Routing Problem

The CVRP natural Formulation (3.6) is not well suited for DW reformulation. This happens because no matter which sets of constraints are chosen to be kept in the master, there will be no decomposition of the subproblem into multiple independent subproblems.

The following extended formulation is more suited for DW reformulation. It is actually a formulation for the more general Asymmetric CVRP (ACVRP), defined over a graph $G_D = (V, A)$ where A contains a pair of opposite arcs (i, j) and (j, i) for each edge $e = \{i, j\} \in E$. However, it can also be used for the CVRP by only defining symmetric costs. The value $U = |V_+|$ is an upper bound on the number of routes in any solution. For each $a = (i, j) \in A$ and $u \in [U]$, let y_a^u be a binary variable indicating that route u includes the travel from i to j and let f_a^u be a continuous variable indicating the corresponding load of the vehicle (the sum of the demands already collected in the route) during that travel, if $y_a^u = 1$. If $y_a^u = 0$,

then $f_a^u = 0$.

$$
\begin{aligned}
z_{\text{IP}} \;=\; \min \quad & \sum_{u\in[U]}\sum_{a\in a} c_a y_a^u && && (4.27a)\\
\text{s.t.} \quad & \sum_{u\in[U]}\sum_{a\in\delta^-(i)} y_a^u = 1 && i\in V_+ && (4.27b)\\
& \sum_{a\in\delta^-(i)} y_a^u - \sum_{a\in\delta^+(i)} y_a^u = 0 && i\in V,\, u\in[U] && (4.27c)\\
& \sum_{a\in\delta^-(i)} y_a^u \le 1 && i\in V,\, u\in[U] && (4.27d)\\
\sum_{a\in\delta^-(i)} f_a^u - \sum_{a\in\delta^+(i)} f_a^u = -\,d_i & \sum_{a\in\delta^-(i)} y_a^u && i\in V_+,\, u\in[U] && (4.27e)\\
& f_a^u = 0 && a\in\delta^+(0),\, u\in[U] && (4.27f)\\
& f_a^u \le W y_a^u && a\in A,\, u\in[U] && (4.27g)\\
& (\boldsymbol{y},\boldsymbol{f}) \in \mathbb{Z}_+^{|A|U}\times\mathbb{R}_+^{|A|U}. && && (4.27h)
\end{aligned}
$$

By keeping (4.27b) in the master, the remaining constraints decompose into U independent sets of constraints. As those sets define identical subproblems, all P^u, $u\in[U]$, are equivalent to the following polyhedron P:

$$
\begin{aligned}
& \sum_{a\in\delta^-(i)} y_a - \sum_{a\in\delta^+(i)} y_a = 0 && i\in V && (4.28a)\\
& \sum_{a\in\delta^-(i)} y_a \le 1 && i\in V && (4.28b)\\
\sum_{a\in\delta^-(i)} f_a - \sum_{a\in\delta^+(i)} f_a = -\,d_i & \sum_{a\in\delta^-(i)} y_a && i\in V_+ && (4.28c)\\
& f_a = 0 && a\in\delta^+(0) && (4.28d)\\
& f_a \le W y_a && a\in A && (4.28e)\\
& (\boldsymbol{y},\boldsymbol{f}) \in \mathbb{R}_+^{|A|}\times\mathbb{R}_+^{|A|}. && && (4.28f)
\end{aligned}
$$

Extended formulation (4.27) is in format (4.8), but with $\boldsymbol{h}=\boldsymbol{0}$ and $\boldsymbol{D}=\boldsymbol{0}$, since the continuous variables f do not appear neither in (4.27a) nor in (4.28a). Therefore, the columns in the resulting Master LP will only depend on the y part of the points $(\boldsymbol{y},\boldsymbol{f})\in P$. In fact, the relevant set of points is $Q = Proj_{\boldsymbol{y}}(\{(\boldsymbol{y},\boldsymbol{f})\in P,\, \boldsymbol{y}\in\mathbb{Z}^{|A|}\})$, i.e., the set of all possible individual CVRP routes, each route being represented

as the incidence vector of its set of arcs. Note that constraints (4.28a) and (4.28b) alone force any binary $\boldsymbol{y}$ to define a set of vertex-disjoint cycles. The remaining constraints in (4.28) eliminate all cycles that do not contain the depot and also enforce the vehicle capacity on the single cycle that passes by the depot. Actually, the point $\mathbf{0}$, representing the empty route, is also a valid solution. As we are in the case that all subproblems are identical ($K = 1$, see Section 2.3.2), the linear relaxation of the reformulated IP yields the following Master LP:

$$z_{\mathrm{M}} = \min \quad \sum_{\boldsymbol{q} \in Q} \Big(\sum_{a \in a} c_a q_a \Big) \lambda_{\boldsymbol{q}} \tag{4.29a}$$

$$\text{s.t.} \quad \sum_{\boldsymbol{q} \in Q} \Big(\sum_{a \in \delta^-(i)} q_a \Big) \lambda_{\boldsymbol{q}} = 1 \qquad i \in V_+ \tag{4.29b}$$

$$\sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} \leq U \tag{4.29c}$$

$$\boldsymbol{\lambda} \geq \mathbf{0}. \tag{4.29d}$$

As $U = |V_+|$, the convexity constraint (4.29c) is redundant and *can be removed*, so, the reformulated IP is an SPP. Let $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_{|V_+|})$ be the vector with the dual variables associated to (4.29b). The pricing step in the CGA is performed by solving the following subproblem:

$$\min \ \bar{c} \ = \sum_{a=(i,j) \in A} (c_a - \pi_j^*) y_a \tag{4.30a}$$

$$\text{s.t.} \qquad \boldsymbol{y} \in Q. \tag{4.30b}$$

The pricing subproblem corresponds to the following LCOP:

Definition 4.6: Capacitated Minimum Cost Elementary Cycle. Instance: Directed graph $G_D = (V, A)$, where $V = \{0\} \cup V_+$, vertex 0 represents a depot and V_+ the set of customers; arc costs $\bar{c}_a$, $a \in A$ (unrestricted in sign); integer positive demands d_i, $i \in V_+$; and integer vehicle capacity W. Solutions: routes in G starting and ending at the depot and such that the sum of the demands of the customers in it does not exceed W. Goal: minimize the sum of the cost of the arcs in the route.

Here there is a problem. The above LCOP is strongly $\mathcal{NP}$-hard and may also be quite hard in practice. The solution almost always adopted in practice is to re-

lax the vertex elementarity in the definition of the subproblem. This means that instead of pricing actual routes, one can price q-routes: walks starting and ending at the depot respecting the capacity constraint [Christofides et al., 1981]. A customer vertex i may be visited more than once, but each visit consumes d_i units of capacity. Therefore, the number of q-routes is finite. Pricing q-routes can be done in pseudopolynomial $O(W|A|)$ time by DP. In fact, pricing q-routes without 2-cycles (subcycles of format $i-j-i$) can be done by only doubling the number of states in the DP, so the complexity remains $O(W|A|)$. Is that relaxation valid in a BPA for the CVRP? Yes, because the coefficient of a q-route variable in the constraint in (4.29b) corresponding to customer i is equal to the number of times that it visited i. If that coefficient is greater than 1, then the variable should have value zero in any integer solution (variables like that are called *non-proper*, see Note 4.4). Therefore, the formulation remains valid. However, the additional q-route variables may have a positive value in a fractional solution, weakening the formulation. Yet, the q-route formulation is still significantly stronger than the original Formulation (4.27). Stronger route relaxations include q-routes without s-cycles for $s \geq 3$ [Irnich and Villeneuve, 2006] and ng-routes [Baldacci et al., 2011].

Effective robust cutting and branching can be performed over the asymmetric aggregated original variables $y_a = \sum_{u \in [U]} y_a^u$, $a \in A$. Those variables are binary and have value 1 if the arc a is used in some route. However, if the CVRP instance is defined over an undirected graph $G = (V, E)$, it is recommended to cut and branch over the more aggregated variables $x_e = y_{ij} + y_{ji}$, $e = \{i, j\} \in E$, in order to avoid another symmetry (the same undirected route can be represented in two different ways over the directed variables). Those x variables are exactly those in Formulation (3.6). Rounded Capacity Cuts (3.6c) and any other cuts known for that formulation can be used in a robust BCPA. Constraints (4.29b) define a set-partitioning problem. Non-robust cuts valid for it can be used in a non-robust BCPA. Of course, one should be very careful not to make the pricing intractable (see Chapter 10).

4.5. Nuances of Robustness

Branchings and cuts defined over the original formulation variables are robust since the new dual variables from the introduced constraints only affect the objective function in the pricing subproblems, ensuring that their structure remains unaltered.

Yet, this is not the full history. There are more subtle cases that will be illustrated with three examples.

4.5.1. Graph Coloring Problem: robustness "by luck"

Consider the following classic $\mathcal{NP}$-hard problem.

Definition 4.7: Graph Coloring Problem (GCP). Instance: undirected graph $G = (V, E)$. Solutions: the vertex-colorings of G, i.e., mappings $f : V \mapsto \mathbb{Z}_{\geq 1}$ (colors are represented by positive integers) such that for every edge $\{i, j\} \in E$, $f(i) \neq f(j)$. Goal: Minimize the number of distinct colors used.

Let U be an upper bound on the minimum number of colors, obtained by any heuristic. Let y_u, $u \in [U]$, be a binary variable indicating whether color u is used and let x_i^u, $i \in V$, $u \in [U]$, be a binary variable indicating whether vertex i receives color u. A GCP formulation is:

$$
\begin{aligned}
z_{\text{IP}} = \quad & \min \quad \sum_{u \in [U]} y^u & & & (4.31a)\\
& \text{s.t.} \quad \sum_{u \in [U]} x_i^u = 1 & & i \in V & (4.31b)\\
& & x_i^u \leq y^u & i \in V, u \in [U] & (4.31c)\\
& & x_i^u + x_j^u \leq 1 & \{i, j\} \in E, u \in [U] & (4.31d)\\
& & (\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{B}^{|V|(U+1)}. & & (4.31e)
\end{aligned}
$$

The above formulation is big and suffers from symmetry, but its DW decomposition (keeping (4.31b) in the master and doing some simplifications) can yield the SPP formulation used in Mehrotra and Trick [1996], where the variables correspond to the independent sets of G. An independent set (a.k.a. stable set) is a vertex-set $V' \subseteq V$ such that no pair of vertices in it is connected by an edge. Vertices in an independent set can always receive the same color. Let $Q = \{\boldsymbol{\chi}(V') : V' \text{ is an independent set of G}\}$, so q_i indicates whether vertex i belongs to the in-

dependent set corresponding to $\boldsymbol{q} \in Q$. The formulation is:

$$\min \quad \sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} \tag{4.32a}$$

$$\text{s.t.} \quad \sum_{\boldsymbol{q} \in Q} q_i \lambda_{\boldsymbol{q}} = 1 \qquad i \in V \tag{4.32b}$$

$$\boldsymbol{\lambda} \in \mathbb{Z}_+^{|Q|}. \tag{4.32c}$$

Its linear relaxation can be solved by the CGA, the pricing subproblem being the following LCOP:

Definition 4.8: Independent Set Problem (ISP), a.k.a. Stable Set Problem. Instance: undirected graph $G = (V, E)$, weights w_i, $i \in V$. Solutions: sets $V' \subseteq V$ such that for every edge $e = \{u, v\} \in E$, $|V' \cap e| \leq 1$. Goal: Maximize $\sum_{i \in V'} w_i$.

The dual variables $\boldsymbol{\pi}$ associated to (4.32b) provide the weights. If a solution V' is such that $1 - \sum_{i \in V'} \pi_i^* < 0$, then a column with negative reduced cost is found. The ISP is closely related to the Vertex Cover Problem (Definition 3.5): a set V' is an independent set if and only if its complement $V \setminus V'$ is a vertex cover (for example, in Figure 3.3, $\{4, 6\}$ is an independent set and $\{1, 2, 3, 5\}$ is a vertex cover). Therefore, it is also $\mathcal{NP}$-hard. Yet, there are algorithms with a good practical performance for many instances with up to a few hundred vertices (Note 4.7).

In order to obtain integer solutions in their proposed BPA, Mehrotra and Trick used the branching scheme introduced in Ryan and Foster [1981] for the SPP (see Note 3.11). In this context, one should branch by choosing a pair of vertices $i, j \in V$ such that the linear expression $\sum_{\boldsymbol{q} \in Q \,:\, q_i + q_j = 2} \lambda_{\boldsymbol{q}}^*$ is fractional (as the RFB is complete, if no such pair exists then $\boldsymbol{\lambda}^*$ should be integer):

- In the left node i and j should be in the same independent set, i.e, have the same color. This is imposed by removing from the RMLP columns corresponding to vectors $\boldsymbol{q}$ such that $q_i + q_j = 1$ and forbidding the generation of new columns with that feature. By merging vertices i and j into a single vertex (if this operation produces parallel edges, a single copy of those edges is kept), a new graph G' is produced. The independent sets priced over G' will either contain the merged vertex (leading to a column containing both i and j) or not (getting a column that does not contain both i and j). Note that the weight

of the merged vertex in the resulting pricing subproblems is given by $\pi_i^* + \pi_j^*$. In fact, it is possible to have a small gain in performance, at the cost of more bookkeeping in the BPA code, by also merging the rows corresponding to i and j to a single row in the RMLP.

- In the right node i and j should be in different independent sets, i.e, have distinct colors. This is imposed by removing from the RMLP columns corresponding to vectors $\boldsymbol{q}$ such that $q_i + q_j = 2$ and forbidding the generation of new columns with that feature. This is done by *including the new edge* $\{i, j\}$ *into* E, producing a new graph G'. The independent sets priced over G' will not contain both i and j.

We have here an example of a robust BPA where the branching is performed over (a linear expression on) the generated λ variables. In fact, the pricing can still be performed by a black-box ISP code in both children nodes. "By luck", the modifications in the pricing induced by the branching can be done by only changing the input graph given to that ISP solver. There are no changes in the subproblem structure and pricing is not likely to become slower even after many branchings are performed.

4.5.2. BPP: robustness depends on how subproblems are defined

Given that many other problems where CG is applied also have an SPP structure, it would be good if RFB could always be applied in a robust way. The issue is not so simple. For example, consider the BPA for the Bin Packing Problem presented in Vance et al. [1994], which used RFB. In that context, it means selecting items i and j in $[J]$ and branching on the disjunction that either i and j are packed in the same bin or in distinct bins. Indeed, the first possibility is easily imposed by merging i and j to a super item with length $w_i + w_j$, so, the priced columns either pack i and j together or do not pack i or j. The resulting pricing subproblem is still a binary KP problem. However, in the other child node, it is necessary to change the pricing subproblem by including the constraint that it is not possible to include both i and j in a solution. Up to a point, as long as the pairs of items that can not be together are not overlapping, the modified pricing subproblem could still be efficiently solved

in pseudo-polynomial time, as a Multiple Choice KP (Note 4.12). However, if too many branchings are needed, overlaps start to appear, and the pricing has to be done by a much less efficient general MIP solver. The strategy proposed by Vance et al. [1994] was to explore the "fast-pricing nodes" in the BPA tree first, hoping to quickly find an optimal integer solution and never having to really solve the "slow-MIP-pricing nodes". In fact, on IRUP instances, when an optimal solution is found, the procedure in line 11 of Algorithm 1 immediately prunes all open nodes, and the BPA ends.

Nowadays, we would implement that BPA in a slightly different way. The so-called Knapsack Problem with Conflicts (KPC), which can be used for solving the pricing subproblem after an arbitrary number of Ryan-Foster branchings, started to be studied in Yamada et al. [2002]. In spite of being a strongly $\mathcal{NP}$-hard problem, now there are quite efficient combinatorial algorithms for it. In particular, the codes in Coniglio et al. [2021] are several orders of magnitude faster than general MIP solvers. This means that RFB for the BPP is robust if it is assumed that, in the worst case, the pricing will be solved as a KPC. More precisely, we would say that *the pricing is robust with respect to the KPC.*

A reader may now argue that the above definition of robustness has a flaw: almost everything would be robust with respect to a general MIP! For example, as shown in Section 4.3.1, there are tricks to model the modifications in the pricing subproblems induced by Chvátal-Gomory Cuts of arbitrary rank over the generated variables. Our answer is the following:

- Relatively few successful BPA/BCPAs perform pricing using MIP solvers. In most cases, success depends on the use of specialized codes that can not handle general non-robustness. But even when the pricing is solved by a MIP solver, success relies on the fact those MIPs are well-solved in practice. The MIP modifications required by non-robustness consistently make them harder, the more non-robust branchings/cuts the worse. For example, the tricks for handling general CGCs not only increase MIP size, they also make them much weaker. A few dozen such cuts may already lead to intractability. So, branchings/cuts that make pricing MIPs harder are still classified as non-robust.

The whole concept of robustness is intended to capture the following practical distinction. Robust branchings/cuts are robust in the sense that one may use them

without fearing that they will make the pricing intractable. On the other hand, non-robust branchings/cuts (which play a crucial role in the most advanced BCPAs) are non-robust in the sense that one should be very careful in using them in order to avoid intractability.

4.5.3. CSP: robustness depends on what one considers as the original formulation

Valério de Carvalho [1998] proposed a strong flow formulation for the CSP having a pseudo-polynomially large number of variables. Let $G = (V, A)$ be a directed graph having vertex-set $V = \{0, \ldots, W\}$ and arc-set $A = \cup_{j\in[J]} A_j \cup A_0 \cup (W, 0)$, where $A_j = \{(v - w_j, v) \,|\, w_j \leq v \leq W\}$ and $A_0 = \{(v - 1, v) \,|\, 1 \leq u \leq W\}$. Each path from vertex 0 to vertex W corresponds to a cutting pattern, arcs in A_j are uses of item j while arcs in A_0 may correspond to waste. Figure 4.11 shows the graph G for an instance with $J = 2$, $w_1 = 2$, $w_2 = 3$, and $W = 5$. Define a variable x_a for each $a \in A$. The formulation is:

$$\min \quad z = x_{(W,0)} \tag{4.33a}$$

$$\text{s.t.} \quad \sum_{a\in\delta^-(i)} x_a - \sum_{a\in\delta^+(i)} x_a = 0 \qquad i \in V \tag{4.33b}$$

$$\sum_{a\in A_j} x_a = d_j \qquad j \in [J] \tag{4.33c}$$

$$\boldsymbol{x} \in \mathbb{Z}_+^{|A|}. \tag{4.33d}$$

The "return flow" variable $x_{(W,0)}$ counts many cutting patterns are used to satisfy the demands. If there is an item j' such that $d_{j'} = 1$ (so $A_0 = A_{j'}$), the constraint in (4.33c) corresponding to j' should be relaxed to $\geq$. Formulation (4.33) is potentially large, having $O(JW)$ variables. Moreover, it also suffers from symmetry, in the sense that the same cutting pattern may correspond to many $0 - W$-paths. For example, in Figure 4.11 it can be seen that there are three paths representing the pattern that uses two copies of item 1. Symmetry makes branching or cutting over the flow variables less effective. Several preprocessing techniques have been proposed for removing many arcs (and even vertices) from G in order to decrease formulation size and mitigate symmetry. Of course, there should remain at least one $0 - W$-path

representing each cutting pattern.

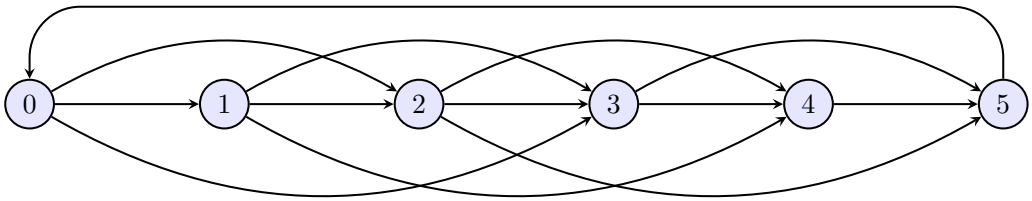


Figure 4.11: Graph for Valério de Carvalho's CSP formulation

The DW reformulation of (4.33) that keeps Constraints (4.33c) in the master yields a subproblem with a network flow structure. By also performing a path+cycle decomposition (see Section 2.5.2) and aggregating the variables that correspond to the same cutting pattern, one obtains exactly the Kantorovitch-Gilmore-Gomory formulation. By the way, as network flow problems have the integrality property, Theorem 4.1 implies that Formulation (4.33) is equally strong as KGG formulation.

As already shown in Section 4.4.2, it is possible to solve the KGG linear relaxation with the CGA, using the excellent knapsack codes available (see Note 4.6). However, if one considers (4.33) as the original formulation, it becomes possible to perform robust branching/cutting over its arc flow variables, as long as the pricing starts to be solved as a shortest $0 - W$-path problem over G. This may not be particularly effective because the preprocessing of the graph G usually fails to remove all its symmetry. Anyway, the CSP example illustrates the main point of this section: the same CG formulation can be interpreted as being derived from different original formulations. Therefore, one can choose one of those original formulations to perform robust branching or cutting. However, that choice may restrict the algorithms that can be used in the pricing.

4.6. Case Study: Software Clustering

The Software Clustering Problem is a classic software engineering problem [Parnas, 1972]. The maintenance of a large software has to be distributed to teams. The parts attributed to each team should be as independent from each other as possible. Some tools can produce from the source code a Modular Dependency Graph, which is a

connected graph $G = (V, E)$ where the vertices correspond to software modules and edges represent dependencies. Additionally, for each edge $e = \{i, j\} \in E$ there is a positive value $c_e = c_{ij}$ measuring the degree of dependency between modules i and j. A clustering is a partitioning of V into mutually disjoint non-empty clusters, that will define the parts attributed to each team. Many criteria for evaluating clusterings were proposed. The popular Turbo Modularization Quality (TurboMQ) criterion [Mitchell, 2002] is the following. Given a cluster $S \in V$, $S \neq \emptyset$, its *Cluster Factor* is defined as:

$$CF(S) = \frac{\sum_{e \in E(S)} c_e}{\sum_{e \in E(S)} c_e + \frac{1}{2} \sum_{e \in \delta(S)} c_e}.$$

Note that $0 \leq CF(S) < 1$. Higher values of $CF(S)$ indicate clusters with significant internal dependencies compared to external ones. The goal is to find a clustering that maximizes the sum of its cluster factors. The number of clusters in the partitioning may be fixed or not. We assume the latter case.

Köhler et al. [2013] introduced MIP formulations for the above problem. This task is not simple due to the TurboMQ criterion being a sum of fractional functions, necessitating the use of linearization techniques. We present their best-performing formulation. Let U be an upper bound on the number of clusters in an optimal solution. Binary variables x_i^u and t_e^u indicate whether vertex i and edge e belong to cluster $u \in [U]$, respectively. Continuous variables r^u represent the cluster factor of cluster $u \in [U]$. Some clusters u might end up without any vertices, meaning that they are not used in the solution. Those empty clusters should have $r^u = 0$. A key observation is that the clustering factor of a non-empty cluster u is:

$$r^u = \frac{2 \sum_{e \in E} c_e t_e^u}{\sum_{\{i,j\} \in E} c_{ij} (x_i^u + x_j^u)}.$$

Define auxiliary variables s_i^u to represent the product $r^u x_i^u$. The formulation follows:

$$\max\ z\ = \sum_{u\in[U]} r^u \tag{4.34a}$$

$$\text{s.t.} \quad \sum_{u\in[U]} x_i^u = 1 \qquad i \in V \tag{4.34b}$$

$$t_{ij}^u \le x_i^u,\ t_{ij}^u \le x_j^u \qquad \{i,j\} \in E, u \in [U] \tag{4.34c}$$

$$t_{ij}^u \ge x_i^u + x_j^u - 1 \qquad \{i,j\} \in E, u \in [U] \tag{4.34d}$$

$$r^u \le \sum_{i\in V} x_i^u \qquad u \in [U] \tag{4.34e}$$

$$s_i^u \le r^u,\ s_i^u \le x_i^u \qquad i \in V, u \in [U] \tag{4.34f}$$

$$s_i^u \ge r^u + x_i^u - 1 \qquad i \in V, u \in [U] \tag{4.34g}$$

$$\sum_{\{i,j\}\in E} c_{ij}(s_i^u + s_j^u) = 2\sum_{e\in E} c_e t_e^u \qquad u \in [U] \tag{4.34h}$$

$$0 \le r^u \le 1 \qquad u \in [U] \tag{4.34i}$$

$$0 \le t_e^u \le 1 \qquad e \in E, u \in [U] \tag{4.34j}$$

$$x_i^u \in \{0,1\},\ 0 \le s_i^u \le 1 \qquad i \in V, u \in [U]. \tag{4.34k}$$

Formulation (4.34) suffers from symmetry and weak linear relaxation bounds. Even with the several enhancements proposed in Köhler et al. [2013], including preprocessing, additional valid inequalities, and symmetry-breaking constraints, MIP solvers struggle to solve instances with a few dozen vertices.

Kramer et al. [2016] applied DW decomposition to that formulation. By keeping (4.34b) in the Master, the remaining constraints decompose into U identical subproblems. After a few simplifications, one obtains the following SPP:

$$\max \quad \sum_{\boldsymbol{q}\in Q} c_{\boldsymbol{q}} \lambda_{\boldsymbol{q}} \tag{4.35a}$$

$$\text{s.t.} \quad \sum_{\boldsymbol{q}\in Q} q_i \lambda_{\boldsymbol{q}} = 1 \qquad i \in V \tag{4.35b}$$

$$\boldsymbol{\lambda} \in \mathbb{Z}_+^{|Q|}, \tag{4.35c}$$

where $Q = \{\boldsymbol{\chi}(S) : S \subseteq V, S \neq \emptyset\}$. For $\boldsymbol{q} = \boldsymbol{\chi}(S) \in Q$, q_i indicates whether vertex i belongs to S and $c_{\boldsymbol{q}} = CF(S)$. The linear relaxation of (4.35) can be solved by

the CGA, the pricing subproblem being the following MIP:

$$\max \; z = r - \sum_{i \in V} \pi_i^* \tag{4.36a}$$

$$\text{s.t.} \quad t_{ij} \leq x_i, \quad t_{ij} \leq x_j, \quad t_{ij} \geq x_i + x_j - 1 \qquad \{i,j\} \in E \tag{4.36b}$$

$$r \leq \sum_{u \in V} x_i \tag{4.36c}$$

$$s_i \leq r, \quad s_i \leq x_i, \quad s_i \geq r + x_i - 1 \qquad i \in V \tag{4.36d}$$

$$\sum_{\{i,j\} \in E} c_{ij}(s_i + s_j) = 2 \sum_{e \in E} c_e t_e \tag{4.36e}$$

$$0 \leq r \leq 1 \tag{4.36f}$$

$$0 \leq t_e \leq 1 \qquad e \in E \tag{4.36g}$$

$$x_i \in \{0,1\}, \; 0 \leq s_i \leq 1 \qquad i \in V, \tag{4.36h}$$

where vectors $\boldsymbol{\pi}^*$ are RMLP optimal values for the dual variables of Constraints (4.35b). MIP (4.36) still has a weak linear relaxation. However, it does not suffer from symmetry and is much smaller than (4.34), having $O(U)$ times fewer variables and constraints. Therefore, at least for moderated-sized instances, those pricing MIPs can be solved in quite reasonable times. The BPA in Kramer et al. [2016] uses the Ryan-Foster branching, which is non-robust. Forcing a pair of vertices to belong to the same cluster actually makes the pricing MIPs a bit easier (a constraint in format $x_i = x_j$ is easily handled by a MIP solver already in its preprocessing phase, leading to the elimination of one of those variables). On the other hand, many constraints with format $x_i + x_j \leq 1$, forcing pairs of vertices to be in different clusters, can make the pricing MIPs significantly harder. In practice, that non-robustness is not an issue in the resulting BPA: the linear relaxation of (4.34) is so strong that very few branchings are needed.

We reproduce the main computational results in Kramer et al. [2016]. Table 4.1 compares the solution of (4.34) (already with the enhancements proposed in Köhler et al. [2013]) by commercial solver CPLEX 12.2 with three BPA variants. The 45 test instances were divided into three sets: small (up to 24 vertices), medium (up to 40 vertices), and large (up to 60 vertices). The time limit was 11,000 seconds. For each approach the number of solved instances and the average running times (counting only the solved instances) are given.

It can be seen that the most basic BPA variant (BPA I), the one that optimally

Table 4.1: Comparison of a MIP solver and BPA on the Software Clustering Problem

Instances sets	MIP	BPA I	BPA II	BPA III
Small instances				
Avg. time (s)	0.60	1.38	0.63	0.05
Inst. solved	15	15	15	15
Medium instances				
Avg. time (s)	762	73.5	11.0	3.09
Inst. solved	16	17	17	17
Large instances				
Avg. time (s)	5649	1673	75.1	52.3
Inst. solved	3	13	13	13

Source: Kramer et al. [2016].

solves the pricing MIP (4.36) in each iteration to generate a single column, is already better than CPLEX and can solve all 45 instances. Improved running times are obtained by the BPA II variant that calls several modified pricing MIPs to generate multiple columns in each iteration. The best running times are obtained by the BPA III variant that uses the following pricing heuristic. It can be proved that an optimal solution only contains clusters over sets S that induce connected subgraphs of G. Before the CG begins, all those connected sets S such that $|S| \leq 6$ are enumerated and stored in a table, sorted in increasing order of cardinality. The heuristic pricing consists in evaluating the reduced cost of the columns corresponding to the sets in that table. The MIP pricing (4.36) is only called when no column with negative reduced cost is found by the heuristic. In 43 out of the 45 instances, the root node solution was already integer and there was no need for branching. In the remaining two instances a single branching was enough.

The software clustering case was selected to illustrate a situation when a rather straightforward application of DW decomposition already leads to significant gains, even when the pricing is solved by a general MIP solver. While the proposed BPAs were shown to be efficient in those instances with up to 60 vertices, solving larger instances would probably require more sophisticated BPAs. The exact pricing by MIP is likely to become a bottleneck. So, one should develop pricing heuristics able to find clusters with cardinality larger than six, to reduce the number of calls to the exact MIP. At some size even a single call to solve (4.36) may become unpractically

time-consuming. In that case, one would need to improve the exact pricing, either by finding a stronger MIP formulation or even by devising specialized combinatorial algorithms for it (akin to those that are used in BPAs for Graph Coloring Problems). Another alternative for handling big software clustering instances is giving up solution optimality and creating an effective and efficient CG-based heuristic that only performs heuristic pricing (see Chapter 6).

4.7. Assessment of Column Generation for solving MIPs and LCOPs

In a similar manner to Chapter 2, we conclude this chapter by evaluating the effectiveness of a Dantzig-Wolfe reformulation and Branch(-Cut)-and-Price as a practical alternative for solving a Mixed Integer Programming problem, in comparison to directly solving it with the BCA available in a MIP solver. Our focus here is on situations where the user is seeking proven optimal solutions. For cases where good enough heuristic solutions are sufficient, the reader should refer to Section 6.5. Our assessment will be split into separate cases.

4.7.1. General MIPs

By general MIPs we mean MIPs, often coming from real-world applications, without a clear special structure, like, for example, those collected in the MIPLIB repository. There had been recent progress on this use of Column Generation [Bergner et al., 2015, Kruber et al., 2017, Khaniyev et al., 2018, Basso and Ceselli, 2022, 2023]. These works are paving the way for the development of future general-purpose MIP solvers based on Column Generation and on the BPA/BCPA that rival the existing MIP solvers based on the BCA across a substantial portion of the instances. That hope is founded on the fact that the MIPs obtained by a suitable DW reformulated can be stronger than the original MIPs, leading to smaller branch-and-bound trees. Yet, there are still formidable obstacles in that direction:

- The solution of the reformulated MIP by the BPA/BCPA is not likely to be effective unless the decomposition leads to at least several multiple subproblems. In an extreme scenario where only a single subproblem exists, the cor-

responding pricing subproblem would be nearly as large as the original MIP, and lacking any specific structure, it must be solved by a general-purpose MIP solver. There is almost no hope that this can be efficient. Decomposition into only a few subproblems is still not likely to be efficient, since the resulting pricing MIPs would still be too large. The issue with large pricing subproblems is aggravated by the fact, already mentioned in Chapter 2 on DW decomposition for LP, that very few subproblems lead to slow CG convergence.

- Therefore, it is very important to find a decomposition that leads to many smaller subproblems. However, general MIPs usually do not have a suitable block-angular matrix structure. What can be usually found is a double-bordered block-diagonal structure (see Note 2.9). Therefore, it is necessary to automatically determine not only which constraints will be kept in the master but also which variables will be shared among different subproblems. This is a complex pattern recognition task, complicated by the following tradeoff: more shared variables permit decomposition into more subproblems, which in principle has a positive effect; however, as also explained in Note 2.9, more shared variables make the CGA convergence slower.

At the time of writing, Column Generation for solving general MIPs is still a niche technique, that may supplant the standard BCA-based MIP solvers in relatively few cases. It should be noted that the latter solvers are based on much more mature techniques, created over decades of extensive research, both in academia and in industry. We believe that there is a larger potential for improvements on the less researched CG-based methods.

4.7.2. MIPs formulating specific LCOPs

How to know when DW reformulation and Column Generation are likely to be a good approach for a certain LCOP, instead of solving directly its original formulation? This is one of the most important questions that the readers of this book may pose. As expected in such a complex matter, there is no definitive answer. In fact, sometimes the choice of the method may even depend on the characteristics of the particular LCOP *instances* that one wishes to solve. For example, part of the GAP instances are better solved with the original formulation and others with the CG approach. Yet, we may provide some guidelines. Some definitions are needed:

Definition 4.9: An original LCOP formulation is said to be *compact* if it is not too large, so its linear relaxation can be "quickly" solved by a simplex or interior-point method.

This is a practical definition, not a mathematically precise one. Note that several authors use the expression *compact formulation* as a synonymous of *original formulation*. We believe that such nomenclature can be misleading and should be avoided. For example, the exponentially-sized original CSP formulation (4.23) is certainly not compact!

Definition 4.10: An original LCOP formulation is said to be *symmetric* if what is essentially a single LCOP solution has many alternative representations as MIP solutions, to the point that branching or cutting over the original MIP variables may be ineffective.

Again, we intentionally provided a practical definition of a symmetric formulation. Now we can provide the following guidelines:

- If the original LCOP formulation is compact and asymmetric the DW reformulation is likely to pay only if it provides a significantly strengthened formulation, leading to much smaller BB trees. The reason is that evaluating a tree node using CG is usually substantially more expensive than solving a compact LP (see the assessment of CG for solving LPs in Section 2.7). An example of such a situation is the GAP. The original formulation (4.16) is compact and asymmetric, but a BPA may still perform much better in instances where the reformulated MIP is significantly stronger (see Note 4.8). It is important to note that strengthened formulations are often obtained in cases where the resulting pricing subproblems are $\mathcal{NP}$-hard. Yet, CG effectiveness may also depend on having good specialized algorithms for those $\mathcal{NP}$-hard subproblems.

- If the original LCOP formulation is not compact and/or is symmetric the DW reformulation may pay even if the resulting formulation is not stronger. For example, consider the pseudo-polynomially large original Formulation (4.33) for the CSP. Its DW decomposition leads to a NFP subproblem. As it has the integrality property, the resulting reformulation (equivalent to the KGG formulation) will not be stronger. Yet, for large values of W, it can be much

more practical to solve the reformulated problem by CG.

- If the original LCOP formulation is not compact and/or is symmetric the DW reformulation may pay even if the pricing still has the be solved as a not-so-strong MIP. The case study in Section 4.6 illustrates that situation.
- An additional circumstance that may favor the use of a DW reformulation is when it is possible to use effective branching or cutting (usually non-robust) over the generated variables, something that is simply not possible in the original formulation.

Branch-and-Price and Branch-Cut-and-Price algorithms are the best known way of solving many important $\mathcal{NP}$-hard combinatorial problems. Yet, successfully designing such algorithms can be tricky. A key point to consider is the pricing subproblems' structure and how branching and cutting operations may affect it.

Notes

4.1. The fundamental Theorem 4.1 was proved in Geoffrion [1974] in the context of Lagrangian Relaxation for integer programming. Magnanti et al. [1976] realized that it also applied to DW decomposition for integer programming.

References that discuss general CG for integer programming include Barnhart et al. [1998], Wilhelm [2001], Vanderbeck and Savelsbergh [2006], Gamrath [2010], Vanderbeck and Wolsey [2010], Lübbecke [2011], Sadykov [2019], Simonetti et al. [2022]. The book Desaulniers et al. [2005] starts with a primer on CG [Desrosiers and Lübbecke, 2005] and has another 11 contributed chapters covering many aspects of the subject. Integer programming textbooks like Conforti et al. [2014], Wolsey [2020] also have chapters on CG. Very recently, the preprint of a full book on the topic was made available as Desrosiers et al. [2024].

4.2. DW IP reformulation by convexification. The scheme described in Section 4.1 is known as *DW IP reformulation by discretization* because the resulting reformulated IP (4.3) has one variable for each point in $Int(P)$. In an alternative scheme, known as *DW IP reformulation by convexification* and fully described in Vanderbeck and Savelsbergh [2006] or in Gamrath [2010], the reformulated IP would only have variables for points in $Ext(Conv(Int(P)))$, the extreme integer points in P. This is correct because the other points in $Int(P)$ can be represented as convex combinations of those extreme integer points. The resulting subproblem IP (4.6) would remain the same (actually, in order to be technically correct, one would have to use an IP solving method that guarantees that the returned solution is in $Ext(Conv(Int(P)))$, even if there are alternative optimal solutions in $Int(P) \setminus Ext(Conv(Int(P)))$).

If $Int(P) \subseteq \mathbb{B}^n$, as often happens in the context of LCOPs, then $Int(P) = Ext(Conv(Int(P)))$ and both schemes are actually identical. Consider the cases where $Int(P) \supset Ext(Conv(Int(P)))$, so the alternatives are distinct. In general, there is no practical gain in choosing IP DW reformulation by convexification. A rare exception can be found in Desaulniers [2010] and Desaulniers et al. [2016] where the authors realized that the restriction to points in $Ext(Conv(Int(P)))$ could accelerate their pricing algorithm. However, there is a big limitation in IP DW reformulation by convexification: as the generated variables λ can not be assumed to be integer, it is not possible to perform branching or cutting on them. This already rules out that scheme in the advanced BCP algorithms that rely on strong non-robust cuts over the generated variables.

4.3. Branching constraints/cuts in the subproblems. The robust BPA scheme presented in Section 4.2.2 treats branching constraints over original variables as additional constraints that are kept in the Master. There is an alternative BPA scheme that may treat those branching constraints as if they are included in the subproblems.

Consider an MLP in format (2.15) but such that all subproblems are distinct (the case $K = U$) and branching constraints over a single original variable x_j^k. The children nodes would have their subproblem k redefined to polyhedra $P'^k = P^k \cap (x_j^k \leq \lfloor x_j^{k*} \rfloor)$ and $P''^k = P^k \cap (x_j^k \geq \lceil x_j^{k*} \rceil)$, respectively. Such

a branching is *usually* still robust. For example, in the GAP, a constraint $x_j^k \leq 0$ can be enforced by removing from the RMLP all incompatible columns, corresponding to points $\boldsymbol{q}$ such that $q_j = 1$ and $q_{J+k} = 1$, and by eliminating job j from subproblem k, forbidding the generation of new columns with that feature. A constraint $x_j^k \geq 1$ is enforced by removing the incompatible columns from the RMLP and by forcing job j to be in any solution from subproblem k. This can be done by eliminating job j from subproblem k, solving the residual Binary KP with machine capacity $W - w_j^k$, and then including j in the solution. This scheme is robust. In both children nodes, the pricing can still be solved by a black-box Binary KP solver. In the $\geq$ children, it is even possible to remove the row corresponding to job j from the RMLP. According to fundamental Theorem 4.1, including a constraint in the subproblems could possibly be stronger than keeping it in the master. However, in practice, this is seldom the case for single original variable branching constraints. In the GAP example, as in any problem where $Int(P^k)$ only has binary vectors, we have that $Conv(Int(P'^k)) = Conv(Int(P^k)) \cap (x_j^k = 0)$ and $Conv(Int(P''^k)) = Conv(Int(P^k)) \cap (x_j^k = 1)$ (the proof is left as an exercise to the reader). Therefore, there is no gain in strength. We do not enthusiastically recommend implementing this BPA scheme for problems like GAP. The gains by having slightly smaller Knapsack problems in the pricing and slightly smaller RMLPs in some nodes may not compensate for the increased coding effort.

Note that the scheme can not be applied to cases where the original formulation leads to many identical subproblems. As already said, in those cases, branching over a single variable x_j^u is ineffective due to solution symmetries, and also because it makes formerly identical subproblems distinct, leading to additional pricing subproblems. Moreover, it is also not possible to branch on the aggregated original y variables defined in (2.19c). This happens because a branching constraint like $y_j^k = \sum_{u \in U(k)} x_j^u \geq \lceil y_j^{k*} \rceil$ has non-zero coefficients in all subproblems in $U(k)$. If such a constraint is not kept in the master, those subproblems will not be independent anymore, completely breaking the pricing structure. For example, consider the CVRP case and its original formulation (4.27). It is simply not possible to introduce in (4.28) the branching constraint $x_e = y_{ij} + y_{ji} = \sum_{u \in [U]} (y_{ij}^u + y_{ji}^u) \geq 1$, for some edge $e = \{i, j\}$. Observe that forcing the pricing subproblem in the child node corresponding to $x_e \geq 1$ to only generate routes that contain edge e is wrong since other

routes not having e are still needed.

If the original formulation has K groups of distinct subproblems, general cuts that only have non-zero coefficients on the original variables $\boldsymbol{x}^k$ corresponding to a certain subproblem $k \in [K]$ such that $|U(k)| = 1$ can be introduced into subproblem polyhedron P^k. However, it is unlikely that this will be robust. In the GAP robust BCPA example shown in Section 4.4.1, cut $x_1^2 + 2x_2^1 + 2x_3^1 + x_4^2 \leq 3$ is equivalent to $-x_1^1 + 2x_2^1 + 2x_3^1 - x_4^1 \leq 1$. The latter cut can be introduced into P^1. However, it would destroy its KP structure. The resulting pricing would need to be solved as a MIP.

Overall, branching/cuts over the original variables can be handled robustly by including them in the master. Sometimes, it is possible to include them in the subproblems robustly, with some potential gains in performance.

4.4. Non-proper generated variables. Note 2.7 proposes a categorization of the variables involved in a DW decomposition. However, in DW IP reformulation, the generated variables may be additionally classified as proper or non-proper.

Definition 4.11: Non-proper variable. A generated variable is *non-proper* if it can not assume a positive value in a feasible integer solution.

Non-proper variables may appear with positive values in fractional DWM solutions, weakening the resulting bounds. Why not always prevent the pricing from generating non-proper variables? In reality, this is a trade-off. As seen in Section 3.4.2 on CVRP, elementary routes relaxations like q-routes and ng-routes deliberately include additional non-proper variables in the formulation to make the pricing subproblem much more tractable. For example, ng-routes are favored in modern BCPAs because they achieve an excellent balance between bound quality and pricing difficulty. Another example of a situation where non-proper variables may exist in the CSP is given in Exercise E 4.9. In situations where the DW reformulation for IP leads to a single subproblem non-proper variables always exist.

Recognizing the non-proper variables is essential in diving heuristics, as described in Chapter 6. Indeed, one can not fix a non-proper variable to a positive integer value without causing infeasibility.

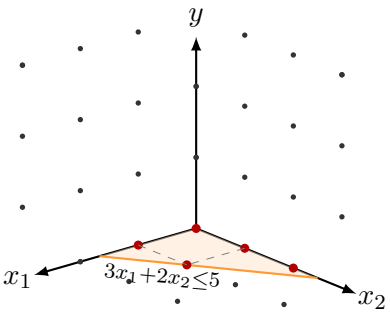
4.5. Projection of cuts in an extended space. A DW reformulation can be viewed as a method for obtaining an extended formulation with an exponential number of additional variables, as can be seen in the explicit reformulated IP (4.2) that has both x and λ variables. As shown in Section 4.3.1, a single cut over the latter variables may project into more than one cut in the original x space. This is not something particular to DW reformulation, it may happen with extended formulations in general.

For example, consider the polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^2 : 3x_1 + 2x_2 \leq 5, \boldsymbol{x} \geq \mathbf{0}\}$, shown in Figure 4.12a, which provides a formulation for $X = \{(0,0), (0,1), (0,2), (1,0), (1,1)\}$. Polyhedron $P' = \{(x_1, x_2, y) \in \mathbb{R}^3 : 3x_1 + 2x_2 \leq 5, y \geq x_1, y \geq x_2, (x_1, x_2, y) \geq \mathbf{0}\}$ provides an extended formulation for X. As can be seen in Figure 4.12b, $Proj_{\boldsymbol{x}}(P') = P$. Therefore, that extended formulation is not stronger. However, suppose one adds the cut $x_1 + y \leq 2$ to P', which is represented by the green hyperplane in Figure 4.12c. Then, the projection of the resulting polyhedron becomes exactly $Conv(X)$. This means that a single new cut in the extended space (together with other already existing inequalities) implies the two facet-defining inequalities $x_1 \leq 1$ and $x_1 + x_2 \leq 2$ in the original space (Figure 4.12d).

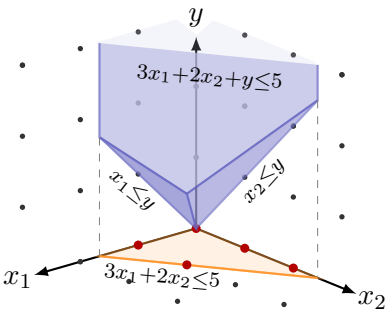
4.6. Knapsack solvers. The most basic knapsack problem variant is defined as follows:

Definition 4.12: Binary Knapsack Problem (Binary KP). Instance: J items; profits c_j and integer weights w_j, $j \in [J]$; integer capacity W. Solutions: Subsets J' of $[J]$ such that $\sum_{j \in J'} w_j \leq W$. Goal: maximize $\sum_{j \in J'} c_j$.

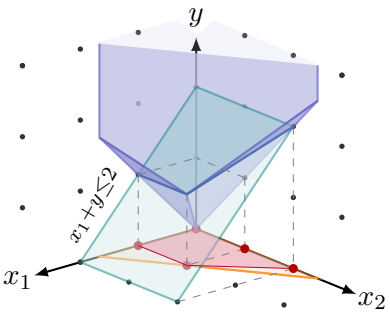
The Binary KP is weakly $\mathcal{NP}$-hard and can be solved in pseudo-polynomial $O(JW)$ time by a Dynamic Programming Algorithm. This is already a guarantee that a Binary KP can only be difficult if W is large. Yet, after much research, as reported in the books Martello and Toth [1990] and Kellerer et al. [2004], there are algorithms with a practical performance that is much better than that. The best existing algorithms have been available, for over 20 years, as open-source C codes on the page kept by David Pisinger [Pisinger]. The performance of those algorithms is very impressive, on the hardest class of



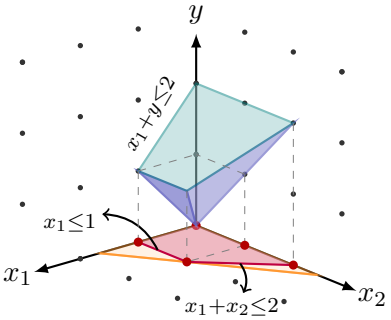
(a) Polyhedron P



(b) Extended polyhedron P' and its projection P



(c) Insertion of cut $x_1 + y \leq 2$ over polyhedron P'



(d) Projection of the resulting polyhedron matches $Conv(X)$

Figure 4.12: Example of projection of cuts in extended space

instances with $J = 10,000$ and $W = 10^6$, the average running times reported in Pisinger [2005] for the Combo code is around 20 milliseconds!

Pisinger's page also contains highly efficient codes for other variants, including the following ones: Bounded KP, where the same item can appear multiple times in J', up to a given bound u_j (the Integer KP being the particular case where the bounds are infinite); and Multiple Choice KP, where the items are partitioned into disjoint classes and at most one item from each class can appear in J'.

4.7. ISP and MWCP solvers. CG-based algorithms for the Graph Coloring Problem need to solve Independent Set Problems (ISP) (see Definition 4.8) for their pricing. The ISP is equivalent to the Maximum Weighted Clique Problem (MWCP) in the complement graph. Thus, an ISP solver can solve MWCP instances and vice-versa. Specialized combinatorial branch-and-bound algorithms provide the best performance in finding proven optimal ISP/MWCP solutions. To our knowledge, the best available open-source codes are TSM-MWC Jiang and Li [2017] based on the algorithm by Jiang et al. [2018] and the hybrid ISP solver Held [2022] based on the algorithm by Held et al. [2012] for lower density graphs and CLIQUER algorithm Niskanen and Östergård [2010] by Östergård [2001] for higher density graphs. San Segundo et al. [2019] provide an experimental comparison between the best available MWCP solvers, and also present an efficient branch-and-bound algorithm for this problem, whose implementation is however not available online. According to them, instances with up to several hundreds of nodes can be solved in a short time, except for the very sparse (for the ISP) graphs. When graphs are dense, much larger instances (with up to several thousand nodes) may be solved. Finally, we can mention that modern MIP solvers are competitive in solving small and medium-sized ISP instances with very sparse graphs.

4.8. GAP solvers. General MIP solvers perform reasonably well on the natural formulation of the Generalized Assignment Problem. They can recognize the structure of Constraints (4.16c)-(4.16d) and separate cuts valid for the binary knapsack polyhedra defined by $Conv(Q^k)$, $k \in [K]$, like lifted cover cuts (see

Wolsey [2020]). However, they often fail in hard instances. As reported in Pessoa et al. [2020], a top commercial MIP solver, with a time limit of 1 hour, could only solve 1 out of the 30 instances with up to 90 jobs proposed in Nauss [2003]; while a BCPA could solve 28 of those instances with the same time limit. The harder GAP instances are usually those with tight capacities W_k and a negative correlation between costs c_j^k and loads w_j^k. It is worth reducing gaps by fully convexifying the knapsack constraints for those instances. The first GC-based algorithm for GAP was the BPA in Savelsbergh [1997]. Pigatti et al. [2005] improved that BPA by adding dual stabilization, a technique presented in Chapter 7. Avella et al. [2010] got good results using Fenchel Cuts as an alternative way of fully convexifying the knapsack constraints, as described in Note 4.15. The algorithm in Posta et al. [2012] achieved the current best results for most instances using Lagrangian Relaxation, a method closely related to CG, as described in Section 5.4.2. Finally, the generic BCPA in VRPSolver [Pessoa et al., 2020], even though it solves the problem as a "VRP" (see Note 4.12), not taking advantage of specialized knapsack algorithms in the pricing, may obtain the best results on instances with very few jobs per machine.

4.9. The true CSP formulation in Kantorovich (1939). Almost all authors in the last 20 years attributed the weak CSP Formulation (4.23) to Kantorovich [1960], a translation of Kantorovich [1939]. We read that reference carefully, including its original Russian version, and found no trace of that formulation! Instead, we discovered that Kantorovich proposed a CSP formulation, including the variant with multiple stock sizes, similar to Gilmore and Gomory [1961], but implicitly assuming that the number of possible cutting patterns is small, so they can be enumerated by hand.

The main Kantorovich's CSP model (page 380 in Kantorovich [1960]) corresponds to the following variant. Suppose that a factory produces a certain article. Each article requires d_j units of item j, $j \in [J]$. There are K stock types. For each stock type $k \in [K]$, Q^k is its set of cutting patterns and u^k is the number of units of that stock that are available. The objective is to produce the maximum number of articles. The symbols used in that description were adapted to match those in Section 4.4.2, but apart from that we now present the model in its original phrasing:

"We have the following conditions for the determination of the unknowns $\lambda_{\boldsymbol{q}}^k$*:*

$$
\begin{aligned}
&1)\ \lambda_{\boldsymbol{q}}^k \geq 0 \text{ and equal to whole numbers};\\
&2)\ \sum_{\boldsymbol{q}\in Q^k} \lambda_{\boldsymbol{q}}^k = u^k;\\
&3)\ \frac{\sum_{k\in[K]} \sum_{\boldsymbol{q}\in Q^k} q_1 \lambda_{\boldsymbol{q}}^k}{d_1} = \frac{\sum_{k\in[K]} \sum_{\boldsymbol{q}\in Q^k} q_2 \lambda_{\boldsymbol{q}}^k}{d_2} = \ldots = \frac{\sum_{k\in[K]} \sum_{\boldsymbol{q}\in Q^k} q_J \lambda_{\boldsymbol{q}}^k}{d_J}
\end{aligned}
$$

and that their common value be a maximum."

Translating that to modern notation, we obtain:

$$
\begin{aligned}
\max \quad & z && && (4.37a)\\
\text{s.t.} \quad & \sum_{k\in[K]} \sum_{\boldsymbol{q}\in Q^k} q_j \lambda_{\boldsymbol{q}}^k = d_j\, z && j \in [J] && (4.37b)\\
& \sum_{\boldsymbol{q}\in Q^k} \lambda_{\boldsymbol{q}}^k = u^k && k \in [K] && (4.37c)\\
& \boldsymbol{\lambda} \geq \mathbf{0} \quad \text{and integer}, && && (4.37d)
\end{aligned}
$$

where variable z represents the "common value", which is nothing but the number of articles produced. Kantorovich did not restrict that formulation to 1D cutting. On the contrary, several of the mentioned cases of use are 2D cutting (sheets of glass or iron, boards, etc).

The chapter proceeds by presenting *"a very simple problem"* that corresponds to the standard 1D CSP with a single stock type: how to cut 100 copies of each of three items with lengths 2.9, 2.1, and 1.5 using the minimum number of stocks of length 7.4? In other words, the instance $J = 3$, $w = (\,2.9\ 2.1\ 1.5\,)$, $d = (\,100\ 100\ 100\,)$, and $W = 7.4$. Six cutting patterns are enumerated and the optimal solution is given. Finally, a problem corresponding to an instance of his more general CSP model is posed: if an article requires one copy of each of three items, with lengths 2.9, 2.1, and 1.5, and there are 100 stocks of length 7.4 and 50 stocks of length 6.4, what is the maximum number of articles that can be produced? The optimal solution (producing 161 articles) is obtained with his *Method of Resolving Multipliers*, which essentially consists of dualizing some constraints, solving the Lagrangian Dual Problem (see Chapter 5) and

then recovering the primal solution (which is rounded to integers if necessary).

The CSP is only a small part of Kantorovich [1939]. The main text proposes LP models for nine families of industrial production problems. Appendix 1 presents the Method of Resolving Multipliers. Appendix 2 is the detailed numerical solution of a real-world problem (having 36 variables and 12 constraints) from a plywood plant. Finally, Appendix 3, titled "Theoretical Supplement", provides analytical and geometrical proofs of the existence of optimal resolving multipliers.

The subsequent development of linear programming in Stalinist Soviet Union was much delayed because some authorities found that its concepts clashed with the orthodox economic theories of the country (see Gardner [1990], Polyak [2002], Vershik [2007], Bollard [2020], Boldyrev and Düppe [2020], Ellman [2022] and Uchoa and Sadykov [2024] for more details on the historical facts mentioned in this paragraph). Kantorovitch had ambitious goals for linear programming and hoped that it could be used not only on local-level industrial problems but also for centralized economic planning. In 1942 he submitted a manuscript named *The Best Use of Economic Resources* to Gosplan, the powerful Soviet planning agency. After its strong condemnation, he was forced to keep it unpublished. The ideological objections to linear programming are related to the Labor Theory of Value, which asserts that *the value of a good is 100% determined by the amount of labor required to produce it*. It was observed that the dual variables (the conspicuous optimal resolving multipliers) could be interpreted as prices that would not fit in that theory, which is central to Marxism. Note that the then-competing "capitalist" Marginalist Theory of Value states that *the value of a good is given by how much gain one additional unit of it brings*, a concept that is quite consistent with LP duality. In fact, linear programming and its duality theory would later become a major influence on Western economics (see for example the classic book Dorfman et al. [1958]). Anyway, only in 1956, after Stalin's death, Kantorovitch could finally openly discuss and teach linear programming.

4.10. The 0-th Column Generation algorithm. General LP was banned in the USSR during Stalin's rule. Yet, Kantorovich could publish a bit on specific LP applications: two papers on the transportation problem and Kantorovich

and Zalgaller [1951], a 200-page book only on the CSP. The book was developed from a real application: planning the cutting of steel in the Leningrad Egorov Railroad Car plant, which happened in the late 1940s. Besides 1D cutting, that book also deals with 2D rectangular guillotine and 2D circular cutting problems. The solution methods are based on the model introduced in Kantorovich [1939], but viewed as an LP, not as an IP. In other words, the fractional use of a cutting pattern was allowed. The modeling assumption is that item demands represent proportions. For example, suppose that the manufactured articles require 2 copies of item 1, 4 copies of item 2, and 1 copy of item 3. The actual number of articles that will be manufactured is unknown, as the factory will be operated for an undetermined period of time. So, the CSP is solved with demands $d = (\,2\ 4\ 1\,)$. Its fractional solution will determine the proportions in which each cutting pattern should be used. The CSP optimal solution value is the average number of stocks used per article.

Unlike in Kantorovich [1939], the cutting patterns are not assumed to be enumerated in advance. In fact, *Kantorovich and Zalgaller [1951] proposes an iterative approach that can be regarded as a complete column generation algorithm.* They propose finding improving patterns by what we now call reduced costs and state the optimality criterion.

We reproduce here the example found in the first edition of that book [1] (published in Russian and never translated to Western languages), the steps for solving the CSP instance having $J = 3$, $w = (\,1400\ 950\ 650\,)$, $d = (\,2\ 4\ 1\,)$, and $W = 5000$. We kept the notation very close to the original, except that here the dual variables (called "indices" in the original text) are notated as π_1, π_2, and π_3. The starting solution only uses single-item patterns: $(\,3\ 0\ 0\,)$ with value 2/3, $(\,0\ 5\ 0\,)$ with value 4/5, $(\,0\ 0\ 7\,)$ with 1/7; solution cost is ≈ 1.61. After two patterns are generated, the solution is $(\,3\ 0\ 1\,)$ with value 2/3, $(\,0\ 5\ 0\,)$ with value 71/91, $(\,0\ 1\ 6\,)$ with value 1/18; solution cost is ≈ 1.51.

[1] We thank Alexander Lazarev and Michael Khachay for photographing page by page a copy available at the Moscow State University library.

At that point, the indices are calculated by solving:

$$\begin{cases} 3\pi_1 + \pi_3 = 1 \\ 5\pi_2 = 1 \\ \pi_2 + 6\pi_3 = 1 \end{cases} \Rightarrow \begin{array}{l} \pi_1 = 13/45 \\ \pi_2 = 1/5 \\ \pi_3 = 2/15. \end{array}$$

By solving an integer knapsack problem, the improving pattern $(1, 3, 1)$ is found ($13/45 + 3/5 + 2/15 = 46/45 > 1$). Associate variables x, y, z to the current patterns and θ the new one. We have that:

$$\begin{cases} 3x + \theta = 2 \\ 5y + z + 3\theta = 4 \\ x + 6z + \theta = 1 \end{cases} \Leftrightarrow \begin{cases} 3x = 2 - \theta \\ 5y + z = 4 - 3\theta \\ x + 6z = 1 - \theta. \end{cases}$$

Solving the 3×3 linear system (considering the θ terms as constants in the RHS), the following expressions are obtained:

$$x = \frac{2-\theta}{3}, \qquad z = \frac{1-2\theta}{18}, \qquad y = \frac{71-52\theta}{90}.$$

So, when θ increases, the first value which nullifies is z (when $\theta = \frac{1}{2}$). Thus $(0, 1, 6)$ is replaced by $(1, 3, 1)$. It can be deduced that $x = \frac{1}{2}$ and $y = \frac{1}{2}$. The cost of the new solution is thus 1.5. Recalculate the indices by solving

$$\begin{cases} 3\pi_1 + \pi_3 = 1 \\ 5\pi_2 = 1 \\ \pi_1 + 3\pi_2 + \pi_3 = 1 \end{cases} \Rightarrow \begin{array}{l} \pi_1 = 3/10 \\ \pi_2 = 2/10 \\ \pi_3 = 1/10. \end{array}$$

By solving another integer knapsack problem, it is shown that no improving pattern exists and that the current CSP solution is optimal. This means that the proposed CG does not use the Method of Resolving Multipliers. Instead, *it uses something similar to the Revised Simplex Algorithm, anticipating Dantzig [1953] (for a particular case).*

But how the integer knapsack problems were solved? Kantorovitch and Zalgaller proposed the so-called *Scale of Indices method*, which can be viewed as a graphical version of a Dynamic Programming algorithm. The optimal scale of indices corresponding to the last knapsack problem in the above example is shown in Figure 4.13 (this figure and Figure 4.14 are almost identical to

the figures found in their book). The indices values are multiplied by 10 in order to make them integers. The scale of indices indicates the best solution value for each knapsack capacity up to W, the solutions themselves are also indicated. As the value for $W = 500$ is 10 (1.0 after dividing it by 10), it is shown that there is no improving pattern and the current CSP solution is optimal.

The optimal scale of indices is iteratively constructed using two sheets of semi-transparent calc paper, as illustrated in Figure 4.14. Two copies of the starting scale of indices (shown at the top of that figure) should be plotted, one on each sheet of semi-transparent paper. The starting scale should have the values corresponding to single-item solutions (1 for capacity 65, 2 for capacity 95, and 3 for capacity 140), plus some possibly non-optimal values for larger values of capacity. Then one of the copies is shifted, as illustrated in the middle of the figure. The dashed regions indicate better knapsack solutions. Those improvements are marked, leading to the improved scale shown at the bottom of the figure. The procedure is repeated until no improvement is possible.

In today's context, the approach seems odd. However, in the era before computers, it was a widespread practice for engineers to utilize mechanical analog tools, such as slide rules, to speed up computations. Due to its parallel structure, the Scale of Indices method is capable of evaluating several potential improvements simultaneously, convergence is usually fast. Nonetheless, similar to most mechanical analog techniques, the method suffers from low numerical precision. The Dynamic Programming knapsack algorithm with explicit stage-by-stage numerical calculations proposed by Richard Bellman in the mid-50s can have arbitrary precision.

Kantorovich and Zalgaller [1951] is an extensive and mature work that certainly had a significant impact in the USSR, even deserving a second edition in 1971. *Yet, it had a negligible impact in the Western world, where is nearly unknown until today.* In particular, the proposed CGA did not influence the mainstream development of the field. This is why it was called the "0-th Column Generation algorithm" in Uchoa and Sadykov [2024]

4.11. CSP and BPP CG-based solvers. The first widely known practical use of column generation was the method proposed in Gilmore and Gomory [1961]

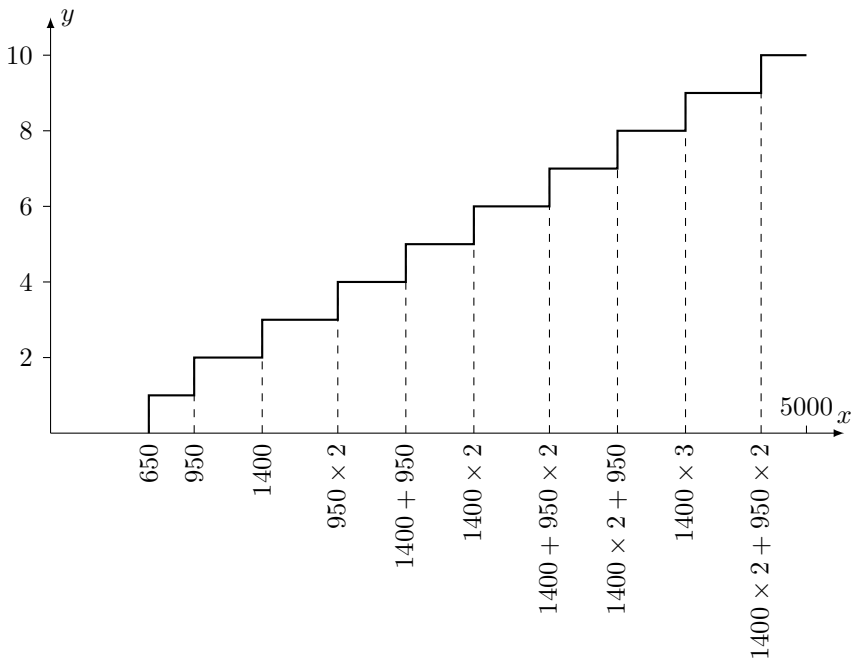


Figure 4.13: An optimal "Scale of Indices".

for the CSP (including the variant with multiple stock sizes). They independently rediscovered the cutting-pattern-based formulation in Kantorovich [1939, 1960], which we will refer to as Kantorovich-Gilmore-Gomory formulation and proposed solving its linear relaxation by CG, using Dynamic Programming to solve the integer knapsack pricing subproblems. A more advanced version of that method, which was already being used in the routine operation of a large paper mill, appears in Gilmore and Gomory [1963]. That version proposes alternative methods for solving the knapsack subproblems and even includes a number of other practical constraints, like limits on the number of knives available for cutting the paper rolls. Extensive computational results are presented and discussed, including the effects of having a larger/smaller stock size or multiple stock sizes on the waste. Gilmore and Gomory [1965] considers cutting of 2D rectangular stocks.

Many subsequent works improved on those seminal papers, often proposing better ways of finding an integer solution from a fractional solution of the CGA. Given the remarkable strength of the Kantorovich-Gilmore-Gomory

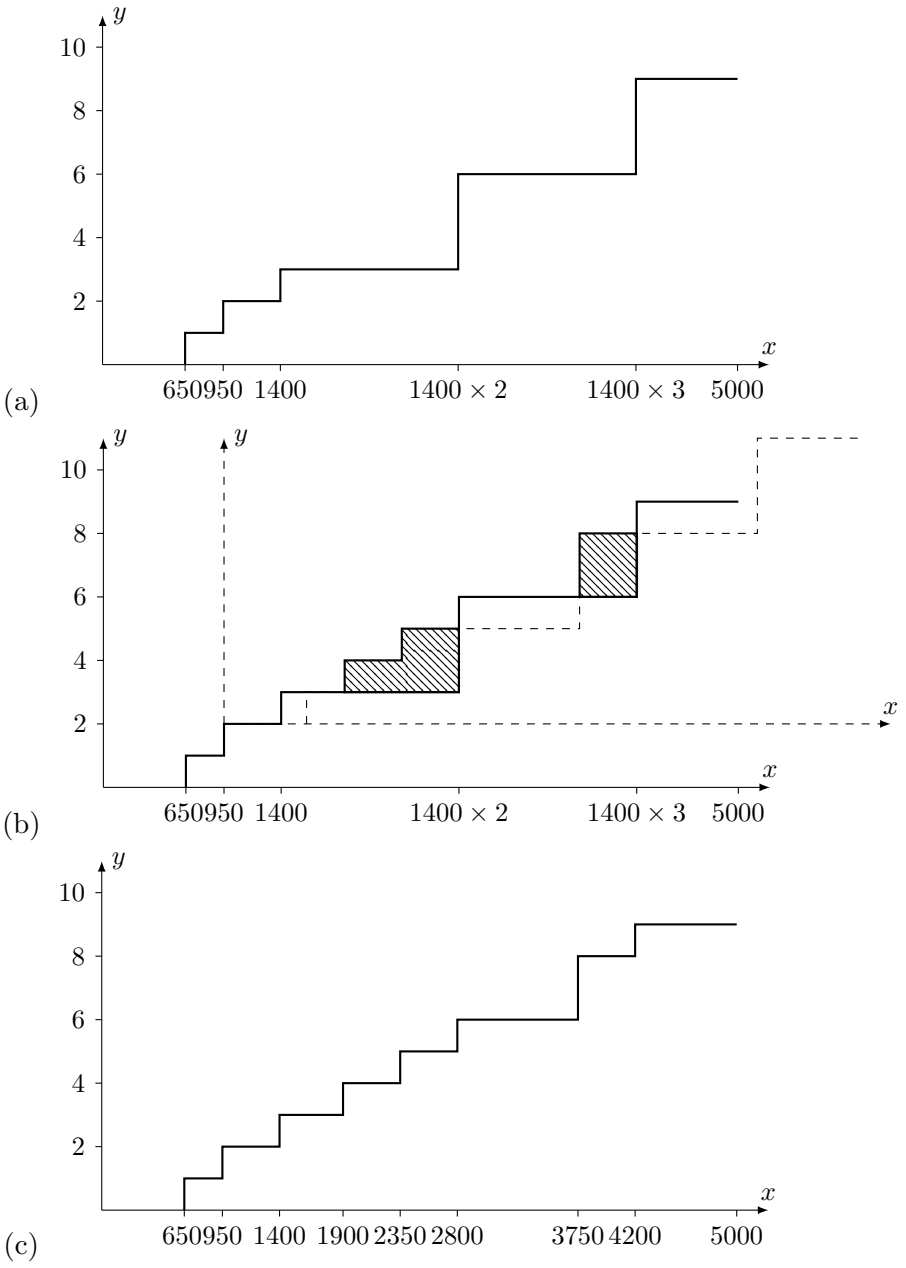


Figure 4.14: The initial scale of indices is shown on (a). In (b), a copy of that scale shifted by $(950, 2)$ obtains the improvements depicted as dashed regions. The resulting improved scale is shown in (c). The procedure should be repeated until no improvement is possible.

bound (see Conjecture 4.1), in most instances, the obtained solutions could be proved to be near-optimal or even optimal. Yet, the first CG-based exact algorithm for the CSP (actually, for the Bin Packing Problem, a.k.a. Binary CSP) was the BPA in Vance et al. [1994]. Devising a branching rule that is both practically and theoretically good in BPAs or BCPAs for the BPP/CSP is a challenge and many alternatives have been proposed, including Vance [1998], Vanderbeck [1999], Valério de Carvalho [1999], Belov et al. [2005], Belov and Scheithauer [2006], Delorme and Iori [2020], Wei et al. [2020], Pessoa et al. [2021b], de Lima et al. [2022], Baldacci et al. [2023] and da Silva and Schouery [2024].

The CSP and BPP are strongly $\mathcal{NP}$-hard but very well-solved in practice. A respectable fraction of the instances can be quickly solved by only checking if some fast combinatorial lower bound matches the value of a solution obtained by simple heuristics (like First-Fit Decreasing) or by more sophisticated ones (like Alvim et al. [2004], Loh et al. [2008], Quiroz-Castellanos et al. [2015]). As shown in Fekete and Schepers [2001] (see also Clautiaux et al. [2010], Alves et al. [2016]), those combinatorial bounds correspond to *dual feasible functions* for approximating the Kantorovich-Gilmore-Gomory bound. The very advanced CG-based algorithms that are being proposed recently compete on solving adversarial instances, created with the intent of being hard, and even in those instances, finding a solution proven to be not more than one unit away from the optimal is quite easy. In other words, they are fighting over at most one stock! Yet, that research effort is definitely worth it because there are lots of other more complex cutting and packing problems (including 2D and 3D variants) that are not so well-solved by CG or by any other known method. In those problems, the CG bounds are still very good but not so almost incredibly tight (Conjecture 4.1 may not hold) and the pricing subproblems are much harder. So, BPAs and BCPAs need to close significant gaps and need to do so without making the subproblems intractable.

4.12. VRP BCP solvers. As described in Section 3.4.2, in the early 2000s BCAs were the dominant approach for the CVRP. BPAs based on the MLP (4.29) perform poorly. However, after Fukasawa et al. [2006], Baldacci et al. [2008, 2011], Røpke [2012], Contardo and Martinelli [2014], Pecin et al. [2017b], it is

known that algorithms that combine CG with cuts are much better than BCAs on most CVRP instances. Yet, there are still some instances, those where an optimal solution has only a few very long routes, where BC algorithms over Formulation (3.6) still perform better. In fact, the algorithm in Fukasawa et al. [2006] checked at the root node whether the CG was experiencing severe slow convergence (an issue that is typical on instances with long routes) and sometimes could automatically switch to a BCA similar to the one in Lysgaard et al. [2004].

However, other VRP variants are still more challenging for BCAs, which are rarely the best option.

Definition 4.13: Vehicle Routing Problem with Time Windows (VRPTW). Instance: Directed graph $G = (V, A)$, where $V = \{0\} \cup V_+$, vertex 0 represents a depot and V_+ the set of customers; arc costs c_a and integer arc times t_a, $a \in A$; integer positive demands d_i and integer time windows $[a_i, b_i]$, $i \in V_+$; and integer vehicle capacity W and integer time horizon T. Solutions: sets of routes in G that, together, visit all customers exactly once. A route is a vertex-elementary cycle (a cycle that does not repeat vertices) that passes by the depot such that the sum of the demands of the customers in it does not exceed W. Moreover, routes are assumed to start at time 0, they should visit customers at a time within their time windows (it is possible to arrive early and wait) and then return to the depot before T. Goal: minimize the sum of the cost of the arcs in the routes (some adopt the hierarchical goal of first minimizing the number of routes and then the total cost).

The VRPTW is perhaps the second most widely studied [2] VRP variant. It

[2] The VRPTW is "overrepresented" in the CG literature, appearing more often than the CVRP. This has historical reasons: the early BP algorithms (see Note 4.19) worked very well on VRPTW instances with narrow time windows, but failed in the more classic CVRP.

would be possible to have a VRPTW formulation akin to Formulation (3.6):

$$\min \quad \sum_{e \in A} c_a y_a \tag{4.38a}$$

$$\text{s.t.} \quad \sum_{a \in \delta^-(i)} y_a = 1 \qquad i \in V_+ \tag{4.38b}$$

$$\sum_{a \in \delta^+(i)} y_a = 1 \qquad i \in V_+ \tag{4.38c}$$

$$\sum_{a \in \delta^-(S)} y_a \geq k(S) \qquad S \subseteq V_+ \tag{4.38d}$$

$$\boldsymbol{y} \in \mathbb{Z}_+^{|A|}, \tag{4.38e}$$

where $k(S)$ is the minimum number of routes needed for visiting all customers in S, taking into account both capacities and time windows. Inequalities (4.38d), proposed by Kohl et al. [1999] and known as k-Path Cuts, are fundamentally different from RCCs (3.6c). While the separation of RCCs is $\mathcal{NP}$-hard but well-done in practice by heuristics, the separation of k-Path Cuts (even of only 2-Path Cuts, the subfamily where $k(S) = 2$) is a much harder problem, since its decision version does not belong to $\mathcal{NP}$. To merely check if a given inequality in format (4.38d) is indeed a valid k-Path Cut, one has to calculate $k(S)$, a strongly $\mathcal{NP}$-hard problem. This means that Formulation (4.38) is not a promising start for a BCA. The existing BCAs for the VRPTW (like Bard et al. [2002]) start from extended formulations that use auxiliary variables and weak Big-M constraints (Note 3.12) for controlling the time windows. The VRPTW is much better solved by BCPAs, like Kohl et al. [1999], Kallehauge et al. [2006], Desaulniers et al. [2008], Jepsen et al. [2008], Baldacci et al. [2011], Pecin et al. [2017a], Sadykov et al. [2021].

The VRP has hundreds of variants, perhaps even thousands depending on the adopted taxonomy. Nearly every day some new CG-based algorithm for a VRP variant is proposed, we have no hope of mentioning them all. However, there is a generic BCPA solver that obtains very good results over a large number of variants, including the most classic ones. VRPSolver [Pessoa et al., 2020] defines an optimization model, introducing the concepts of mapping, packing sets, and elementarity sets, that generalize and unify many previous ideas. The VRPSolver model is generic enough to encompass many

VRP variants and a significant number of other non-VRP problems too (like GAP, BPP, Parallel Machine Scheduling, etc). The BCPA in VRPSolver incorporates many advanced elements proposed by different authors in the last two decades (see Poggi and Uchoa [2014] and Costa et al. [2019] for surveys) and was coded on top of BapCod CG framework [Sadykov and Vanderbeck, 2021]. It uses the pricing algorithms in [Sadykov et al., 2021]. Modeling with VRPSolver (code available at [VRPSolver]) can be tricky and users often need creativity to fit more complex problems into its model. Callback routines can be used to separate cuts valid for a specific application. *Effective modeling with VRPSolver may require knowledge about how the underlying BCPA works* (a knowledge offered in this book!). VRPSolver obtained most of the current best results for CVRP and VRPTW (check CVRPLIB page Uchoa et al. [2014] for continuously updated results). BCPAs using VRPSolver appeared in more than 40 published articles, including those on the following VRP variants: Heterogeneous Fleet [Pessoa et al., 2018, 2020], Backhauls [Queiroga et al., 2020], Two-Echelon [Marques et al., 2020], Time-dependent and multi-trips [Adamo et al., 2021], Multi-Depot [Sadykov et al., 2021], Robust CVRP [Pessoa et al., 2021a], Two-Echelon Stochastic Multi-period [Mohamed et al., 2023], Multi-shuttle Crane Scheduling [Polten and Emde, 2022], Clustered [Freitas et al., 2022], Electric Vehicle [Subramanyam et al., 2022], Split-Delivery [Balster et al., 2023], Cumulative [Damião et al., 2023], Capacitated Location Routing [Liguori et al., 2023], Intermediate Stops [Roboredo et al., 2023], Differential Harvesting [Volte et al., 2023], Simultaneous Pickup and Delivery [Praxedes et al., 2024], Muti-Depot Open Route [Soares and Roboredo, 2024]. A much more user-friendly but with limited functionalities VRPSolver interface is VRPSolverEasy [Errami et al., 2023] (code available at [VRPSolverEasy]).

4.13. Farley's bound. In order to possibly reduce the number of CG iterations in a BPA for the CSP, one may try to prune nodes using Theorem 2.8 statement that $z_{\mathrm{M}} \geq z_{\mathrm{RM}} + U\bar{c}^*$, where U, the number of identical subproblems in the original formulation, in this case is given by the value of the best known integer solution. In the BPA run depicted in Figure 4.9, even if an optimal $U = 7$ was already known from the beginning, that bound would not save any iteration.

For example, after $S_{1.1.4}$, we would have that $z_{\mathrm{M}} \geq 151/24 + 7 \times -1/24 = 6$. However, another bounding mechanism proposed in Farley [1990] could have saved one iteration.

Consider an MLP in the following format:

$$z_{\mathrm{M}} = \min \quad \sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} \tag{4.39a}$$

$$\text{s.t.} \quad \sum_{\boldsymbol{q} \in Q} (\boldsymbol{A}\boldsymbol{q}) \lambda_{\boldsymbol{q}} = \boldsymbol{b} \tag{4.39b}$$

$$\boldsymbol{\lambda} \geq \boldsymbol{0}. \tag{4.39c}$$

The format, which includes the CSP case, is special because all variables have the same positive cost (w.l.o.g. assumed to be 1) and there is no convexity constraint.

Theorem 4.2: *At any iteration of the CGA for solving (4.39), the value* $\frac{z_{RM}}{1-\overline{c}^*}$ *is a lower bound on its optimal cost* z_M.

Proof. The dual of the MLP (4.39) is:

$$\max \quad \boldsymbol{\pi}\boldsymbol{b} \tag{4.40a}$$

$$\text{s.t.} \quad \boldsymbol{\pi}\boldsymbol{A}\boldsymbol{p}_q \leq 1 \qquad \boldsymbol{q} \in Q. \tag{4.40b}$$

Let $\boldsymbol{\pi}^*$ be an optimal dual solution with value $z_{\mathrm{RM}} = \boldsymbol{\pi}^*\boldsymbol{b}$ of some RMLP and let $\overline{c}^* = \min\{1 - \boldsymbol{\pi}^*\boldsymbol{A}\boldsymbol{q} : \boldsymbol{q} \in Q\}$ be the optimal solution value of the corresponding pricing subproblem. For any $\boldsymbol{q} \in Q$, $\boldsymbol{\pi}^*\boldsymbol{A}\boldsymbol{q} \leq 1 - \overline{c}^* \implies \frac{\boldsymbol{\pi}^*}{1-\overline{c}^*}\boldsymbol{A}\boldsymbol{p}_q \leq 1$. Therefore, the dual solution $\frac{\boldsymbol{\pi}^*}{1-\overline{c}^*}$ is feasible for (4.40) and has value $\frac{z_{\mathrm{RM}}}{1-\overline{c}^*}$. □

It is interesting to compare the proof of Theorem 2.8 with the proof of Farley's bound. While the former obtains a dual feasible solution by increasing the dual variable of the convexity constraint, the latter does that by scaling down the π variables.

In the BPA run depicted in Figure 4.9, after $S_{1.1.4}$ we would know that $z_{\mathrm{M}} \geq \frac{151/24}{25/24} = 6.04$. That bound would prove the optimality of an integer solution

with value 7 and (if such a solution was already known) would avoid the last CGA iteration.

4.14. Benders Decomposition for IP. Benders decomposition for LP is 100% equivalent to performing DW decomposition on its dual (see Note 2.10). However, this symmetry breaks down when dealing with IPs, when they become quite distinct techniques.

We begin by outlining the classic variant that appears in Benders [1962], where the variables that remain in the Benders Master are integers and those that go to the subproblems are continuous. Consider a MIP in format (2.39) but with all the n variables x being non-negative integers. A derivation similar to that in Note 2.10 results in the following Master Benders IP:

$$z_{\text{BM}} = z_{\text{IP}} = \min \quad \boldsymbol{c}\boldsymbol{x} + \sum_{u\in[U]} z_u \tag{4.41a}$$

$$\text{s.t.} \quad z_u \geq \boldsymbol{q}^\intercal(\boldsymbol{b}^u - \boldsymbol{A}^u\boldsymbol{x}) \qquad u \in [U],\, \boldsymbol{q} \in Q^u \tag{4.41b}$$

$$\boldsymbol{r}^\intercal(\boldsymbol{b}^u - \boldsymbol{A}^u\boldsymbol{x}) \leq 0 \qquad u \in [U],\, \boldsymbol{r} \in R^u \tag{4.41c}$$

$$\boldsymbol{x} \in \mathbb{Z}^n_+. \tag{4.41d}$$

This is handled through an iterative process where one solves a sequence of Restricted Master Benders IPs. Given an optimal integer solution $(\boldsymbol{x}^*\, \boldsymbol{z}^*)$ to such a RMBIP, for each $u \in [U]$, the subproblem LP (2.44) is solved. An unbounded ray in its dual provides a violated feasibility cut in (4.41c); an optimal dual solution may provide a violated optimality cut in (4.41b). When the process converges, $\boldsymbol{x}^*$ is part of an optimal MIP solution, the optimal values for continuous variables y may be retrieved from the last subproblems. This kind of Benders decomposition can be applied, for example, to network design problems where the relatively few binary variables indicate which links should be built, while the many continuous variables are associated with multi-commodity flows ensuring the desired connectivity properties [Costa, 2005].

The classic Benders decomposition for IP may be time-consuming because each time a RMBIP is solved, even if it only differs from the previous RMBIP by having a handful of additional constraints, a new BB search tree is built. An alternative that become much more popular since the 2000s (referred to

as *Branch-and-Benders-Cut* by some authors) is solving (4.41) by a BCA, keeping a single search tree and using the Benders subproblem LPs to generate cuts either over fractional or integer solutions $(\boldsymbol{x}^* \boldsymbol{z}^*)$. The root node of that BCA uses the Benders decomposition for LP described in Note 2.10 to solve the linear relaxation of the original MIP, which may permit the practical use of some strong but large extended formulations. Actually, as discussed in Botton et al. [2013], it is not mandatory to perform a full separation of Benders cuts for fractional solutions at all nodes of the search tree. Remarkably, automatic Benders decomposition and Branch-and-Benders-Cut were implemented and are currently offered as one of the alternatives for MIP solving in CPLEX [Bonami et al., 2020] and in SCIP [Maher, 2021]. According to IBM CPLEX promoting material: *"Benders decomposition is faster than traditional branch-and-cut for 5% of nontrivial MIP models. That number might not seem impressive but for certain types of MIP problems Benders decomposition is much faster."*

More details on Benders decomposition can be found in surveys like Rahmaniani et al. [2017] or in textbooks like Wolsey [2020]. However, there are two major differences between Benders and DW decomposition worth highlighting:

- The first major difference is conceptual. The CG mechanism requires dual Master solutions and primal subproblem solutions. This means that in standard DW decomposition for IP, the subproblems are solved as IPs. This is what enables DW bounds to be potentially stronger, as formalized in Theorem 4.1. The primal subproblem solutions can be obtained by essentially any technique, including BB-based MIP solvers, Dynamic Programming, etc. In contrast, Benders mechanism requires primal Master solutions and dual subproblem solutions. This means that even if their variables are integer, subproblems are traditionally solved as LPs (or as convex optimization problems where *strong duality exists* [Geoffrion, 1972]). Benders cuts are generated from dual rays and dual optimal solutions, which BB-based MIP solvers or Dynamic Programming can not provide.

  Actually, as surveyed in Fakhri et al. [2017], there are many proposed ideas for performing Benders decomposition with integer subproblems (in order to obtain stronger node bounds in a Branch-and-Benders-Cut

algorithm). For example, if a subproblem IP happens to be infeasible, one may use combinatorial arguments to generate a "feasibility cut" to remove the current Master Benders solution. Yet, those ideas are still under development and may be either less general or less efficient than desired. *Those difficulties are expected considering that general IP does not have strong duality.* A significant number of recent works deal with the special case of integer subproblems happening in the Benders decomposition for two-stage stochastic programming (see Küçükyavuz and Sen [2017]), for example, proposing the so-called Lagrangian Benders cuts, like those in Chen and Luedtke [2022].

- The second major difference is practical: Benders decomposition is a more mature technique than DW decomposition, in the sense that it already found its way into some of the mainstream optimization software. This is definitely not true for DW decomposition. No major commercial MIP solver offers even automatic *basic* BP! Many of the most advanced BCP algorithms of today remain research prototypes.

  The authors of this book have had informal discussions with some of the leading developers of existing commercial MIP solvers. They unanimously expressed that incorporating advanced BCP capabilities into their solvers would not happen soon, due to several technical hurdles. One of them is that the highly complex BCA codes in these solvers were designed to handle the addition of cuts, not columns. This does not mean that implementing Benders on those codes is easy. Far from that. As described in Bonami et al. [2020] and Maher [2021], advanced techniques should be used for dealing with convergence and numerical stability issues, one needs to develop matrix analysis algorithms for finding the right blocks in automatic decomposition, devising a good strategy on when generating Benders cuts can be tricky. However, at least a new Branch-and-Benders-Cut algorithm fits well into existing Branch-and-Cut codes. On the contrary, incorporating BCP into an existing BC code would first require major refactoring.

  A second difficulty that currently keeps advanced BCP away from commercial MIP solvers is that it would also require the additional development of some special-purpose pricing solvers. Indeed, for many of the most spectacular BCP successes, including vehicle routing, pricing by

MIP is simply not efficient. Instead, advanced Dynamic Programming labeling algorithms (Chapter 9) should be used.

4.15. Avoiding Branch-and-Price with Fenchel Cuts. Despite all the progress made in the last decades with the so-called stabilization techniques (Chapter 7), CG is still prone to slow convergence. The issue is compounded by the fact that in many cases the pricing may be slow. In some *successful* BCPAs, like for example, Pessoa et al. [2020], spending a few seconds in each round of pricing and a few minutes in each tree node is not unusual. The approach may still pay if the root node bound is really strong, leading to small search trees. In contrast, the BCAs implemented in modern MIP solvers are usually much faster on reoptimizing nodes and can explore much larger search trees in reasonable times. Even in cases where CG does not present convergence issues and the pricing subproblems are easy (for example, when they correspond to Knapsack Problems that can be solved in a fraction of a millisecond), building a complete BP or BCP algorithm can be a challenging task. Current MIP solvers do not support CG. This means that the BP/BCP may have to be coded by the user. An extreme option is using existing solvers only for solving LPs, with everything else being coded from scratch. Some people do that. Happily, there are some open-source frameworks implementing BCP, like ABACUS, BaPCod, Coluna, DIP, and SCIP GCG. However, those frameworks are still far less developed than the best existing MIP solvers on features like presolving, automatic cut separation, branching selection, search tree management, parallelism, and primal heuristics, among others. This means that in many cases the initial advantage of CG in having a significantly better root node bound will be lost due to a not-so-good BP/BCP implementation.

Those observations lead to the following question. *What if one could use the same pricing algorithm used in a CG to separate cuts and reinforce a linear relaxation, obtaining the same strong bound that the CG would obtain? The reinforced MIP could then be given to a MIP solver who would finish the optimization.* The potential advantages are big. First, each remaining node of the search tree would require a lot less time to be solved. Second, there would be no need for implementing a complete BP/BCP. In fact, one would be profiting from all the advanced features already implemented in the MIP solver.

On the other hand, there would be a drawback. A BP/BCP performs CG in every node, which means that all fractional solutions along the algorithm, even after an arbitrary number of branchings, are guaranteed to remain in the polyhedra defined by the convex hull of the integer solutions of each subproblem. This is not necessarily true in that alternative setting, which may lead to less improvements in the bounds after each branching and larger search trees. Anyway, it is quite possible that the aforementioned advantages more than compensate for that drawback.

Consider an IP having the following format:

$$
\begin{aligned}
z_{\text{IP}} = \min \quad & \boldsymbol{cx} && && \text{(4.42a)}\\
\text{s.t.} \quad & \boldsymbol{Ax} = \boldsymbol{b} && && \text{(4.42b)}\\
& \boldsymbol{x}^u \in P^u && u \in [U] && \text{(4.42c)}\\
& \boldsymbol{x} \in \mathbb{Z}^n, && && \text{(4.42d)}
\end{aligned}
$$

where bounded polyhedron P^u, $u \in [U]$, is defined by a block of linear constraints over $\boldsymbol{x}^u$, a vector containing a subset of the variables in $\boldsymbol{x}$. Assume that relatively few of the n variables appear in each vector $\boldsymbol{x}^u$ (although *those vectors may overlap*, i.e., the same variable may appear in different vectors). Let $Q^u = Int(P^u)$, $u \in [U]$. It is possible to obtain the bound $z_{\text{M}} = \min \boldsymbol{cx}$ s.t. $\boldsymbol{Ax} = \boldsymbol{b}$; $\boldsymbol{x}^u \in Conv(Q^u), u \in [U]$ by separating Fenchel Cuts, as proposed in Boyd [1993, 1994], who coined that name from their relation with Fenchel Duality [3]. Given a fractional solution $\boldsymbol{x}^*$, for each block $u \in [U]$ solve the following Fenchel LP:

$$
\begin{aligned}
z_{\text{F}}^{u*} = \min \quad & \boldsymbol{\alpha x}^{u*} - \beta && && \text{(4.43a)}\\
\text{s.t.} \quad & \boldsymbol{\alpha q} \geq \beta && \boldsymbol{q} \in Q^u && \text{(4.43b)}\\
& (\boldsymbol{\alpha}, \beta) \in S, && && \text{(4.43c)}
\end{aligned}
$$

where $\boldsymbol{x}^{u*}$ is the restriction of $\boldsymbol{x}^*$ to vector x^u and (4.43c) are normalization constraints (in many cases simply setting $\beta = 1$ is a valid normalization). If

[3] Fenchel Cuts can also be viewed as coming from the equivalence between optimization and separation [Grötschel et al., 1981] (see Note 3.14) or even from Farkas Lemma (Note 2.5)

$(\boldsymbol{\alpha}^*, \beta^*)$ is an optimal solution to (4.43) with $z_{\mathrm{F}}^{u*} < 0$ then $\boldsymbol{\alpha}^* \boldsymbol{x}^u \geq \beta^*$ is a valid violated Fenchel Cut separating $\boldsymbol{x}^{u*}$ from $Conv(Q^u)$; otherwise $\boldsymbol{x}^{u*} \in Conv(Q^u)$. Sometimes it is possible to enumerate all points $\boldsymbol{q} \in Q^u$ and solve (4.43) directly. If $|Q|^u$ is small, say up to a few thousand points, this is a highly practical (and often overlooked) way of significantly strengthening a formulation, see Santos et al. [2012] for an example.

However, more frequently, there are far too many points in Q^u for enumeration, so rows (4.43b) should be generated dynamically. Given a solution $(\boldsymbol{\alpha}^*, \beta^*)$ to an LP containing only part of the constraints (4.43b), one solves the "pricing subproblem" $z_{\mathrm{P}}^* = \min \boldsymbol{\alpha}^* \boldsymbol{x}^u - \beta^*$ s.t. $\boldsymbol{x}^u \in Q^u$. If $z_{\mathrm{P}}^* < 0$, then the inequality in (4.43b) corresponding to the found optimal solution should be added. This is repeated until $z_{\mathrm{P}}^* = 0$. At that moment, $(\boldsymbol{\alpha}^*, \beta^*)$ is an optimal solution to the complete LP (4.43).

Fenchel Cuts have the potential for completely replacing BP algorithms. Indeed, one may separate them to achieve the strong bound z_{M} at the root node and then use a generic MIP solver to finish the optimization. Unfortunately, this is seldom practical. The problem is a double-convergence issue: first, one has to solve the pricing many times for generating each single Fenchel Cut; second, several Fenchel Cuts per block may be needed for reaching bound z_{M}.

Avella et al. [2010] is one of the rare examples where this worked fine for a classic problem, namely, the GAP:

- In that case, the blocks correspond to the binary knapsack constraints in the natural GAP formulation (4.16). In the tested instances, each such knapsack constraint involved up to a few hundred variables. The direct solution of the Fenchel LPs (4.43) turned out to be impractical due to slow convergence. The solution was to only consider in a Fenchel LP for block u the fractional variables, i.e., the variables x_j such that $0 < x_j^* < 1$. Constraints (4.43b) should consider the points in Q^u in that restricted space. In their experience, there were never more than 26 fractional variables in a block, which resulted in a much better convergence. However, the obtained Fenchel Cut, which in principle is valid and strong only for the restricted space, should be made valid and strong for the original block u. This can be done by sequential lifting, solving an additional Knapsack Problem for each non-fractional variable.

Even with all the clever tricks employed in Avella et al. [2010], success was only possible because the Knapsack subproblems are extremely well-solved. As far as we know from the literature, Fenchel Cuts was never successful in replacing a BPA in situations where the pricing subproblems are expensive. For further discussions on the topic see Ralphs and Galati [2005], Boccia et al. [2008], Lamothe et al. [2023].

4.16. Avoiding Branch-and-Price with DW Block Cuts. Consider again IP (4.42). The improved bound obtained by convexifying the U blocks can also be obtained by solving its Explicit Master by CG:

$$
\begin{aligned}
z_{\mathrm{M}} = \min \quad & \boldsymbol{c}\boldsymbol{x} && && (4.44a)\\
\text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && && (4.44b)\\
& \boldsymbol{x}^u = \sum_{\boldsymbol{q}\in Q^u} \boldsymbol{q}\,\theta^u_{\boldsymbol{q}} && u \in [U] && (4.44c)\\
& \sum_{\boldsymbol{q}\in Q^u} \theta^u_{\boldsymbol{q}} = 1 && u \in [U] && (4.44d)\\
& \boldsymbol{x} \geq \boldsymbol{0}, \qquad \boldsymbol{\theta} \geq \boldsymbol{0}. && && (4.44e)
\end{aligned}
$$

Let $\boldsymbol{\pi}$, $\boldsymbol{\mu}^u$, $u \in [U]$, and $\boldsymbol{\nu}$ be the vectors of dual variables associated with constraints (4.44b), (4.44c), and (4.44d), respectively. The pricing subproblem for generating θ variables in block u is $\bar{c}^{u*} = \min -\boldsymbol{\mu}^{u*}\boldsymbol{x}^u - \nu_u$ s.t. $\boldsymbol{x}^u \in Q^u$. Chen et al. [2024] proved the following result:

Theorem 4.3: *Let $(\boldsymbol{\pi}^*\, \boldsymbol{\mu}^{1*} \ldots \boldsymbol{\mu}^{U*}\, \boldsymbol{\nu}^*)$ be an optimal dual solution of (4.44). Then, its optimal solution value z_M can be recovered as the value of an optimal solution of the following LP containing valid inequalities (4.45c):*

$$
\begin{aligned}
\min \quad & \boldsymbol{c}\boldsymbol{x} && && (4.45a)\\
\text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && && (4.45b)\\
& -\boldsymbol{\mu}^{u*}\boldsymbol{x}^u \geq \nu^*_u && u \in [U]. && (4.45c)
\end{aligned}
$$

It is quite surprising that a whole block of constraints can be "represented" by a single constraint! Indeed, one can augment IP (4.42) with Constraints (4.45c)

and give it to a MIP solver. This looks like a fantastic way of obtaining much stronger and still compact formulations. However, there are some important caveats:

- Explicit Master LPs are notoriously hard to solve by CG due to slow convergence (caused by having too many dual variables).
- While Constraints (4.45c) indeed raise the linear relaxation bound to z_{M}, they are still a very incomplete description of polyhedra $Conv(Q^u)$, $u \in [U]$. This means that additional cutting or branching will have trouble on further increasing the bounds. In fact, the BCA may remain stuck at bound z_{M} for quite a long time, leading to a "blindness effect" similar to (but not as extreme) the one caused by an objective function cut (Note 3.8)

Nevertheless, Chen et al. [2024] managed to obtain success with these *DW Block Cuts* on the Multiple Knapsack Assignment Problem and the Temporal Knapsack Problem. Those authors circumvent the CG convergence problems by using the Lagrangian method Level [Lemaréchal et al., 1995] to approximately solve (4.44). The applications were chosen because they have a quite large number of overlapping blocks. As a result, up to a few hundred DW Block Cuts can be separated, forming a richer set of constraints that may avoid getting stuck for a long time in solutions with value z_{M}. Moreover, additional techniques for improving the quality of the DW Block Cuts are also provided. Actually, those authors define DW Block Cuts as those in format $\boldsymbol{\alpha}\boldsymbol{x}^u \geq z^*$ obtained by solving $z^* = \min \boldsymbol{\alpha}\boldsymbol{x}^u$ s.t. $\boldsymbol{x}^u \in Q^u$ for some well-chosen (but in principle arbitrary) vector $\boldsymbol{\alpha}$.

We remark that if the original IP has non-overlapping blocks, which is the classical situation where CG is applied, one may separate strong DW Block Cuts using the traditional DW Master instead of the Explicit Master. Consider an IP given by an LP in format (2.11) plus integrality constraints. After aggregating identical subproblems, a DW decomposition yields a Master LP in format (2.15).

Theorem 4.4: *Let $(\boldsymbol{\pi}^*, \boldsymbol{\nu}^*)$ be an optimal dual solution of (2.15). Then, its optimal solution value z_M can be recovered by solving the following LP (over aggregated aggregated original variables, like in (2.19)) containing valid*

inequalities (4.46c)*:*

$$\min \quad \sum_{k\in[k]} \boldsymbol{c}^k \boldsymbol{y}^k \tag{4.46a}$$

$$\text{s.t.} \quad \sum_{k\in[K]} \boldsymbol{A}^k \boldsymbol{y}^k = \boldsymbol{b} \tag{4.46b}$$

$$(\boldsymbol{c}^k - \boldsymbol{\pi}^* \boldsymbol{A}^k)\boldsymbol{y}^k \geq U^k \nu_u^* \qquad k \in [K]. \tag{4.46c}$$

Proof. Notice that $z_{\mathrm{M}} = \min \sum_{k\in[k]} \boldsymbol{c}^k \boldsymbol{y}^k$ s.t. $\sum_{k\in[K]} \boldsymbol{A}^k \boldsymbol{y}^k = \boldsymbol{b}$, $\boldsymbol{y}^k \in U^k \circ Conv(Q^k), k \in [K]$, where $U^k \circ Conv(Q^k)$ is the polyhedron formed by the points that can be obtained as the sum of U^k points in $Conv(Q^k)$. As $0 \leq \overline{c}^{k*} = \min(\boldsymbol{c}^k - \boldsymbol{\pi}^* \boldsymbol{A}^k)\boldsymbol{x}^k - \nu_u^*$ s.t. $\boldsymbol{x}^k \in Q^k$, for every point $\boldsymbol{x}' \in Conv(Q^k)$ we have that $(\boldsymbol{c}^k - \boldsymbol{\pi}^* \boldsymbol{A}^k)\boldsymbol{x}' \geq \nu_u^*$. Therefore, $(\boldsymbol{c}^k - \boldsymbol{\pi}^* \boldsymbol{A}^k)\boldsymbol{y}^k \geq U^k \nu_u^*$ is valid for $U^k \circ Conv(Q^k)$. This already shows that the value of an optimal solution of (4.46) can not be larger than z_{M}. The dual of (4.46) is:

$$\max \quad \boldsymbol{\rho}\, \boldsymbol{b} + \sum_{k\in[U]} \sigma_u U^k \nu_u^* \tag{4.47a}$$

$$\text{s.t.} \quad \boldsymbol{\rho}\, \boldsymbol{A}^k + \sigma_u (c^k - \boldsymbol{\pi}^* \boldsymbol{A}^k) \leq c^k \qquad k \in [U] \tag{4.47b}$$

$$\boldsymbol{\sigma} \geq \boldsymbol{0}. \tag{4.47c}$$

The solution ($\boldsymbol{\rho} = \boldsymbol{\pi}^*$, $\boldsymbol{\sigma} = \boldsymbol{1}$) is dual feasible and thus provides a lower bound of $\boldsymbol{\pi}^* \boldsymbol{b} + \sum_{k\in[U]} U^k \nu_u^* = z_{\mathrm{M}}$ to the value of an optimal solution of (4.46). □

This means that the typical CGA already provides a set of DW Block Cuts that could used to reinforce a formulation and avoid the need for a BPA! Consider the example in Section 4.1. The last pricing subproblem in the solution of the DW reformulation of (4.7) is $\overline{c}^* = \min -85/27x_1 - 85/27x_2 + 255/27$, s.t. $\boldsymbol{x} \in Int(P)$. So, the generated DW Block cut is $-85/27x_1 - 85/27x_2 \geq -255/27$ or $x_1 + x_2 \leq 3$. Adding this cut to (4.7) indeed increases its linear relaxation value to $z_{\mathrm{LP}} = -11.07 = z_{\mathrm{M}}$. In the GAP example of Section 4.4.1, the last pricing subproblems at $S_{1.1}$ (see Figure 4.7) would generate DW Block Cuts $-7x_2^1 - 7x_3^1 \geq -7$ and $-7x_1^2 - 3x_2^2 - 4x_3^2 - 7x_4^2 \geq -14$. In the CVRP, after solving MLP (4.29), a single DW Block Cut (all subproblems U are identical and are aggregated) would be generated. In a similar way, in nearly all

problems where DW decomposition is currently applied, one can perform the standard CG only once, include the resulting DW Block Cuts in the original formulation, and give it a MIP solver, which would start from strong bound z_{M}. There would be no need to build a complete BPA. Is this too good to be true?

We experimented on seven representative GAP instances from the literature (their names indicate the number of machines and jobs), as shown in Table 4.2. We run Gurobi 10.0 MIP solver with a time limit of 10 minutes using a single thread of an Apple M2 processor over its original formulation (4.16) and over the strengthened formulation obtained by adding the K DW Block Cuts obtained from an optimal solution of (4.18). It can be seen that the DW Block Cuts have a disastrous effect on MIP solver performance! For example, on C-20-200 they caused the dual bound to not move during the 10-minute time limit. In contrast, even starting from a lower root bound, the original formulation (4.16) solves the instance in 1 second. As additional information, Gurobi can not solve the last three instances in that table, even in very long runs.

Unfortunately, DW Block Cuts generated from the standard DW Master are unlikely to work well in any problem. To explain that, we need to understand better the differences between an Explicit Master and its corresponding DW Master, which is done in the next note.

4.17. Explicit Master vs Dantzig-Wolfe Master. The Explicit Master (EM), an LP containing both the original and generated variables, was used in Chapter 2 and in this chapter as an intermediate step in deriving the DW decomposition. However, a deeper exploration of the EM may provide conceptual insights. For example, the fact that cuts over the original variables are robust (something that was noticed only in the late 1990s, see Note 4.20) is quite obvious in the EM. As another example, Theorem 4.1, as can be seen in its provided proof, becomes almost self-evident when one considers the EM. However, the EM is also a practical alternative way of applying the CG technique that has some potential advantages:

- The EM is more general and can be used in cases where the subproblems

Table 4.2: Effect of DW Block Cuts on MIP solver performance

Instance	z_{LP}	z_M	z_{IP}	Original formulation without and with DW Block Cuts			
				z_{root}	best bound	nodes	time
C-10-200	2795.41	2803.95	2806	2803.72	opt	6127	1.22
				2803.94	2804.96	969K	600
C-20-200	2376.91	2390.17	2391	2389.62	opt	2291	1.00
				2390.17	2390.17	118K	600
C-20-400	4774.15	4780.18	4782	4779.33	opt	14K	11.0
				4780.18	4780.18	64K	600
D-5-100	6345.41	6349.92	6353	6348.62	opt	115K	17.3
				6349.92	opt	566K	53.2
D-10-100	6323.46	6341.45	6347	6334.52	6343.84	1.21M	600
				6341.45	6341.45	1.19M	600
D-20-100	6142.53	6176.14	6185	6165.57	6175.40	282K	600
				6176.14	6176.14	533K	600
D-25-90-e1	5566.11	5617.96	5627	5606.07	5618.97	168K	600
				5617.96	5617.96	577K	600

Source: original experiment for the book.

have overlapping variables without the need for creating copies of those variables, as discussed in Note 2.9.

- The EM allows the direct fixing of the original variables by reduced costs.

We explain the latter difference between an EM and its corresponding DWM, as observed in Poggi de Aragão and Uchoa [2003] and Longo [2004]. By simplicity, we consider an IP in format (4.1), having a single subproblem, but also assume that $P \subseteq \mathbb{R}^n_+$. The EM is:

$$z_{\mathrm{M}} = \min \quad \boldsymbol{cx} \tag{4.48a}$$

$$\text{s.t.} \quad \boldsymbol{Ax} = \boldsymbol{b} \tag{4.48b}$$

$$\boldsymbol{x} - \sum_{\boldsymbol{q} \in Q} \boldsymbol{q}\lambda_{\boldsymbol{q}} = \boldsymbol{0} \tag{4.48c}$$

$$\sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} = 1 \tag{4.48d}$$

$$\boldsymbol{x}, \quad \boldsymbol{\lambda} \geq \boldsymbol{0}, \tag{4.48e}$$

while the corresponding DWM is (4.5). After solving (4.48) by CG one may use the reduced costs of the x variables for fixing some of them to zero (Note 3.6). Such fixing is robust, not affecting the structure of the pricing in a BPA. On the other hand, using those reduced costs to fix some generated λ variables to zero is not practical: one may indeed fix them in a Restricted Master, but unless the pricing subproblem is changed (non-robust) those fixed variables may be generated again. Anyway, if instead one solves the DWM (4.5) then it is *not* possible to fix x variables directly.

Theorem 4.5: *An optimal dual solution to DWM* (4.5) *yields an optimal dual solution of* (4.48) *where the reduced cost of all original variables x is zero.*

Proof. The dual of (4.48) is:

$$\max \quad \boldsymbol{\pi}\,\boldsymbol{b} \qquad + \nu \tag{4.49a}$$

$$\text{s.t.} \quad \boldsymbol{\pi}\,\boldsymbol{A} + \boldsymbol{\mu} \qquad \leq \boldsymbol{c} \tag{4.49b}$$

$$- \boldsymbol{\mu}\,\boldsymbol{q} + \nu \leq 0 \qquad \boldsymbol{q} \in Q. \tag{4.49c}$$

Let $(\boldsymbol{\pi}^*, \nu^*)$ be an optimal solution to the dual of (4.5). Then $(\boldsymbol{\pi}^*, \boldsymbol{\mu}^* = (\boldsymbol{c} - \boldsymbol{\pi}^*\boldsymbol{A}), \nu^*)$ is an optimal dual solution of (4.49). Note that (4.49c) is satisfied because $(\boldsymbol{c} - \boldsymbol{\pi}^*\boldsymbol{A})\boldsymbol{q} \geq \nu^*$ for all $\boldsymbol{q} \in Q$ and that (4.49b) is satisfied with equality. The last fact implies that for every $j \in [n]$, the reduced cost $\bar{c}_j = c_j - \boldsymbol{\pi}^*\boldsymbol{a}_j + \boldsymbol{\mu}^* = 0$. □

This means that the significant extra effort in solving an EM (the additional dual variables and the existence of many alternative optimal dual solutions lead to convergence issues) may be somehow counterbalanced by gains in terms of fixing. Indeed, it is possible to use perturbations to guide the EM into producing large reduced costs for a chosen subset of the x variables. This is done by introducing slack variables in (4.49b), which is rewritten as $\boldsymbol{\pi}\boldsymbol{A} - \boldsymbol{\mu} + \bar{\boldsymbol{c}} = \boldsymbol{c}$, and including small positive coefficients ϵ_j for those variables in the objective function (4.49a), which assumes the format $\max \boldsymbol{\pi}\boldsymbol{b} + \nu + \boldsymbol{\epsilon}\bar{\boldsymbol{c}}$. This is equivalent to adding constraints $\boldsymbol{x} \geq \boldsymbol{\epsilon}$ to the primal EM (4.48).

While all this is interesting, *we still do not recommend the use of the EM*

instead of the classic DWM in the more classical situations where the subproblems do not overlap. The reason is that, although fixing of the original variables by reduced costs is an essential part of modern advanced BCPAs, the fixing can often be done efficiently on the DWM using the techniques (that require additional calls to variants of the pricing subproblem) described in Chapter 11.

Anyway, now we can understand why DW Block Cuts generated from the DWM (Note 4.16) are unlikely to work well. As can be seen in the proof of Theorem 4.4, LP (4.46) has an optimal dual solution where all DW Block Cuts have dual value 1 and all y variables have reduced cost zero (dual constraints (4.47b) are satisfied with equality). This means that a primal solution of (4.46) is contained in a high-dimensional face formed by alternative optimal solutions with value z_{M}. The overall result is a strong "blindness effect" in a MIP solver, as bad as to the one caused by an objective function cut (Note 3.8).

4.18. Naturally decomposable LCOPs. Section 4.7.2 discussed when it can be better to apply DW reformulation to a certain LCOP original formulation instead of solving it directly. But there is an even deeper question. Which LCOPs themselves are more suitable to CG-based methods? The first thing to consider is whether the LCOP solutions can be decomposed into higher-level objects. For example, vehicle routing solutions are naturally decomposed into routes. Those routes are independent of each other, except for the fact that together they should visit each customer once. CSP solutions are decomposable into uses of cutting patterns, GAP solutions are decomposable into individual machine assignments, and GCP solutions are decomposable into independent sets. These higher-level objects in the solution structure explain the existence of MIP formulations that, after a suitable DW reformulation, lead to multiple independent subproblems, being more favorable to CG. Of course, success also depends on the structure of the resulting subproblems, which should permit efficient pricing.

Many LCOPs are not naturally decomposable. For example, TSP solutions do not seem to admit decomposition into objects larger than individual edges/arcs. However, there are many non-decomposable LCOPs where the removal of part of the constraints leads to a relaxed LCOP with a nice structure, being much

easier than the original LCOP. For example, the Held-Karp 1-tree relaxation for the TSP (Section 5.4.1). In many of those cases, CG is better replaced with Lagrangian Relaxation methods. However, when cutting is necessary to obtain strong bounds, CG may still be recommended (see Section 5.5 for an in-depth discussion). As an example, Abeledo et al. [2013], Roberti and Mingozzi [2014], Bulhões et al. [2018] use CG for the Traveling Deliveryman Problem (a.k.a. Cumulative TSP, Time-dependent TSP or Minimum Latency Problem), a problem that is not decomposable but *relaxable*. The adopted relaxations were q-routes without s-cycles in Abeledo et al. [2013] and dynamic ng-routes in the latter works.

4.19. Early Branch-and-Price algorithms. After Gilmore and Gomory [1961, 1963], several successful CG-based algorithms were soon proposed, not only for cutting and packing but also for other problems. Those algorithms mostly solve the Master LP and use a rounding heuristic to produce integer solutions. Some examples include works by Dzielinski and Gomory [1965] for a lot-sizing problem and by Rao and Zionts [1968] for a transportation units routing problem. Levin [1968] formulates an airline fleet scheduling problem as a multi-commodity flow problem, then reformulates it as a set covering problem and describes a column generation algorithm combined with a hypothetical (i.e., not implemented) branch-and-bound, where branching is performed on arc variables. Simpson [1969] describes a Dantzig-Wolfe decomposition for a multi-period fleet assignment model without discussing ways to obtain integer solutions. Appelgren [1969] treats a ship scheduling problem in which a sequence of cargo transportation requests should be assigned to each ship, and the total profit from optional requests minus the assignment cost is maximized. The solutions obtained by a CGA (the pricing subproblems were solved by DP) were integer on almost all practical instances with up to 40 ships and 50 cargos. In a subsequent paper for the same problem, Appelgren [1971] considers how to optimally solve the relatively few remaining fractional instances. He first implemented a cutting plane algorithm, adding non-robust cuts. Ten out of the 12 tested instances could be solved with a single cut. However, after realizing the potential limitations of the method, an algorithm combining column generation with the BBA was proposed. It was observed

that branching on the MLP variables would also change the structure of the pricing subproblem, making it much harder. Instead, he proposed a robust branching on ship-cargo pairs (equivalent to branching over original binary variables x_{sc} indicating whether ship s takes cargo c). All the tested instances could be optimally solved. Due to the very strong root bounds, the largest search tree only had 11 nodes. To our knowledge, Applegren [1971] is the first Branch-and-Price algorithm.

Yet, that work had little immediate influence and we did not find other full BPAs in the literature until the mid-1980s. Of course, CG was still being used in several applications, including routing and scheduling applications, most importantly for airline optimization, but in a heuristic way. It seems that CG practitioners at that time considered optimal solutions either unnecessary (because the heuristic solutions were good enough) or hopelessly difficult to find. For example, Foster and Ryan [1976] discusses the column generation approach for a vehicle routing problem: *"Unfortunately, it is very slow to converge and rarely gives rise to integer or near integer solutions at the LP optimum. Hence significant computation is required to achieve integrality and optimality. Despite its optimal properties it may therefore only be considered as attractive as a practical method for very small problems."* They then proceed by heuristically restricting the set of feasible routes to only "petal" routes. This restriction allows them to converge much faster and reduce drastically the fractionality of the obtained LP solution. A comprehensive survey on routing and scheduling by Raff [1983] lists some approaches that perform pricing but not a single BPA.

The first BPA for a vehicle routing problem was proposed in Desrosiers et al. [1984] for a VRPTW-like variant without capacity constraints. In that variant, the customers to be serviced by the vehicles are actually trips from fixed origin and destination points. That BPA already used the equivalent of the q-route relaxation and the subproblems were solved by a labeling dynamic programming algorithm. The branching scheme was non-binary (i.e., more than two children nodes could be generated) but robust. The computational results were impressive, solving instances with up to 151 customers. As acknowledged by the authors, those excellent results were possible because the time windows were very narrow (which makes the root lower bounds very

tight) and the number of customers per route was small, usually less than four (which makes convergence quick).

That seminal work spurred the first wave of BPAs for time-constrained routing and scheduling problems, as surveyed in Desrosiers et al. [1995]. The "Montreal Group" created several successful BPAs for problems like Urban Transit Crew Scheduling [Desrochers and Soumis, 1989], Pickup and Delivery with TW [Dumas et al., 1991], VRPTW [Desrochers et al., 1992], Multiple-Depot Vehicle Scheduling [Ribeiro and Soumis, 1994], Door-to-Door Handicapped Transportation [Ioachim et al., 1995]. Many of those BPAs were implemented over the GENCOL framework, which could handle Master problems having Set-Partitioning constraints and subproblems cast as resource-constrained shortest-path problems.

Other influential early BPAs include: CVRP [Agarwal et al., 1989], Min-Cut Clustering [Johnson et al., 1993], Bandwidth Packing [Parker and Ryan, 1993], BPP [Vance et al., 1994], Graph Coloring [Mehrotra and Trick, 1996], GAP [Savelsbergh, 1997], and CSP [Vance, 1998].

4.20. Early Branch-Cut-and-Price algorithms. The first BCPA was proposed in Nemhauser and Park [1991] for the edge coloring problem. The columns correspond to matchings in the graph and, at the beginning of the algorithm, can be priced in polynomial time. However, after non-robust Odd-Cycle cuts are added, the pricing subproblem has to be solved by a general MIP solver.

As can be seen in the comprehensive survey and exposition on BPAs by Barnhart et al. [1998], in the mid-1990s it was not yet clear which kind of cuts could be used in a BCPA without the undesirable effect of changing the structure of the pricing subproblems. Then, several researchers independently created the first robust BCPAs (Lot Sizing [Vanderbeck, 1998], Package Delivery [Kim et al., 1999], VRPTW [Kohl et al., 1999], Machine Scheduling [van den Akker et al., 2000], Supply Chain Management [Felici et al., 2000], and Integer Multicommodity Flow [Barnhart et al., 2000]) by realizing that the dual variables of cuts defined over the variables from an original formulation only affected the subproblems costs, not their structure.

The classification of BCPAs as being *robust* or *non-robust* was proposed in

Poggi de Aragão and Uchoa [2003]. That nomenclature was intended to capture a distinction that was observed by researchers at that time: even when a pricing algorithm could be adapted for handling non-robust cuts, its performance often became very *unstable*. That work also championed robust BCPAs: *"Perhaps an important contribution is simply to point out the vast potential impact that BCP algorithms represent in the practice of integer programming. Up to now, BCP was only applied to (a few) problems where a pure BP already performed well. We argue that the applicability of this technique goes far beyond that class of problems"*. The proof of the concept was the robust BCPA for the CVRP in Fukasawa et al. [2006]. BPAs were already known to be good for the VRPTW with narrow time windows, but they performed poorly on the CVRP (which can be seen as a VRPTW with wide time windows). BCAs were then viewed as the clearly best approach for CVRP (see Section 3.4.2). Table 4.3, taken from Fukasawa et al. [2006], shows the average gaps over a set of 41 instances having from 49 to 134 customers by the following relaxations: the master LP (4.29) solved by a CG pricing q-routes without 2-cycles, the linear relaxation of Edge Formulation (3.6) including the separation of all cuts in Lysgaard et al. [2004], and their synergetic robust cut-and-price combination, for q-routes without s-cycles, $s \in \{2, 3, 4\}$. The much improved cut-and-price bounds (remember that BBA trees grow exponentially with the gap) allowed all the 18 open instances in that set to be solved.

Table 4.3: CG alone vs Cut Separation alone vs their combination

	CG $s = 2$	Edge + Cuts	CG $s = 2$ + Cuts	CG $s = 3$ + Cuts	CG $s = 4$ + Cuts
Gap	4.76%	2.26%	0.97%	0.82%	0.74%

Source: Fukasawa et al. [2006].

The enthusiasm for robust BCPAs decreased in the following years. It was soon realized that further progress would depend on finding new effective families of robust cuts, which is not an easy task. Note that even families of cuts that are facet-defining may be useless in a robust BCPA if they are already implied by the CG (i.e., by the partial convexification induced by the DW decomposition). For example, Letchford and Salazar-González [2006] proved that

CVRP Generalized Multistar Cuts are already implied by q-routes. In that context, robust BCPAs that exploited cuts defined over pseudo-polynomial extended formulations were proposed: Capacitated Minimum Spanning Tree [Uchoa et al., 2008] and Parallel Machine Scheduling [Pessoa et al., 2010]. This is a potentially powerful approach. The large number of variables in those extended formulations makes it easier to find new families of robust cuts. Yet, the bulky original formulation is essentially hidden by the CG approach.

Meanwhile, other works [Jepsen et al., 2008, Baldacci et al., 2008, 2011] have shown the decisive importance of non-robust cuts for obtaining really small gaps on some important problems and proposed techniques for mitigating their non-robustness. This led in the 2010s to the modern advanced BCPAs that use both robust and "controlled non-robust" cutting, which will be covered in detail in Part Two of this book. Other techniques that are characteristic of those advanced BCPAs include dual stabilization, numerically safe dual bounds, efficient fixing by Lagrangian reduced costs, active fixing, the possibility of going back to the reduced original formulation, column enumeration to pools, integrated primal heuristics, MLP management, advanced branch rules, hierarchical strong branching, and non-robustness control by rollback. Special care is given to designing the pricing algorithms. Indeed, cutting and pricing may achieve a degree of symbiosis, in the sense that the non-robust cuts are already *defined* with the pricing algorithm in mind, to minimize their negative impact on its performance.

4.21. Branch-Cut-and-Price or Branch-Price-and-Cut? The name Branch-*and*-Cut-and-Price, often found in the late 1990s and early 2000s literature, was disrecommended by native speakers [Letchford, 2005] as bad English. However, some authors prefer the name Branch-Price-and-Cut because, as explained in Desrosiers and Lübbecke [2011], it better reflects the algorithm: first columns are priced, and only after that, cuts start to be separated. By that reasoning, Price-Cut-and-Branch would be an even better name! Anyway, in this book we still adopt the more historical name Branch-Cut-and-Price.

Exercises

E 4.1. Consider the IP shown in Exercise E 3.1 (page 109). Perform a DW reformulation that keeps the first constraint in the master (leading to two independent subproblems) and solve the resulting IP by a robust BPA with branching over the original variables (the resulting constraints should be kept in the master). Choose the branching variable by the most fractional rule. Show the BPA tree in detail, like in Figure 4.7.

E 4.2. Solve the same IP using a BPA with branching over the original variables but including the resulting constraints in the subproblems (see Note 4.3).

E 4.3. Solve the same IP using a robust BCPA, separating the following valid inequalities to cut the fractional solutions: $x_2 + x_3 - x_4 \leq -1$ and $-x_2 + x_3 + x_4 \leq 5$.

E 4.4. Solve the same IP using a non-robust BPA that performs branching over the generated variables. Show the required modifications in the pricing subproblems.

E 4.5. Solve the same IP using a non-robust BCPA that separates CG cuts at the root node. Show the required modifications in the pricing subproblems. Suggestion: use multipliers $\boldsymbol{\rho} = (0.1\ 0.6\ 0.09)$ for the constraint kept in the master and for the first and second convexity constraints, respectively. Hint: when in trouble finding CG multipliers for moderate-sized problems, try the MIP in Fischetti and Lodi [2007].

E 4.6. Write the general MLP that is obtained by a DW reformulation of a MIP in format (4.8) that keeps (4.8b) in the master, assuming that P is bounded.

E 4.7. Solve the following GAP instance by a robust BPA that performs branching over the original variables:

		Cost (c_j^k)			Load (w_j^k)			W_k
Jobs		**1**	**2**	**3**	**1**	**2**	**3**	
Machines	**1**	1	8	7	7	3	4	9
	2	6	2	5	3	7	4	8

E 4.8. Consider the following problem:

Definition 4.14: Capacitated Facility Location Problem (CFLP). Instance: J customers and K locations; integer customer demands w_j, $j \in [J]$; opening costs g_k and integer capacities W_k, $k \in [K]$; and assignment costs c_j^k, $j \in [J]$, $k \in [K]$. Solutions: Set $K' \subseteq K$ of open locations and assignments of customers to locations in K' such that the total demand in each location does not exceed its capacity. Goal: minimize opening costs plus assignment costs.

The natural formulation for CFLP uses binary variables y_k to indicate whether location k is opened and binary variables x_j^k to indicate whether customer j is assigned to location k:

$$z_{\text{IP}} = \min \sum_{k\in[K]} \sum_{j\in[J]} c_j^k x_j^k + \sum_{k\in[K]} g_k y_k \tag{4.50a}$$

$$\text{s.t.} \quad \sum_{k\in[K]} x_j^k = 1 \qquad j \in [J] \tag{4.50b}$$

$$\sum_{j\in[J]} w_j x_j^k \le W_k y_k \qquad k \in [K] \tag{4.50c}$$

$$(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{B}^{JK+K}. \tag{4.50d}$$

Perform a DW reformulation on it that keeps (4.50b) in the master. Write the resulting MLP and the pricing subproblems.

E 4.9. Solve the linear relaxation of the KGG formulation for the following CSP

instance: $W = 132$, $\boldsymbol{w} = (\,44\ 33\ 12\,)$, and $\boldsymbol{d} = (\,2\ 3\ 6\,)$. Solve it again, but this time use a bounded KP in the pricing, forbidding non-proper cutting patterns, i.e., those where the number of copies of some item is larger than its demand. Find a proven optimal integer solution.

E 4.10. In the CSP with Multiple Stock Sizes there are K different stock lengths, stock $k \in [K]$ having integer length W_k and unitary cost g_k. The problem was already formulated both in Kantorovich [1939] and in Gilmore and Gomory [1961] in terms of cutting pattern variables. Provide an original formulation for that problem similar to (4.23). Perform a DW reformulation of it and derive, step by step, the KGG formulation.

E 4.11. Generalize Valério de Carvalho's pseudo-polynomial flow formulation for the CSP with multiple stock sizes. Hint: start from Formulation (4.33) for the maximum stock length and only add $K - 1$ arcs.

E 4.12. The Bin Packing with Conflicts is a BPP variant that includes a conflict graph $G = ([J], E)$. If $(i, j) \in E$, items i and j can not be packed in the same bin. The problem can be viewed as a mix between the classic BPP and the Graph Coloring Problem. In fact, if $W = \infty$ it becomes a GCP; if $E = \emptyset$ it becomes the BPP. Propose an original formulation for the problem akin to formulations (4.23) and (4.31). Then perform a DW reformulation over it to obtain an IP suitable for CG.

E 4.13. Consider a MLP in the following format:

$$z_{\mathrm{M}} \;=\; \min \quad \sum_{k\in[K]} \sum_{\boldsymbol{q}\in Q^k} g_k \lambda^k_{\boldsymbol{q}} \tag{4.51a}$$

$$\text{s.t.} \sum_{k\in[K]} \sum_{\boldsymbol{q}\in Q^k} (\boldsymbol{A}^k \boldsymbol{q}) \lambda^k_{\boldsymbol{q}} = \boldsymbol{b} \tag{4.51b}$$

$$\boldsymbol{\lambda} \geq \boldsymbol{0}. \tag{4.51c}$$

The format includes the CSP with multiple stock sizes, g_k being the cost

of stock size $k \in [K]$. Generalize Farley's bound (Note 4.13) for that case. In other words, given a RMLP with optimal value z_{RM} and optimal subproblem values $\overline{c}^{k*}$, $k \in [K]$, obtain a formula giving a proven lower bound on z_{M}.

E 4.14. Consider a MLP in format (4.39) but also having the convexity constraint $\sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} = U$. Generalize Farley's bound (Note 4.13) for that case. Hint: the optimal dual variable value ν^* of the convexity constraint in the RMLP should be used in the formula.

E 4.15. Project Exercise. Implement (perhaps using a CG framework) a general robust BPA for the GAP that performs branching over the original variables. Include in your tests instances with many machines, like those in Nauss [2003]. Compare your results with those from the literature and with the direct solution of original Formulation (4.16) by a MIP solver.

E 4.16. Project Exercise. Implement a general robust BPA for the CFLP that performs branching over the original variables. Include in your tests instances having many locations with small capacities. Compare your results with those from the literature and with solving Formulation (4.50) with a MIP solver.

E 4.17. Project Exercise. Implement a general robust BPA for the Graph Coloring Problem that performs RFB branching, as in Mehrotra and Trick [1996]. Use the ISP/MWCP codes mentioned in Note 4.7 for the pricing. Compare your results with those from the literature and with the direct solution of the original Formulation (3.4) by a MIP solver.

E 4.18. Project Exercise. Implement a BPA for the Rectangular Partition problem described in Exercise E 3.4. You should write the code for solving the pricing subproblem and define a suitable branching scheme. Perform tests on two sets of instances: the first with $L = W = 20$ and the second

with $L = W = 50$. In both cases, the points can be randomly selected. Compare your implementation with solving the full SPP formulation with a MIP solver.

Chapter 5

Lagrangian Relaxation and Column Generation

We present the link between Column Generation and a well-known and very general optimization technique: Lagrangian Relaxation (LR). Applications of LR range from elementary calculus to sophisticated algorithms for solving complex constrained non-linear problems. Dantzig-Wolfe decomposition and CG can be derived from scratch as an application of LR to LPs/IPs. To follow a philosophy adopted in this book, that readers may be able to choose their preferred point of view, we should present that alternative to the CG derivation given in the Chapters 2 and 4. In addition, the resulting Lagrangian Dual Problems have a graphical representation that brings insight into how the CGA works. The LR link is particularly valuable for understanding the stabilization and advanced variable fixing techniques that will be presented in Chapters 7 and 11, respectively. Moreover, we also provide guidelines on when Lagrangian methods are likely to be a good alternative for *replacing* CG.

5.1. General Lagrangian Relaxation

LR was originally developed at the beginning of the 19th century to solve constrained variational calculus problems [Lagrange, 1806]. However, the surprisingly simple ideas that it uses allow it to be presented in this section without compromising the focus of this book.

5.1.1. Basic Results

LR is suitable for optimization problems that become much easier when a certain subset of its constraints is relaxed. Instead of simply ignoring those constraints, the LR method adds to the objective function their violations multiplied by adjustable factors known as Lagrangian multipliers. This is done in the hope that those penalization terms may guide the optimal solution to a region that is feasible for the original problem. In reality, the problem with relaxed constraints *and modified objective function* should be much easier than the original problem. To formalize this notion, let $X \subseteq \mathbb{R}^n$ be a (possibly infinite) non-empty set of points, and let $c : X \mapsto \mathbb{R}$, $\boldsymbol{g} : X \mapsto \mathbb{R}^m$, and $\boldsymbol{h} : X \mapsto \mathbb{R}^r$ be functions defined over such points. The general problem considered here is the following:

$$\begin{aligned} \min \quad & z = c(\boldsymbol{x}) && \text{(5.1a)} \\ \text{s.t.} \quad & \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0} && \text{(5.1b)} \\ & \boldsymbol{h}(\boldsymbol{x}) \leq \boldsymbol{0} && \text{(5.1c)} \\ & \boldsymbol{x} \in X. && \text{(5.1d)} \end{aligned}$$

The LR is applied by keeping (5.1d), while Constraints (5.1b) and (5.1c) are *dualized*, i.e., they are moved to the objective function as penalization terms. For any fixed *Lagrangian multipliers* $\boldsymbol{\pi} \in \mathbb{R}^{1\times m}$ and $\boldsymbol{\rho} \in \mathbb{R}^{1\times r}_{+}$, let $L(\boldsymbol{x}, \boldsymbol{\pi}, \boldsymbol{\rho}) = c(\boldsymbol{x}) + \boldsymbol{\pi}\,\boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{\rho}\,\boldsymbol{h}(\boldsymbol{x})$ and define the Lagrangian Subproblem (LS) as:

$$\begin{aligned} \min \quad & L(\boldsymbol{x}, \boldsymbol{\pi}, \boldsymbol{\rho}) && \text{(5.2a)} \\ \text{s.t.} \quad & \boldsymbol{x} \in X. && \text{(5.2b)} \end{aligned}$$

The resolution of LSs for fixed multipliers defines a function that maps the possible Lagrangian multipliers into the corresponding optimal value of (5.2). The *Lagrangian Dual Function* (LDF) $L(\boldsymbol{\pi}, \boldsymbol{\rho}) = \min_{\boldsymbol{x}\in X} L(\boldsymbol{x}, \boldsymbol{\pi}, \boldsymbol{\rho})$ has as its proper domain the possible multipliers where this minimum is not unbounded. To avoid technicalities, we keep the convention used throughout this book and define $L(\boldsymbol{\pi}, \boldsymbol{\rho})$ for all possible multipliers, saying that $L(\boldsymbol{\pi}, \boldsymbol{\rho}) = -\infty$ if the minimum is unbounded.

Theorem 5.1: *The LDF $L(\boldsymbol{\pi}, \boldsymbol{\rho})$ is concave over its proper domain.*

Proof. For each fixed $\boldsymbol{x} \in X$, $L(\boldsymbol{x}, \boldsymbol{\pi}, \boldsymbol{\rho}) = c(\boldsymbol{x}) + \boldsymbol{\pi}\,\boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{\rho}\,\boldsymbol{h}(\boldsymbol{x})$ is a linear function of $\boldsymbol{\pi}$ and $\boldsymbol{\rho}$. Hence, the finite value $L(\boldsymbol{\pi}, \boldsymbol{\rho}) = \min_{\boldsymbol{x} \in X} L(\boldsymbol{x}, \boldsymbol{\pi}, \boldsymbol{\rho})$ is the pointwise minimum of a number (possibly infinite) of linear functions. Thus, it is concave (but not necessarily strictly concave). □

An LDF $L(\boldsymbol{\pi}, \boldsymbol{\rho})$ is *piecewise linear* if it can be defined over its proper domain as the pointwise minimum of a finite number of linear functions $L(\boldsymbol{x}, \boldsymbol{\pi}, \boldsymbol{\rho})$, $\boldsymbol{x} \in Y$, for some finite $Y \subseteq X$. As will be seen, applications of LR to linear or integer programming lead to piecewise linear LDFs, whereas non-linear programming applications typically do not. The *hypograph of a function* is the set of points lying on or below its graph, over its proper domain, see the examples depicted in Figures 5.2 and 5.3a. In the latter example, the LDF is piecewise linear and its hypograph is a polyhedron.

Theorem 5.2: Weak duality. *Let $\boldsymbol{x}'$ be a feasible solution of (5.1). For any $\boldsymbol{\pi} \in \mathbb{R}^{1\times m}$, $\boldsymbol{\rho} \in \mathbb{R}^{1\times r}_{+}$, $L(\boldsymbol{\pi}, \boldsymbol{\rho}) \leq c(\boldsymbol{x}')$.*

Proof. The proof can be written in one line:

$$L(\boldsymbol{\pi}, \boldsymbol{\rho}) = \min_{\boldsymbol{x} \in X} L(\boldsymbol{x}, \boldsymbol{\pi}, \boldsymbol{\rho}) \leq L(\boldsymbol{x}', \boldsymbol{\pi}, \boldsymbol{\rho}) = c(\boldsymbol{x}') + \boldsymbol{\pi}\,\boldsymbol{g}(\boldsymbol{x}') + \boldsymbol{\rho}\,\boldsymbol{h}(\boldsymbol{x}') \leq c(\boldsymbol{x}').$$

The first and the second equalities hold by the definitions of $L(\boldsymbol{\pi}, \boldsymbol{\rho})$ and $L(\boldsymbol{x}, \boldsymbol{\pi}, \boldsymbol{\rho})$, respectively. The first inequality is true because $\boldsymbol{x}'$ satisfies (5.1d). Since $\boldsymbol{\rho} \geq \boldsymbol{0}$ and $\boldsymbol{x}'$ also satisfies both (5.1b) and (5.1c), second inequality also holds. □

The fundamental Theorem 5.2 shows that if (5.1) has an optimal solution $\boldsymbol{x}^*$ with value $c(\boldsymbol{x}^*) = z^*$, then LR can be used to provide lower bounds on z^*. Since these lower bounds are parameterized by the vectors $\boldsymbol{\pi}$ and $\boldsymbol{\rho}$, it is natural to ask for the best possible lower bound that can be obtained in this way. For that, the *Lagrangian Dual Problem* (LDP) is defined as follows:

$$z_{\text{LD}} = \max\ L(\boldsymbol{\pi}, \boldsymbol{\rho}) \tag{5.3a}$$

$$\text{s.t.} \quad \boldsymbol{\pi} \in \mathbb{R}^{1\times m},\ \boldsymbol{\rho} \in \mathbb{R}^{1\times r}_{+}. \tag{5.3b}$$

In general, LDPs do not provide strong duality. Indeed, there are many cases where $z_{\text{LD}} < z^*$. The following result indicates a situation where strong duality holds.

Theorem 5.3: *Given* $\boldsymbol{\pi} \in \mathbb{R}^{1\times m}$, $\boldsymbol{\rho} \in \mathbb{R}^{1\times r}_{+}$, *and* $\bar{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}\in X} L(\boldsymbol{x}, \boldsymbol{\pi}, \boldsymbol{\rho})$, *if* $\boldsymbol{g}(\bar{\boldsymbol{x}}) = \mathbf{0}$, $\boldsymbol{h}(\bar{\boldsymbol{x}}) \leq \mathbf{0}$, *and* $\boldsymbol{\rho h}(\bar{\boldsymbol{x}}) = 0$ *then* $c(\bar{\boldsymbol{x}}) = L(\boldsymbol{\pi}, \boldsymbol{\rho}) = z_{LD} = z^*$.

Proof. $z^* \leq c(\bar{\boldsymbol{x}}) = c(\bar{\boldsymbol{x}}) + \boldsymbol{\pi}\, \boldsymbol{g}(\bar{\boldsymbol{x}}) + \boldsymbol{\rho}\, \boldsymbol{h}(\bar{\boldsymbol{x}}) = L(\boldsymbol{\pi}, \boldsymbol{\rho}) \leq z_{\text{LD}} \leq z^*$. The first inequality is valid because the conditions of the theorem impose that $\bar{\boldsymbol{x}}$ is feasible for the original problem (5.1). The first and the second equalities are true because $\boldsymbol{\pi g}(\bar{\boldsymbol{x}}) = \boldsymbol{\rho h}(\bar{\boldsymbol{x}}) = 0$ and by the definition of $L(\boldsymbol{\pi}, \boldsymbol{\rho})$, respectively. The second inequality holds by definition of z_{LD}. The last inequality follows from Theorem 5.2. □

This means that a solution to an LS that is feasible to the original problem is also optimal if the "complementary slackness" (see Theorem 1.4) conditions $\boldsymbol{\rho}\, \boldsymbol{h}(\bar{\boldsymbol{x}}) = 0$ are satisfied.

To illustrate the concepts presented so far, consider the following problem with a non-linear objective function:

$$\min z = \frac{1}{x_1} + \frac{1}{x_2} \tag{5.4a}$$

$$\text{s.t.} \quad x_1 + 4x_2 \leq 8 \tag{5.4b}$$

$$x_1, \quad x_2 > 0. \tag{5.4c}$$

Dualizing Constraint (5.4b), for any fixed $\rho \in \mathbb{R}_+$ we have the following Lagrangian Subproblem[1]:

$$\min \quad L(x_1, x_2, \rho) = \frac{1}{x_1} + \frac{1}{x_2} + \rho(x_1 + 4x_2 - 8) \tag{5.5a}$$

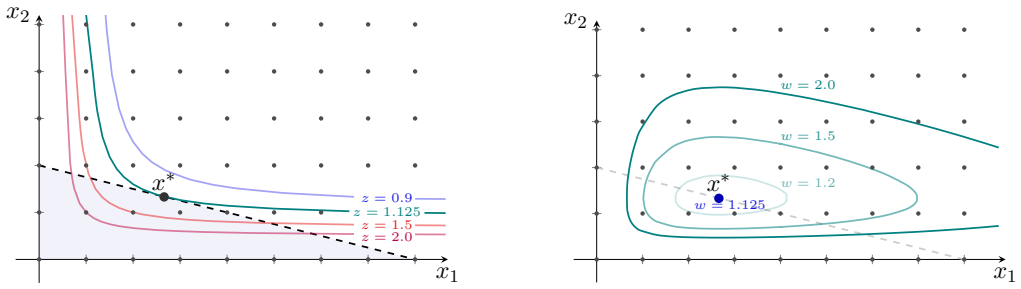
$$\text{s.t.} \quad x_1, x_2 > 0, \tag{5.5b}$$

This LS can be decomposed into two independent problems: $P_1 \equiv \min_{x_1>0}\{1/x_1 + \rho x_1\}$, and $P_2 \equiv \min_{x_2>0}\{1/x_2 + 4\rho x_2\}$. In this case, elementary calculus can be used to solve both P_1 and P_2 analytically as a function of $\rho > 0$. Since both functions are strictly convex for fixed ρ, their minimum values are attained in the only point where the derivative with respect to x_1 or x_2 is zero. Let $\bar{x}_1 = 1/\sqrt{\rho}$ and $\bar{x}_2 =$

[1] Technically, we should have used "inf" instead of "min" in (5.5a) because the problem has no optimal value when $\rho = 0$.

$1/(2\sqrt{\rho})$ be the optimal solutions for P_1 and P_2, respectively. Then, we obtain that $L(\rho) = L(\bar{x}_1, \bar{x}_2, \rho) = 6\sqrt{\rho} - 8\rho$ defines the Lagrangian Dual Function. Since that function is strictly concave, its maximum value is attained at the only point $\rho^* = 9/64 = 0.1406$ where the derivative (now with respect to ρ) is zero. Hence, the best lower bound on the optimal solution of (5.4) that the Lagrangian approach can provide is $z_{\text{LD}} = L(\rho^*) = 9/8 = 1.125$.

In this particular example, LR may do more than provide a valid lower bound. Solving the LS with $\rho = \rho^*$ leads to a solution $\bar{\boldsymbol{x}} = (\bar{x}_1 \;\; \bar{x}_2) = (8/3 \;\; 4/3) = (2.666 \; 1.333)$ that turns out to be feasible for the original problem ($\bar{x}_1 + 4\bar{x}_2 = 8$) and also satisfies the other conditions in Theorem 5.3. Therefore, $c(\bar{\boldsymbol{x}}) = L(\rho^*)$ and $\boldsymbol{x}^* = \bar{\boldsymbol{x}}$ is optimal for (5.4). Figure 5.1a displays contour lines of the original objective function (5.4a), showing that the green contour line corresponding to $z = 1.125$ only intersects the feasible region (depicted in gray) at the point $\boldsymbol{x}^*$. If Constraint (5.4b) is simply relaxed, the resulting problem has no optimal solution because z approaches zero as both x_1 and x_2 go to infinity. However, if that constraint is incorporated into the objective function with penalization $\rho^* = 9/64$, the resulting LS $L(x_1, x_2, \rho^*)$ has as its unique minimum solution the point $\boldsymbol{x}^*$ (some of its contour lines are depicted in Figure 5.1b).



(a) Contour lines of $z = 1/x_1 + 1/x_2$ and the feasible region of the original problem.

(b) Contour lines of $w = L(x_1, x_2, \rho^*)$.

Figure 5.1: The effect of dualizing the constraint $x_1 + 4x_2 \leq 8$ in (5.4) with multiplier $\rho^* = 9/64$

Figure 5.2 depicts $L(\rho) = 6\sqrt{\rho} - 8\rho$, which, according to the proof of Theorem 5.1, is the pointwise minimum of an infinite number of linear functions, one for each point $(x_1, x_2) > 0$. The graph displays the lines corresponding to four selected points: $\boldsymbol{x}^1 = (4, 2)$, $\boldsymbol{x}^2 = (8/3, 4/3)$, $\boldsymbol{x}^3 = (1.4, 0.7)$, and $\boldsymbol{x}^4 = (1, 2)$. For example,

the point $\boldsymbol{x}^1$ leads to the line $L(\boldsymbol{x}^1, \rho) = 3/4 + 4\rho$, which intersects $L(\rho)$ at (1/16, 1). This means that $\boldsymbol{x}^1$ is an optimal solution of (5.5) for $\rho = 1/16$. Point $\boldsymbol{x}^2$ corresponds to the line $L(\boldsymbol{x}^2, \rho) = 9/8$, which intersects $L(\rho)$ at its maximum. The line $L(\boldsymbol{x}^3, \rho) = 15/7 - 3.8\rho$ also intersects $L(\rho)$. On the other hand, the line $L(\boldsymbol{x}^4, \rho) = 3/2 + \rho$ does not touch $L(\rho)$ (so, $\boldsymbol{x}^4$ is never an optimal solution of (5.5)) and is redundant in the definition of this LDF.

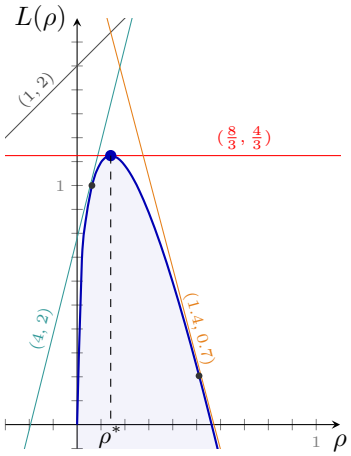


Figure 5.2: LDF $L(\rho) = 6\sqrt{\rho} - 8\rho$ and the lines $L(x_1, x_2, \rho)$ associated with some fixed points $(x_1, x_2) > 0$. The hypograph of $L(\rho)$ is shaded in blue.

5.1.2. Deriving standard LP duality with LR

It is instructive to apply LR to general LP. Consider an LP in the following format:

$$\begin{aligned} \min \quad & \boldsymbol{c}\boldsymbol{x} && \text{(5.6a)} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} && \text{(5.6b)} \\ & \boldsymbol{x} \geq \boldsymbol{0}. && \text{(5.6c)} \end{aligned}$$

By dualizing (5.6b), which is equivalent to $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{0}$, the LS for any fixed $\boldsymbol{\rho} \geq \boldsymbol{0}$ is:

$$\min \quad L(\boldsymbol{x}, \boldsymbol{\rho}) = \boldsymbol{c}\boldsymbol{x} + \boldsymbol{\rho}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) = (\boldsymbol{c} - \boldsymbol{\rho}\boldsymbol{A})\boldsymbol{x} + \boldsymbol{\rho}\boldsymbol{b} \tag{5.7a}$$

$$\text{s.t.} \quad \boldsymbol{x} \geq \boldsymbol{0}. \tag{5.7b}$$

If the vector $\boldsymbol{c} - \boldsymbol{\rho}\boldsymbol{A}$ has a negative component, say the i-th one, then (5.7) is unbounded because x_i can go to infinity. So, the proper domain of the LDF $L(\boldsymbol{\rho})$ only contains points $\boldsymbol{\rho}$ such that $\boldsymbol{c} - \boldsymbol{\rho}\boldsymbol{A} \geq \boldsymbol{0}$. Thus, the optimal solution value of the corresponding LDP does not change if $\boldsymbol{c} - \boldsymbol{\rho}\boldsymbol{A} \geq \boldsymbol{0}$ is included as a constraint in it. Note that $\boldsymbol{x}^* = \boldsymbol{0}$ is always an optimal solution of (5.7) when $\boldsymbol{c} - \boldsymbol{\rho}\boldsymbol{A} \geq \boldsymbol{0}$, which implies that $L(\boldsymbol{\rho}) = L(\boldsymbol{x}^*, \boldsymbol{\rho}) = \boldsymbol{\rho}\boldsymbol{b}$. So, the resulting LDP becomes:

$$\max \quad \boldsymbol{\rho}\,\boldsymbol{b} \tag{5.8a}$$

$$\text{s.t.} \quad \boldsymbol{\rho}\boldsymbol{A} \leq \boldsymbol{c} \tag{5.8b}$$

$$\boldsymbol{\rho} \geq \boldsymbol{0}, \tag{5.8c}$$

which is precisely the LP dual of (5.6). As a result, Theorem 5.2 proves LP weak duality. Yet, as far as we know, no one could prove LP strong duality using only basic LR. The standard proofs of Theorem 2.12 use more advanced results like Farkas' Lemma or the correctness of the Simplex algorithm.

5.2. DW Reformulation for IP as LR

The DW reformulation for IP can be derived using LR. We start with the simplest case where there is a single subproblem, and then, follow the same steps for multiple subproblems partitioned into groups of identical problems. The resulting LDPs have a graphical representation that brings valuable insights into how the CGA works. We generalize Theorems 2.6 and 2.8, opening the way for hybrid CG approaches where the pricing subproblems are sometimes solved with dual values that are not necessarily those provided by RMLPs.

5.2.1. Single Subproblem

Consider the following IP:

$$
\begin{aligned}
z_{\text{IP}} = \min \quad & \boldsymbol{cx} && \text{(5.9a)}\\
\text{s.t.} \quad & \boldsymbol{Ax} = \boldsymbol{b} && \text{(5.9b)}\\
& \boldsymbol{Dx} \geq \boldsymbol{d} && \text{(5.9c)}\\
& \boldsymbol{x} \in P && \text{(5.9d)}\\
& \boldsymbol{x} \in \mathbb{Z}^n, && \text{(5.9e)}
\end{aligned}
$$

where $\boldsymbol{A}$ and $\boldsymbol{D}$ have dimensions $m \times n$ and $r \times n$, respectively; the other vectors have compatible dimensions. Moreover, P is a bounded polyhedron defined by a set of linear constraints. By adding surplus variables to Constraints (5.9c), they could be converted to equalities and incorporated into (5.9b), resulting in a formulation in the form of (4.1). Here, however, we follow what is more frequent in the LR literature and keep equality and inequality constraints separated. Dualizing Constraints (5.9b) and (5.9c), we obtain from (5.9) the following LS for any fixed $\boldsymbol{\pi} \in \mathbb{R}^{1\times m}$, and $\boldsymbol{\rho} \in \mathbb{R}^{1\times r}_{+}$:

$$
\begin{aligned}
\min \quad & L(\boldsymbol{x}, \boldsymbol{\pi}, \boldsymbol{\rho}) = \boldsymbol{cx} + \boldsymbol{\pi}(\boldsymbol{b} - \boldsymbol{Ax}) + \boldsymbol{\rho}(\boldsymbol{d} - \boldsymbol{Dx}) && \text{(5.10a)}\\
\text{s.t.} \quad & \boldsymbol{x} \in P && \text{(5.10b)}\\
& \boldsymbol{x} \in \mathbb{Z}^n. && \text{(5.10c)}
\end{aligned}
$$

Let $Q = Int(P)$. We can rewrite (5.10) as $\min_{\boldsymbol{q}\in Q}\{\boldsymbol{cq} + \boldsymbol{\pi}(\boldsymbol{b} - \boldsymbol{Aq}) + \boldsymbol{\rho}(\boldsymbol{d} - \boldsymbol{Dq})\}$. Then, the LDP becomes:

$$
\begin{aligned}
z_{\text{LD}} = \max \quad & L(\boldsymbol{\pi}, \boldsymbol{\rho}) = \min_{\boldsymbol{q}\in Q}\{\boldsymbol{cq} + \boldsymbol{\pi}(\boldsymbol{b} - \boldsymbol{Aq}) + \boldsymbol{\rho}(\boldsymbol{d} - \boldsymbol{Dq})\} && \text{(5.11a)}\\
\text{s.t.} \quad & \boldsymbol{\pi} \in \mathbb{R}^{1\times m} && \text{(5.11b)}\\
& \boldsymbol{\rho} \in \mathbb{R}^{1\times r}_{+}. && \text{(5.11c)}
\end{aligned}
$$

Introducing an auxiliary variable z to represent the value of $L(\boldsymbol{\pi}, \boldsymbol{\rho})$, the LDP can be rewritten as:

$$
\begin{aligned}
z_{\text{LD}} = \max \quad & z && && (5.12a)\\
\text{s.t.} \quad & z \leq \boldsymbol{cq} + \boldsymbol{\pi}(\boldsymbol{b} - \boldsymbol{Aq}) + \boldsymbol{\rho}(\boldsymbol{d} - \boldsymbol{Dq}) && \boldsymbol{q} \in Q && (5.12b)\\
& z \in \mathbb{R} && && (5.12c)\\
& \boldsymbol{\pi} \in \mathbb{R}^{1\times m} && && (5.12d)\\
& \boldsymbol{\rho} \in \mathbb{R}_{+}^{1\times r}, && && (5.12e)
\end{aligned}
$$

As we show later in this section, this form is suitable for a graphical visualization of $L(\boldsymbol{\pi}, \boldsymbol{\rho})$. However, to establish the link between LR and CG, we use here a slightly modified form. Let $\nu \in \mathbb{R}$ be a new variable defined as:

$$
\nu = z - \boldsymbol{\pi b} - \boldsymbol{\rho d}. \tag{5.13}
$$

We use it to eliminate the z variable from (5.12) by substituting it with $\nu + \boldsymbol{\pi b} + \boldsymbol{\rho d}$. Then, (5.12) becomes:

$$
\begin{aligned}
z_{\text{LD}} = \max \quad & \nu + \boldsymbol{\pi b} + \boldsymbol{\rho d} && && (5.14a)\\
\text{s.t.} \quad & \nu \leq \boldsymbol{cq} - \boldsymbol{\pi Aq} - \boldsymbol{\rho Dq} && \boldsymbol{q} \in Q && (5.14b)\\
& \nu \in \mathbb{R} && && (5.14c)\\
& \boldsymbol{\pi} \in \mathbb{R}^{1\times m} && && (5.14d)\\
& \boldsymbol{\rho} \in \mathbb{R}_{+}^{1\times r}. && && (5.14e)
\end{aligned}
$$

Note that the last variable change removes the influence of the terms $\boldsymbol{\pi b}$ and $\boldsymbol{\rho d}$ over Constraints (5.12b). Let $\boldsymbol{\lambda}$ be the vector of dual variables associated with

Constraints (5.14b). The dual LP of (5.14) is:

$$z_{\mathrm{M}} = \min \quad \sum_{\boldsymbol{q}\in Q} (\boldsymbol{cq})\lambda_{\boldsymbol{q}} \tag{5.15a}$$

$$\text{s.t.} \quad \sum_{\boldsymbol{q}\in Q} (\boldsymbol{Aq})\lambda_q = \boldsymbol{b} \tag{5.15b}$$

$$\sum_{\boldsymbol{q}\in Q} (\boldsymbol{Dq})\lambda_q \geq \boldsymbol{d} \tag{5.15c}$$

$$\sum_{\boldsymbol{q}\in Q} \lambda_{\boldsymbol{q}} = 1 \tag{5.15d}$$

$$\lambda \geq \mathbf{0}, \tag{5.15e}$$

which is precisely the MLP obtained from the DW decomposition of (5.9) that keeps (5.9b) in the master. We have just proved that $z_{\mathrm{LD}} = z_{\mathrm{M}} \leq z_{\mathrm{IP}}$. Moreover, it can be deduced from Theorem 4.1 that z_{LD} can only be larger than z_{LP}, the bound obtained by the IP linear relaxation, if P does not have the integrality property. Actually, that result on the strength of z_{LD} was originally proved for LR applied to IP [Geoffrion, 1974]. Only later it was realized that it also applies to CG [Magnanti et al., 1976].

Consider the Restricted Master LP obtained from (5.15) by only having the subset of the λ variables corresponding to some $S \subseteq Q$:

$$z_{\mathrm{RM}} = \min \quad \sum_{\boldsymbol{q}\in S} (\boldsymbol{cq})\lambda_{\boldsymbol{q}} \tag{5.16a}$$

$$\text{s.t.} \quad \sum_{\boldsymbol{q}\in S} (\boldsymbol{Aq})\lambda_{\boldsymbol{q}} = \boldsymbol{b} \tag{5.16b}$$

$$\sum_{\boldsymbol{q}\in S} (\boldsymbol{Dq})\lambda_{\boldsymbol{q}} \geq \boldsymbol{d} \tag{5.16c}$$

$$\sum_{\boldsymbol{q}\in S} \lambda_{\boldsymbol{q}} = 1 \tag{5.16d}$$

$$\boldsymbol{\lambda} \geq \mathbf{0}, \tag{5.16e}$$

Let $(\boldsymbol{\pi}^*, \boldsymbol{\rho}^*, \nu^*)$ be an optimal dual solution for it. The corresponding pricing sub-

problem is given by:

$$\overline{c}^* = \min \quad (\boldsymbol{c} - \boldsymbol{\pi}^* \boldsymbol{A} - \boldsymbol{\rho}^* \boldsymbol{D})\boldsymbol{x} - \nu^* \tag{5.17a}$$

$$\text{s.t.} \quad \boldsymbol{x} \in P. \tag{5.17b}$$

Theorem 2.6 states that $z_{\text{RM}} + \overline{c}^*$ is a valid lower bound on z_{M}. The next result shows that such a bound can be viewed as a Lagrangian bound:

Theorem 5.4: *If $(\boldsymbol{\pi}^*, \boldsymbol{\rho}^*, \nu^*)$ is an optimal dual solution for (5.16), then $z_{RM} + \overline{c}^* = L(\boldsymbol{\pi}^*, \boldsymbol{\rho}^*)$.*

Proof. The pricing subproblem (5.17) and the LS (5.10) are the same except for a constant term in the objective function, which is $-\nu^*$ for the former and $\boldsymbol{\pi}^*\boldsymbol{b} + \boldsymbol{\rho}^*\boldsymbol{d}$ for the latter. Let $\hat{\boldsymbol{x}}$ be an optimal solution to both subproblems. Then,

$$\begin{aligned} L(\boldsymbol{\pi}^*, \boldsymbol{\rho}^*) = L(\hat{\boldsymbol{x}}, \boldsymbol{\pi}^*, \boldsymbol{\rho}^*) &= \boldsymbol{c}\hat{\boldsymbol{x}} + \boldsymbol{\pi}^*(\boldsymbol{b} - \boldsymbol{A}\hat{\boldsymbol{x}}) + \boldsymbol{\rho}^*(\boldsymbol{d} - \boldsymbol{D}\hat{\boldsymbol{x}}) \\ &= (\boldsymbol{c} - \boldsymbol{\pi}^*\boldsymbol{A} - \boldsymbol{\rho}^*\boldsymbol{D})\hat{\boldsymbol{x}} + \boldsymbol{\pi}^*\boldsymbol{b} + \boldsymbol{\rho}^*\boldsymbol{d} \\ &= (\boldsymbol{c} - \boldsymbol{\pi}^*\boldsymbol{A} - \boldsymbol{\rho}^*\boldsymbol{D})\hat{\boldsymbol{x}} + z_{\text{RM}} - \nu^* \;=\; z_{\text{RM}} + \overline{c}^*, \end{aligned}$$

where the last but one equality holds because, by strong LP duality, $z_{\text{RM}} = \boldsymbol{\pi}^*\boldsymbol{b} + \boldsymbol{\rho}^*\boldsymbol{d} + \nu^*$. □

We give an example that will graphically illustrate the interpretation of the CGA as a method for optimally solving an LDP. Consider the following IP:

$$\begin{array}{rlrcrcl} z_{\text{IP}} = & \min & x_1 & + & 3\,x_2 & & \\ & \text{s.t.} & -x_1 & + & x_2 & \geq & -1 \\ & & x_1 & + & 6\,x_2 & \leq & 12 \\ & & 2\,x_1 & + & 4\,x_2 & \geq & 5 \\ & & 0 & \leq & x_1 & \leq & 3 \\ & & & & \boldsymbol{x} & \in & \mathbb{Z}^2. \end{array} \tag{5.18}$$

Its linear relaxation has value $z_{\text{LP}} = 3$, while $z_{\text{IP}} = 4$. By dualizing its first con-

straint, the obtained LS for fixed $\rho \geq 0$ is:

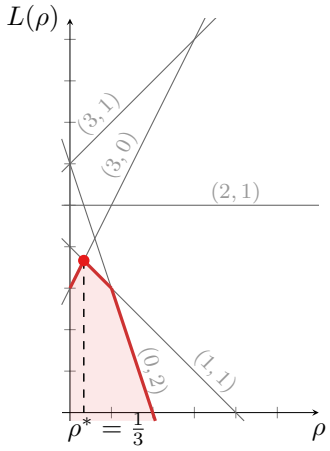
$$\begin{aligned}
\min \quad & L(x_1,x_2,\rho) = x_1 + 3x_2 + \rho(-1 + x_1 - x_2) && \text{(5.19a)}\\
\text{s.t.} \quad & x_1 + 6x_2 \leq 12 && \text{(5.19b)}\\
& 2x_1 + 4x_2 \geq 5 && \text{(5.19c)}\\
& 0 \leq x_1 \leq 3 && \text{(5.19d)}\\
& \boldsymbol{x} \in \mathbb{Z}^2. && \text{(5.19e)}
\end{aligned}$$

Its set of solutions is $Q = \{(0,2),(1,1),(2,1),(3,0),(3,1)\}$. So, the LDP (in format (5.12)) becomes:

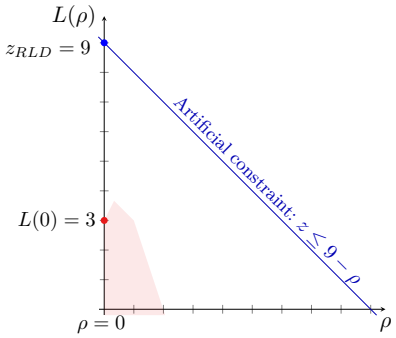
$$\begin{aligned}
z_{\text{LD}} = \max \quad & z && \text{(5.20a)}\\
\text{s.t.} \quad & z \leq 6 - 3\rho && \text{(5.20b)}\\
& z \leq 4 - \rho && \text{(5.20c)}\\
& z \leq 5 && \text{(5.20d)}\\
& z \leq 3 + 2\rho && \text{(5.20e)}\\
& z \leq 6 + \rho && \text{(5.20f)}\\
& \rho \geq 0. && \text{(5.20g)}
\end{aligned}$$

The graph of the LDF $L(\rho)$ is depicted in Figure 5.3a. It can be seen that for $0 \leq \rho < 1/3$ the unique optimal solution of (5.19) is $(3,0)$; for $\rho = 1/3$ the optimal solutions are $(3,0)$, and $(1,1)$; for $1/3 < \rho < 1$ the optimal solution is $(1,1)$; and so on. The lines corresponding to the points $(2,1)$ and $(3,1)$ do not touch $L(\rho)$, therefore, those points are never optimal solutions of (5.19). The value $z_{\text{LD}} = 11/3 = 3.66$ is achieved with $\rho^* = 1/3$.

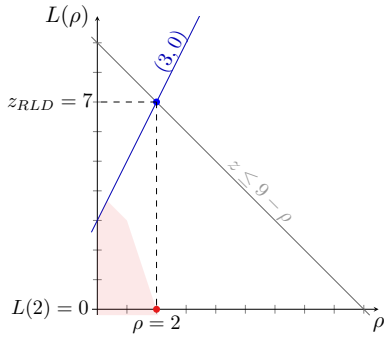
We will now perform a DW reformulation of (5.18), keeping its first constraint in the master, and then apply the CGA. We need an artificial variable a_1 for building the first feasible RMLP. That particular artificial variable is equivalent to adding the point $(0,0)$ to Q, but with a "very large" cost (to keep the visualization neater,



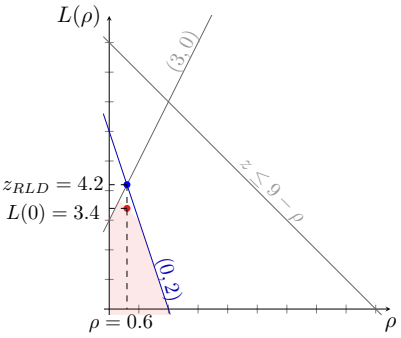
(a) The LDF $L(\rho)$ and its maximum ($\rho^* = 1/3, z_{\text{LD}} = 11/3$). Its polyhedral hypograph is shaded in red



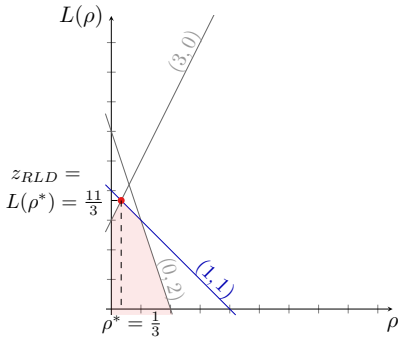
(b) RLDP with $S = \emptyset$ and artificial constraint, maximum ($\rho = 0$, $z_{\text{RLD}} = 9$)



(c) RLDP with $S = \{(3, 0)\}$, maximum ($\rho = 2, z_{\text{RLD}} = 7$)



(d) RLDP with $S = \{(3, 0), (0, 2)\}$, maximum ($\rho = 0.6, z_{\text{RLD}} = 4.2$)



(e) RLDP with $S = \{(3, 0), (0, 2), (1, 1)\}$ and optimal LDP solution ($\rho^* = \frac{1}{3}$, $z_{\text{LD}} = \frac{11}{3}$)

Figure 5.3: Lagrangian Dual Problem and Restricted Lagrangian Dual Problems

we will use 9 instead of the usual 99).

$$\begin{array}{rlrcr} z_{\text{RM}} = & \min & 9\,a_1 & & \\ & \text{s.t.} & 0\,a_1 & \geq & -1 \\ & & a_1 & = & 1 \\ & & a_1 & \geq & 0. \end{array}$$

This RMLP can be solved by only looking at the graph of the corresponding Restricted Lagrangian Dual Problem (RLDP), which is $z_{\text{RLD}} = \max z$ s.t. $z \leq 9 - \rho$, $\rho \geq 0$, shown in Figure 5.3b. Its maximum point is $(\rho = 0, z_{\text{RLD}} = 9)$. So, an optimal value for the dual variable of the RMLP first constraint is $\rho = 0$ and $z_{\text{RM}} = z_{\text{RLD}} = 9$. *In general, artificial variables in RMLPs correspond to artificial dual constraints that prevent the RLDPs from being unbounded.* Anyway, the value of the dual variable of the convexity constraint is readily obtained from (5.13), so $\nu = z_{\text{RM}} + \rho = 9$. Now, solving the LS with $\rho = 0$, one obtains the lower bound $L(0) = 3$ (also visible in Figure 5.3b). Moreover, the optimal solution $\hat{\boldsymbol{x}} = (3, 0)$ and leads to a column with reduced cost $\overline{c}^* = L(0) - z_{\text{RM}} = -6$ and the next RMLP is:

$$\begin{array}{rlrcrcrcr} z_{\text{RM}} = & \min & 9\,a_1 & + & 3\,\lambda_1 & & & & \\ & \text{s.t.} & 0\,a_1 & - & 3\,\lambda_1 & \geq & -1 \\ & & a_1 & + & \lambda_1 & = & 1 \\ & & a_1, & & \lambda_1 & \geq & 0. \end{array}$$

Looking at the new RLDP $z_{\text{RLD}} = \max z$ s.t. $z \leq 9 - \rho$, $z \leq 3 + 2\rho$, $\rho \geq 0$ shown in Figure 5.3c, its maximum point is $(\rho = 2, z_{\text{RLD}} = z_{\text{RM}} = 7)$, so $\nu = z_{\text{RM}} + \rho = 9$. Solving the LS with $\rho = 2$, one obtains the lower bound $L(2) = 0$ (visible in Figure 5.3c). The LS optimal $\hat{\boldsymbol{x}} = (0, 2)$ leads to a column with reduced cost $\overline{c}^* = L(2) - z_{\text{RM}} = -7$ and the next RMLP is:

$$\begin{array}{rlrcrcrcr} z_{\text{RM}} = & \min & 9\,a_1 & + & 3\,\lambda_1 & + & 6\,\lambda_2 & & \\ & \text{s.t.} & 0\,a_1 & - & 3\,\lambda_1 & + & 2\,\lambda_2 & \geq & -1 \\ & & a_1 & + & \lambda_1 & + & \lambda_2 & = & 1 \\ & & a_1, & & \lambda_1, & & \lambda_2 & \geq & 0. \end{array}$$

The new RLDP shown in Figure 5.3d has its maximum at $(\rho = 0.6, z_{\text{RLD}} = z_{\text{RM}} = 4.2)$, so $\nu = z_{\text{RM}} + \rho = 4.8$. Solving the LS with $\rho = 0.6$, the optimal point $(1, 1)$ leads to $L(0.6) = 3.4$ (visible in Figure 5.3d). So, $\overline{c}^* = L(0.6) - z_{\text{RM}} = -0.8$. The

next RMLP (after removing the now unnecessary artificial variable) is:

$$\begin{array}{rlrcrcrcr} z_{\mathrm{RM}} = & \min & 3\,\lambda_1 & + & 6\,\lambda_2 & + & 4\,\lambda_3 & & \\ & \text{s.t.} & -3\,\lambda_1 & + & 2\,\lambda_2 & & & \geq & -1 \\ & & \lambda_1 & + & \lambda_2 & + & \lambda_3 & = & 1 \\ & & \lambda_1, & & \lambda_2, & & \lambda_3 & \geq & 0. \end{array}$$

The RLDP in Figure 5.3e has its maximum at $(\rho = 1/3, z_{\mathrm{RLD}} = z_{\mathrm{RM}} = 11/3)$, so $\nu = z_{\mathrm{RM}} + \rho = 4$. Solving the LS with $\rho = 1/3$, one obtains that $L(1/3) = 11/3$. As $\overline{c}^* = L(1/3) - z_{\mathrm{RM}} = 0$, the last RMLP is optimal and $z_{\mathrm{M}} = z_{\mathrm{RM}} = z_{\mathrm{RLD}} = z_{\mathrm{LD}}$.

The results in this section lead to valuable insights:

- The LP (5.15) obtained by a DW reformulation of an IP is equivalent to the maximization of $L(\boldsymbol{\pi}, \boldsymbol{\rho})$, which is a concave function defined by up to $|Q|$ linear parts, subject only to the non-negativity of $\boldsymbol{\rho}$. As $|Q|$ may be huge, it is usually not possible to know $L(\boldsymbol{\pi}, \boldsymbol{\rho})$ explicitly. However, the pricing subproblem acts as an oracle that given a fixed $(\hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\rho}})$ not only returns the correct value of $L(\hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\rho}})$, but also returns a vector $\hat{\boldsymbol{x}} \in Q$ that defines $L(\hat{\boldsymbol{x}}, \boldsymbol{\pi}, \boldsymbol{\rho})$, a linear part of $L(\boldsymbol{\pi}, \boldsymbol{\rho})$ that touches it at $(\hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\rho}})$.

- The CGA can be viewed as a particular method for maximizing $L(\boldsymbol{\pi}, \boldsymbol{\rho})$, where the tentative $(\hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\rho}})$ values are obtained by solving RMLPs (equivalent to RLDPs) defined by the linear parts of $L(\boldsymbol{\pi}, \boldsymbol{\rho})$ given by the points already evaluated. As the CGA progresses, the partial description of $L(\boldsymbol{\pi}, \boldsymbol{\rho})$ contained in the RMLPs improves, until it converges to a complete local description of it around an optimal $(\boldsymbol{\pi}^*, \boldsymbol{\rho}^*)$, a point such that $L(\boldsymbol{\pi}^*, \boldsymbol{\rho}^*) = z_{\mathrm{LD}} = z_{\mathrm{M}}$.

The realization that the CGA can be viewed as a particular method for solving a Lagrangian Dual Problem indicates that it can be generalized and enhanced by combining it with elements borrowed from LR. In particular, since it is possible to solve the pricing subproblems with arbitrary dual values (as long as they are valid Lagrangian multipliers) and still obtain valid lower bounds, to improve the convergence of the CGA, one may use auxiliary techniques for guessing good dual values, which are those that lead to bounds close to $z_{\mathrm{LD}} = z_{\mathrm{RM}}$. Those so-called "stabilization" techniques for CG will be described in Chapter 7.

5.2.2. Multiple Subproblems

Consider an IP in the following format:

$$
\begin{aligned}
z_{\text{IP}} = \min \quad & \boldsymbol{c}^1\boldsymbol{x}^1 + \cdots + \boldsymbol{c}^U\boldsymbol{x}^U && && \text{(5.21a)}\\
\text{s.t.} \quad & \boldsymbol{A}^1\boldsymbol{x}^1 + \cdots + \boldsymbol{A}^U\boldsymbol{x}^U = \boldsymbol{b} && && \text{(5.21b)}\\
& \boldsymbol{D}^1\boldsymbol{x}^1 + \cdots + \boldsymbol{D}^U\boldsymbol{x}^U \geq \boldsymbol{d} && && \text{(5.21c)}\\
& \boldsymbol{x}^u \in P^u && u \in [U] && \text{(5.21d)}\\
& \boldsymbol{x}^u \in \mathbb{Z}^n && u \in [U]. && \text{(5.21e)}
\end{aligned}
$$

For each $u \in [U]$, $\boldsymbol{A}^u$ and $\boldsymbol{D}^u$ have dimensions $m \times n^u$ and $r \times n^u$, and P^u is a bounded polyhedron. The special structure in this LP is that (5.21d) consists of U independent sets of constraints, each such set being expressed over a distinct subset of the variables.

As done in Section 2.3.2, we also assume that $[U]$ can be partitioned into K maximal groups of identical subproblems. Group $k \in [K]$ has U^k subproblems. The first K subproblems are distinct and the remaining $U-K$ subproblems are identical to some of those first K subproblems. Let $U(k) \subseteq [U]$ be the indices associated with group $k \in [K]$. Dualizing Constraints (5.21b) and (5.21c), we obtain the following LS for any fixed $\boldsymbol{\pi} \in \mathbb{R}^{1\times m}$, and $\boldsymbol{\rho} \in \mathbb{R}_+^{1\times r}$:

$$
\begin{aligned}
\min \quad L(\boldsymbol{x}, \boldsymbol{\pi}, \boldsymbol{\rho}) &= \sum_{u\in[U]} \boldsymbol{c}^u\boldsymbol{x}^u + \boldsymbol{\pi}\left(\boldsymbol{b} - \sum_{u\in[U]} \boldsymbol{A}^u\boldsymbol{x}^u\right) + \boldsymbol{\rho}\left(\boldsymbol{d} - \sum_{u\in[U]} \boldsymbol{D}^u\boldsymbol{x}^u\right)\\
&= \sum_{u\in[U]} (\boldsymbol{c}^u - \boldsymbol{\pi}\boldsymbol{A}^u - \boldsymbol{\rho}\boldsymbol{D}^u)\boldsymbol{x}^u + \boldsymbol{\pi}\boldsymbol{b} + \boldsymbol{\rho}\boldsymbol{d}\\
&= \sum_{k\in[K]} \sum_{u\in U(k)} (\boldsymbol{c}^k - \boldsymbol{\pi}\boldsymbol{A}^k - \boldsymbol{\rho}\boldsymbol{D}^k)\boldsymbol{x}^u + \boldsymbol{\pi}\boldsymbol{b} + \boldsymbol{\rho}\boldsymbol{d} && \text{(5.22a)}\\
\text{s.t.} \quad \boldsymbol{x}^u &\in P^u && u \in [U] && \text{(5.22b)}\\
\boldsymbol{x}^u &\in \mathbb{Z}^{n^u} && u \in [U]. && \text{(5.22c)}
\end{aligned}
$$

By considering (5.22) as defining a single subproblem, all results in Section 5.2.1 remain valid. In particular, by defining $\boldsymbol{c} = (\boldsymbol{c}^1 \dots \boldsymbol{c}^U)$, $\boldsymbol{A} = (\boldsymbol{A}^1 \dots \boldsymbol{A}^U)$, $\boldsymbol{D} = (\boldsymbol{D}^1 \dots \boldsymbol{D}^U)$ and $Q = \{(\boldsymbol{q}^1 \dots \boldsymbol{q}^U) \,|\, \boldsymbol{q}^u \in Int(P^u), u \in [U]\}$ (the solutions in Q are obtained by the possible concatenations of subproblem solutions), we have an LDP in format (5.12). As shown later in this section, this form is still suitable for a

graphical visualization of $L(\boldsymbol{\pi}, \boldsymbol{\rho})$. However, it is interesting to exploit the fact that (5.22) can be decomposed into K independent and distinct subproblems. For each $k \in [K]$, the subproblem to be solved is:

$$\xi^{k*} = \min \quad (\boldsymbol{c}^k - \boldsymbol{\pi}\boldsymbol{A}^k - \boldsymbol{\rho}\boldsymbol{D}^k)\boldsymbol{x}^k \tag{5.23a}$$

$$\text{s.t.} \quad \boldsymbol{x}^k \in P^k \tag{5.23b}$$

$$\boldsymbol{x}^k \in \mathbb{Z}^{n^k}, \tag{5.23c}$$

and $L(\boldsymbol{\pi}, \boldsymbol{\rho}) = \sum_{k\in[K]} U^k \xi^{k*} + \boldsymbol{\pi}\boldsymbol{b} + \boldsymbol{\rho}\boldsymbol{d}$. Defining $Q^k = Int(P^k)$, for each $k \in [K]$, we can rewrite the $L(\boldsymbol{\pi}, \boldsymbol{\rho})$ as $\sum_{k\in[K]} U^k \min_{\boldsymbol{q}\in Q^k}\{(\boldsymbol{c}^k - \boldsymbol{\pi}\boldsymbol{A}^k - \boldsymbol{\rho}\boldsymbol{D}^k)\boldsymbol{q}\} + \boldsymbol{\pi}\boldsymbol{b} + \boldsymbol{\rho}\boldsymbol{d}$, and the LDP becomes:

$$z_{\text{LD}} = \max L(\boldsymbol{\pi}, \boldsymbol{\rho}) = \sum_{k\in[K]} U^k \min_{\boldsymbol{q}\in Q^k}\{(\boldsymbol{c}^k - \boldsymbol{\pi}\boldsymbol{A}^k - \boldsymbol{\rho}\boldsymbol{D}^k)\boldsymbol{q}\} + \boldsymbol{\pi}\boldsymbol{b} + \boldsymbol{\rho}\boldsymbol{d} \tag{5.24a}$$

$$\text{s.t.} \quad \boldsymbol{\pi} \in \mathbb{R}^{1\times m} \tag{5.24b}$$

$$\boldsymbol{\rho} \in \mathbb{R}^{1\times r}_{+}. \tag{5.24c}$$

Defining ν_k, $k \in [K]$, as auxiliary variables, (5.24) can be rewritten as:

$$z_{\text{LD}} = \max \quad \sum_{k\in[K]} U^k \nu_k + \boldsymbol{\pi}\boldsymbol{b} + \boldsymbol{\rho}\boldsymbol{d} \tag{5.25a}$$

$$\text{s.t.} \quad \nu_k \le (\boldsymbol{c}^k - \boldsymbol{\pi}\boldsymbol{A}^k - \boldsymbol{\rho}\boldsymbol{D}^k)\boldsymbol{q}, \qquad k \in [K],\ \boldsymbol{q} \in Q^k \tag{5.25b}$$

$$\boldsymbol{\nu} \in \mathbb{R}^K \tag{5.25c}$$

$$\boldsymbol{\pi} \in \mathbb{R}^{1\times m} \tag{5.25d}$$

$$\boldsymbol{\rho} \in \mathbb{R}^{1\times r}_{+}. \tag{5.25e}$$

For each $k \in [K]$ and $\boldsymbol{q} \in Q^k$, let $\lambda^k_{\boldsymbol{q}}$ be the dual variable associated with each

constraint in (5.25b). The dual LP of (5.25) is:

$$z_{\mathrm{M}} = \min \quad \sum_{k\in[K]} \sum_{\boldsymbol{q}\in Q^k} (\boldsymbol{c}^k \boldsymbol{q})\lambda^k_{\boldsymbol{q}} \tag{5.26a}$$

$$\text{s.t.} \quad \sum_{k\in[K]} \sum_{\boldsymbol{q}\in Q^k} (\boldsymbol{A}^k \boldsymbol{q})\lambda^k_{\boldsymbol{q}} = \boldsymbol{b} \tag{5.26b}$$

$$\sum_{k\in[K]} \sum_{\boldsymbol{q}\in Q^k} (\boldsymbol{D}^k \boldsymbol{q})\lambda^k_{\boldsymbol{q}} \geq \boldsymbol{d} \tag{5.26c}$$

$$\sum_{\boldsymbol{q}\in Q^k} \lambda^k_{\boldsymbol{q}} = U^k \qquad k \in [K] \tag{5.26d}$$

$$\boldsymbol{\lambda}^k \geq \boldsymbol{0} \qquad k \in [K], \tag{5.26e}$$

which is precisely the MLP obtained from the DW decomposition of (5.21), adapted to take advantage of identical subproblems. We can now prove that the lower bounds provided by Theorem 2.8 can be viewed as Lagrangian bounds.

Theorem 5.5: *Consider a restricted master of (5.26), i.e., its restriction to the variables defined over subsets $S^k \subseteq Q^k$, $k \in [K]$. Let $(\boldsymbol{\pi}^*, \boldsymbol{\rho}^*, \boldsymbol{\nu}^*)$ be an optimal dual solution for it with cost z_{RM}. For each $k \in [K]$, the optimal pricing subproblem solution value is $\overline{c}^{k*} = \min(\boldsymbol{c}^k - \boldsymbol{\pi}^* \boldsymbol{A}^k - \boldsymbol{\rho}^* \boldsymbol{D}^k)\boldsymbol{x}^k - \nu^*_k$ s.t. $\boldsymbol{x}^k \in P^k$. Then, $z_{RM} + \sum_{k\in[K]} U^k \overline{c}^{k*} = L(\boldsymbol{\pi}^*, \boldsymbol{\rho}^*)$.*

Proof. Disregarding the constant terms, the pricing subproblems are equivalent to the problems in (5.23). Let $\hat{\boldsymbol{x}}^k$ be an optimal solution, for each $k \in [K]$, and $\hat{\boldsymbol{x}}$ be the concatenation of those subproblem solutions into a single LS solution. Then, we have

$$\begin{aligned}
L(\boldsymbol{\pi}^*, \boldsymbol{\rho}^*) &= L(\hat{\boldsymbol{x}}, \boldsymbol{\pi}^*, \boldsymbol{\rho}^*) \\
&= \sum_{k\in[K]} U^k (\boldsymbol{c}^k - \boldsymbol{\pi}^* \boldsymbol{A}^k - \boldsymbol{\rho}^* \boldsymbol{D}^k)\hat{\boldsymbol{x}}^k + \boldsymbol{\pi}^* \boldsymbol{b} + \boldsymbol{\rho}^* \boldsymbol{d} \\
&= \sum_{k\in[K]} U^k (\boldsymbol{c}^k - \boldsymbol{\pi}^* \boldsymbol{A}^k - \boldsymbol{\rho}^* \boldsymbol{D}^k)\hat{\boldsymbol{x}}^k + z_{\mathrm{RM}} - \sum_{k\in[K]} U^k \nu^*_k \\
&= z_{\mathrm{RM}} + \sum_{k\in[K]} U^k ((\boldsymbol{c}^k - \boldsymbol{\pi}^* \boldsymbol{A}^k - \boldsymbol{\rho}^* \boldsymbol{D}^k)\hat{\boldsymbol{x}}^k - \nu^*_k) \\
&= z_{\mathrm{RM}} + \sum_{k\in[K]} U^k \overline{c}^*_k,
\end{aligned}$$

where the third equality holds because, by strong LP duality, $z_{\mathrm{RM}} = \boldsymbol{\pi}^* \boldsymbol{b} + \boldsymbol{\rho}^* \boldsymbol{d} + \sum_{k \in [K]} U^k \nu_k^*$. □

We give an example that will graphically illustrate the interpretation of the CGA for multiple subproblems as a method for optimally solving an LDP. Consider the following IP:

$$\begin{array}{lrcrcrcrcl}
\min z = & x_1 & + & 3x_2 & - & 3x_3 & + & 9x_4 & & \\
\text{s.t.} & x_1 & + & 2x_2 & - & x_3 & + & 3x_4 & = & 8 \\
& -x_1 & + & x_2 & & & & & \leq & 0 \\
& x_1 & + & x_2 & & & & & \leq & 2 \\
& & & & & 2x_3 & + & x_4 & \leq & 2 \\
& & & & & & & \boldsymbol{x} & \in & \mathbb{Z}^4_+.
\end{array}$$

By dualizing its first constraint, the obtained LS is:

$$\begin{array}{lrcrcrcrcl}
\min & L(\, x_1, \, x_2, \, x_3, \, x_4, \, \pi) = & & & & & & & & \\
& (1-\pi)x_1 & + & (3-2\pi)x_2 & + & (-3+\pi)x_3 & + & (9-3\pi)x_4 & + & 8\pi \\
\text{s.t.} & -x_1 & + & x_2 & & & & & \leq & 0 \\
& x_1 & + & x_2 & & & & & \leq & 2 \\
& & & & & 2x_3 & + & x_4 & \leq & 2 \\
& & & & & & & \boldsymbol{x} & \in & \mathbb{Z}^4_+.
\end{array}$$

This LS decomposes into two subproblems:

$$\begin{array}{lrcrcrcl}
\xi^{1*} = \min & (1-\pi)x_1 & + & (3-2\pi)x_2 & & & \\
\text{s.t.} & -x_1 & + & x_2 & \leq & 0 & \\
& x_1 & + & x_2 & \leq & 2 & \\
& x_1, & & x_2 & \in & \mathbb{Z}_+, &
\end{array} \tag{5.27}$$

and

$$\begin{array}{lrcrcrcl}
\xi^{2*} = \min & (-3+\pi)x_3 & + & (9-3\pi)x_4 & & & \\
\text{s.t.} & 2x_3 & + & x_4 & \leq & 2 & \\
& x_3, & & x_4 & \in & \mathbb{Z}_+, &
\end{array} \tag{5.28}$$

having $Q^1 = \{(0\,0), (1\,0), (2\,0), (1\,1)\}$ and $Q^2 = \{(0\,0), (1\,0), (0\,1), (0\,2)\}$ as their solution sets. Consider an equivalent DW reformulation that keeps the first constraint in the master and its solution by the CGA. Noticing that both subproblems have

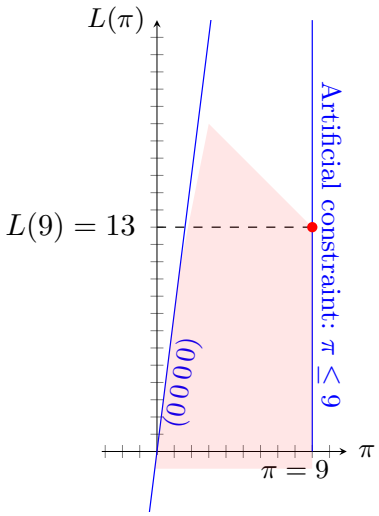
(0 0) as solutions and introducing an artificial variable with cost 9, the first RMLP is:

$$\begin{array}{rlrcrcrcl} z_{\mathrm{RM}} = & \min & 9a_1 & + & 0\lambda_1^1 & + & 0\lambda_1^2 & & \\ & \text{s.t.} & a_1 & + & 0\lambda_1^1 & + & 0\lambda_1^2 & = & 8 \\ & & & & \lambda_1^1 & & & = & 1 \\ & & & & & & \lambda_1^2 & = & 1 \\ & & & & & & (\boldsymbol{a}, \boldsymbol{\lambda}) & \geq & \mathbf{0}. \end{array}$$

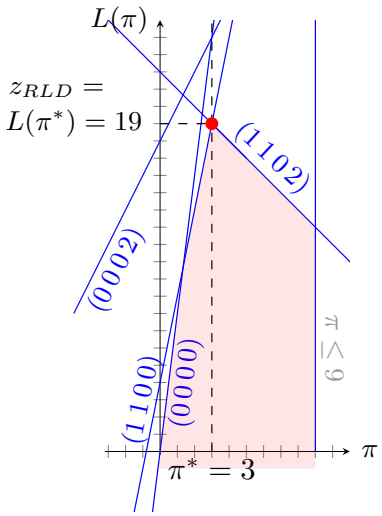
This RMLP can be solved by only looking at the graph of the corresponding Restricted Lagrangian Dual Problem (RLDP) in format (5.12), which is $z_{\mathrm{RLD}} = \max z$ s.t. $\pi \leq 9$, $z \leq 8\pi$, where the second constraint corresponds to the concatenation of the solutions $(0,0) \in Q^1$ and $(0,0) \in Q^2$. This is shown in Figure 5.4a. Its maximum point is $(\pi = 9, z_{\mathrm{RLD}} = 72)$. So, an optimal value for the dual variable of the RMLP first constraint is $\pi = 9$ and $z_{\mathrm{RM}} = z_{\mathrm{RLD}} = 72$. The artificial dual constraint $\pi \leq 9$ prevented that RLDP from being unbounded. Solving the subproblems defined in (5.27) and (5.28) using $\pi = 9$, one gets $\xi^{1*} = -23$ with $\hat{\boldsymbol{x}}^1 = (1\,1)$ and $\xi^{2*} = -36$ with $\hat{\boldsymbol{x}}^2 = (0\,2)$, respectively, so $L(9) = -23 - 36 + 72 = 13$. The next RMLP is:

$$\begin{array}{rlrcrcrcrcrcl} z_{\mathrm{RM}} = & \min & 9a_1 & + & 0\lambda_1^1 & + & 0\lambda_1^2 & + & 4\lambda_2^1 & + & 18\lambda_2^2 & & \\ & \text{s.t.} & a_1 & + & 0\lambda_1^1 & + & 0\lambda_1^2 & + & 3\lambda_2^1 & + & 6\lambda_2^2 & = & 8 \\ & & & & \lambda_1^1 & & & + & \lambda_2^1 & & & = & 1 \\ & & & & & & \lambda_1^2 & & & + & \lambda_2^2 & = & 1 \\ & & & & & & & & & & (\boldsymbol{a}, \boldsymbol{\lambda}) & \geq & \mathbf{0}. \end{array}$$

Here we can see the "magic of multiple subproblems" on CG convergence. The introduction of *two* columns in the RMLP yields *three* additional constraints in the RLDP. This happens because the $S^1 = \{(0\,0), (1\,1)\}$ and $S^2 = \{(0\,0), (0\,2)\}$ combine into $S = \{(0\,0\,0\,0), (0\,0\,0\,2), (1\,1\,0\,0), (1\,1\,0\,2)\}$, so the RDLP is $z_{\mathrm{RLD}} = \max z$ s.t. $\pi \leq 9, z \leq 8\pi$, $z \leq 2\pi + 18$, $z \leq 5\pi + 4$, $z \leq -\pi + 22$, shown in Figure 5.4b. Its maximum point is $(\pi = 3, z_{\mathrm{RLD}} = 19)$. So, an optimal value for the dual variable of the RMLP first constraint is $\pi = 3$ and $z_{\mathrm{RM}} = z_{\mathrm{RLD}} = 19$. Solving the subproblems using $\pi = 3$, one gets $\xi^{1*} = -5$ and $\xi^{2*} = 0$, so $L(3) = -5+0+24 = 19$. So, $\pi^* = 3$ and $z_{\mathrm{M}} = z_{\mathrm{LD}} = 19$ is the optimal Lagrangian Dual.



(a) RLDP with $S = \{(0\,0\,0\,0)\}$ and maximum $(\pi = 9, z_{\text{RLD}} = 72)$



(b) RLDP with $S = \{(0\,0\,0\,0), (0\,0\,0\,2), (1\,1\,0\,0), (1\,1\,0\,2)\}$ and optimal LDP solution $(\pi = 3, z_{\text{LD}} = 19)$

Figure 5.4: Lagrangian Dual Problem and sequence of Restricted Lagrangian Dual Problems

5.3. LR Resolution Methods

LR resolution methods are iterative algorithms that solve LDPs assuming that there exists a practically efficient auxiliary algorithm for solving the corresponding Lagrangian Subproblem at a given point. Such methods should generate a sequence of points $(\boldsymbol{\pi}^0\ \ \boldsymbol{\rho}^0), (\boldsymbol{\pi}^1\ \ \boldsymbol{\rho}^1), \ldots$ such that $L(\boldsymbol{\pi}^t,\ \boldsymbol{\rho}^t)$ is guaranteed to converge to the optimum value of (5.3), i.e., $\lim_{t\to\infty} L(\boldsymbol{\pi}^t,\ \boldsymbol{\rho}^t) = z_{\text{LD}}$. It should be noted that the convergence does not need to be (and rarely is) monotonic, meaning that at some iterations t we may have $L(\boldsymbol{\pi}^t,\ \boldsymbol{\rho}^t) < L(\boldsymbol{\pi}^{t-1},\ \boldsymbol{\rho}^{t-1})$. Of course, a main concern when designing an LR resolution method is the practical speed of the convergence.

There are numerous families of LR resolution methods. We present two such families: the subgradient methods, which are the most typical "Lagrangian Methods"; and the cutting plane methods, which are closely related (sometimes even equivalent) to CG.

5.3.1. Subgradient Methods

We discuss some subgradient methods for solving an LDP in format (5.3), which corresponds to the maximization of a concave function. In that case, subgradient methods use *supergradient* vectors. The reason for the methods' name is historical: they were first proposed for the minimization of convex functions, where subgradient methods indeed use subgradient vectors.

Theorem 5.6: *Let* $\boldsymbol{x}^0 \in X$, $\boldsymbol{\pi}^0 \in \mathbb{R}^{1\times m}$, *and* $\boldsymbol{\rho}^0 \in \mathbb{R}^{1\times r}_+$ *such that* $L(\boldsymbol{\pi}^0, \boldsymbol{\rho}^0) = L(\boldsymbol{x}^0, \boldsymbol{\pi}^0, \boldsymbol{\rho}^0)$. *Then,* $\boldsymbol{s} = \begin{pmatrix} \boldsymbol{g}(\boldsymbol{x}^0) \\ \boldsymbol{h}(\boldsymbol{x}^0) \end{pmatrix}$ *is a supergradient of function* $L(\boldsymbol{\pi}, \boldsymbol{\rho})$ *at* $(\boldsymbol{\pi}^0 \;\; \boldsymbol{\rho}^0)$, *i.e., for any* $\boldsymbol{\pi} \in \mathbb{R}^{1\times m}$ *and* $\boldsymbol{\rho} \in \mathbb{R}^{1\times r}_+$,

$$L(\boldsymbol{\pi}, \boldsymbol{\rho}) \leq L(\boldsymbol{\pi}^0, \boldsymbol{\rho}^0) + (\,(\boldsymbol{\pi}\;\boldsymbol{\rho}) - (\boldsymbol{\pi}^0\;\boldsymbol{\rho}^0)\,)\,\boldsymbol{s}.$$

Proof.

$$\begin{aligned} L(\boldsymbol{\pi}, \boldsymbol{\rho}) &= \min_{\boldsymbol{x}\in X} L(\boldsymbol{x}, \boldsymbol{\pi}, \boldsymbol{\rho}) \leq L(\boldsymbol{x}^0, \boldsymbol{\pi}, \boldsymbol{\rho}) = c(\boldsymbol{x}^0) + \boldsymbol{\pi}\,\boldsymbol{g}(\boldsymbol{x}^0) + \boldsymbol{\rho}\,\boldsymbol{h}(\boldsymbol{x}^0) \\ &= c(\boldsymbol{x}^0) + \boldsymbol{\pi}^0\,\boldsymbol{g}(\boldsymbol{x}^0) + \boldsymbol{\rho}^0\,\boldsymbol{h}(\boldsymbol{x}^0) + \\ &\qquad\qquad (\boldsymbol{\pi} - \boldsymbol{\pi}^0)\,\boldsymbol{g}(\boldsymbol{x}^0) + (\boldsymbol{\rho} - \boldsymbol{\rho}^0)\,\boldsymbol{h}(\boldsymbol{x}^0) \\ &= L(\boldsymbol{\pi}^0, \boldsymbol{\rho}^0) + (\boldsymbol{\pi} - \boldsymbol{\pi}^0)\,\boldsymbol{g}(\boldsymbol{x}^0) + (\boldsymbol{\rho} - \boldsymbol{\rho}^0)\,\boldsymbol{h}(\boldsymbol{x}^0) \\ &= L(\boldsymbol{\pi}^0, \boldsymbol{\rho}^0) + (\,(\boldsymbol{\pi}\;\boldsymbol{\rho}) - (\boldsymbol{\pi}^0\;\boldsymbol{\rho}^0)\,)\,\boldsymbol{s}. \end{aligned}$$

The first, second and fourth equalities are true by definition of $L(\boldsymbol{\pi}, \boldsymbol{\rho})$, $L(\boldsymbol{x}, \boldsymbol{\pi}, \boldsymbol{\rho})$ and $L(\boldsymbol{\pi}^0, \boldsymbol{\rho}^0)$, respectively. □

Consider the LDP of the example in Section 5.1.1 (its LDF $L(\rho) = 8\rho - 6\sqrt{\rho}$ is depicted in Figure 5.2). According to Theorem 5.6, supergradients at $\rho^1 = 1/16$, $\rho^2 = 9/64$, and $\rho^3 = 25/49$ are $h((4\;2)) = 4$, $h((8/3\;4/3)) = 0$, and $h((7/5\;7/10)) = -3.8$, respectively. Thus, $L(\rho) \leq \frac{3}{4} + 4\rho$, $L(\rho) \leq 1.125$ and $L(\rho) \leq \frac{15}{7} - 3.8\rho$. Each line depicted in Figure 5.2 corresponds to the points where one of the previous inequalities holds with equality. *As this LDF is differentiable, all supergradients are also gradients.* Therefore, there is a unique supergradient of $L(\rho)$ at each point $\rho^0 \geq 0$.

Alternatively, consider the LDP of the example in Section 5.2.1. Its LDF $L(\rho)$ (depicted in Figure 5.3a) is non-differentiable at some points, like in $\rho^0 = 1/3$. Depending on which optimal solution $\boldsymbol{x}^0$ of $L(\boldsymbol{x}, \rho^0)$ is used, Theorem 5.6 yields different supergradients. In particular, $h((1\ 1)) = -1$ and $h((3\ 0)) = 2$ are supergradients. Indeed, any $s \in [-1, 2]$ is a supergradient of $L(\rho)$ at $\rho^0 = 1/3$.

Subgradient methods for LR are iterative algorithms that use supergradients (possibly together with other information) to generate a sequence of points converging to an optimal solution of (5.3).

5.3.1.1. The Subgradient Method

There are many possibilities to generate a sequence that guarantees convergence. The Subgradient Method is characterized by always using the supergradient direction at the current point to generate the next one. Variants of that method essentially differ in the way the step size is chosen. The general Subgradient Method is presented in Algorithm 3:

Algorithm 3 The Subgradient Method for Solving (5.3)

1: $t \leftarrow 0$
2: $(\ \boldsymbol{\pi}^0\ \ \boldsymbol{\rho}^0\ ) \leftarrow$ initial multipliers
3: **repeat**
4: $\quad \boldsymbol{x}^t \leftarrow \arg\min_{\boldsymbol{x} \in X} L(\boldsymbol{x}, \boldsymbol{\pi}^t, \boldsymbol{\rho}^t)$ ▷ Solve the LS
5: $\quad \boldsymbol{s} \leftarrow \begin{pmatrix} \boldsymbol{g}(\boldsymbol{x}^t) \\ \boldsymbol{h}(\boldsymbol{x}^t) \end{pmatrix}$ ▷ Calculate a supergradient
6: $\quad$ Compute step size δ^t
7: $\quad (\ \boldsymbol{\pi}^{t+1}\ \ \boldsymbol{\rho}^{t+1}\ ) \leftarrow \boldsymbol{\gamma}\left((\ \boldsymbol{\pi}^t\ \ \boldsymbol{\rho}^t\ ) + \delta^t\, \boldsymbol{s}^\intercal\right)$
8: $\quad t \leftarrow t + 1$
9: **until** stopping condition is met
10: **return** $(\ \boldsymbol{\pi}^t\ \ \boldsymbol{\rho}^t\ )$

- The multipliers are initialized in Line 2. They are often set to zero values. However, they may be hot-started with a (hopefully) better guess.
- Line 4 solves the LS with the current multipliers to find an optimal solution $\boldsymbol{x}^t$ (any optimal solution is OK), which is used for calculating the supergradient s in Line 5.

- The computing of the step size δ^t at Line 6 tries to balance theoretical convergence even in the worst case with good practical performance.

- In Line 7, the supergradient s defines the direction to move from the current point to the next one. The actual movement in that direction depends on the scalar step size δ^t. The truncating function $\boldsymbol{\gamma}(\,(\boldsymbol{\pi}\ \boldsymbol{\rho})\,) = (\,\boldsymbol{\pi}\ \max\{\boldsymbol{\rho}, \mathbf{0}\}\,)$ (where the maximum value of vectors is taken element-wise) is necessary to ensure that the next point is feasible.

  Some authors replace Line 7 with $(\,\boldsymbol{\pi}^{t+1}\ \ \boldsymbol{\rho}^{t+1}\,) \leftarrow \boldsymbol{\gamma}((\,\boldsymbol{\pi}^t\ \ \boldsymbol{\rho}^t\,) + \delta^t \frac{\boldsymbol{s}^\intercal}{||\boldsymbol{s}||})$, normalizing the supergradient direction. If so, δ^t should be called an *step length*, as $||(\,\boldsymbol{\pi}^{t+1}\ \ \boldsymbol{\rho}^{t+1}\,) - (\,\boldsymbol{\pi}^t\ \ \boldsymbol{\rho}^t\,)|| \leq \delta^t$, with equality holding when $(\,\boldsymbol{\pi}^{t+1}\ \ \boldsymbol{\rho}^{t+1}\,) = (\,\boldsymbol{\pi}^t\ \ \boldsymbol{\rho}^t\,) + \delta^t \frac{\boldsymbol{s}^\intercal}{||\boldsymbol{s}||}$.

- The stopping condition at Line 9 may be as simple as only an iteration count. More sophisticated conditions try to determine whether the algorithm is already very close to the optimum. For example, checking if the difference between the current and the previous point is smaller than a given tolerance, or if the best Lagrangian bound already achieved does not improve significantly after a certain number of iterations.

The classic LR algorithm for solving the TSP by Held and Karp [1971] used as step size at iteration t:

$$\delta^t = \alpha \frac{\overline{L} - L(\boldsymbol{\pi}^t, \boldsymbol{\rho}^t)}{||\boldsymbol{s}||^2}, \tag{5.29}$$

where the denominator is the square of the Euclidean norm of the subgradient, $\overline{L}$ is an upper bound on z_{LD}, and α is a value such that $0 < \alpha \leq 2$. Held et al. [1974] proposed starting with $\alpha = 2$ and decreasing α by a fixed factor after every block of iterations, each block being defined by a certain number of iterations. When the method is being used for solving the Lagrangian Dual Problem of an IP, the cost of any known feasible solution to the original IP can provide a value for $\overline{L}$.

A Subgradient Method variant with a good definition of the step size can work reasonably well in many cases. However, in many other practical cases, its convergence can be very slow and this can not be fixed by finding better steps. The problem is that the supergradient direction is not necessarily a good direction to move towards the optimum value of (5.3). An example of this behavior is illustrated in Figure 5.5, where the contour lines of a 2D piecewise linear function $L(\boldsymbol{\pi})$ are

depicted as triangles. The optimum solution is the point inside the innermost triangle. Each point $\boldsymbol{\pi}^t$, for $t = 0, \ldots, 5$, is labeled by the number t in the figure, and the adjacent arrows indicate the supergradient directions. Note that, except for the unlikely case that the method falls exactly into one of the non-differentiable points, any supergradient computed in the region that surrounds the depicted points will have one of the two directions shown in the figure. In this case, the only way to reach the optimal value in fewer iterations would be to adopt a much larger step size to escape from that region, but that would require a global knowledge of the function, which is not available.

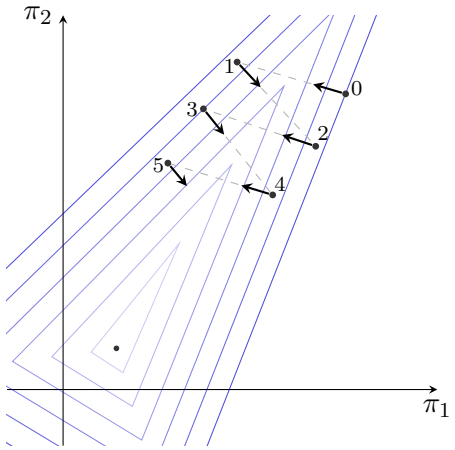


Figure 5.5: A bad case for the Subgradient Method.

5.3.1.2. Conjugate Subgradient and Bundle Methods

One way to mitigate the bad behavior of the Subgradient Method is to replace the single supergradient direction with a combination of several supergradients already computed. This is the idea behind the quite similar methods proposed by Lemaréchal [1975] and Wolfe [1975] (as both articles were published in the same journal issue, they cite each other and refer to the set of supergradients used to compute a direction as a *bundle*). We outline how directions are obtained in the Conjugate Subgradient method as proposed in Wolfe [1975]:

- Let $\boldsymbol{B}$ be the matrix where the rows are the supergradients in the current

bundle, which always contains the supergradient at the current point. The used direction is the convex combination of those supergradients with minimum Euclidian norm. In other words, $\boldsymbol{d} = \boldsymbol{\lambda}^* \boldsymbol{B}$ where $\boldsymbol{\lambda}^*$ is the solution of $z^* = \min ||\boldsymbol{\lambda B}||$ s.t. $\boldsymbol{\lambda 1} = 1, \boldsymbol{\lambda} \geq \boldsymbol{0}$, a quadratic optimization problem that can be efficiently solved by specialized methods (see Wolfe [1976] for a deeper discussion). If $z^* = 0$ then $\boldsymbol{d} = \boldsymbol{0}$ is not a valid direction. In such case there are two possibilities:

- If the points that generated the supergradients used in the convex combination (i.e., those corresponding to rows i such that $\lambda_i^* > 0$) are sufficiently close, it is considered that those points are already near-optimal solutions of (5.3). The algorithm stops.
- Otherwise, the bundle is reset to contain only the current supergradient (a discussed variant of the method keeps some other supergradients in the bundle).

If $z^* > 0$ direction $\boldsymbol{d}$ is valid. However, instead of using a step size given by a formula, the next point is determined by a *line search* over direction $\boldsymbol{d}$. The supergradient of that next point is added to the bundle.

We use the example in Figure 5.3a to give an intuition on the above stopping criteria. Consider the bundle obtained by the supergradients at points $\rho^1 = 0$ and $\rho^2 = 2$. Matrix $\boldsymbol{B} = \binom{2}{-3}$ gives $z^* = 0$, meaning that ρ^1 and ρ^2 sampled $L(\rho)$ to the left and to the right of the optimal point $\rho^* = 1/3$. However, those two points are too far apart, so there is no guarantee that any of them is close to the optimum and the algorithm can not stop. Points $\rho^3 = 0.99$ and $\rho^4 = 1.01$ are close, but the bundle with their supergradients $\boldsymbol{B} = \binom{-1}{-3}$ yields $z^* = 1 > 0$, meaning that those points were not properly sampled around ρ^*. Finally, points $\rho^5 = 0.33$ and $\rho^6 = 0.34$ are close and the bundle with their supergradients yields $\boldsymbol{B} = \binom{2}{-1}$ and $z^* = 0$, so the algorithm can stop. Indeed, those examples also provide intuition on why the method chooses directions with minimum norms. By doing so, at each non-resetting iteration, the convex hull of the bundle gets closer to the origin. It is a way of inducing the creation of a bundle that samples $L(\pi, \rho)$ "around" (in all its dimensions) the set of its optimal solutions.

5.3.2. Cutting Plane (Dual Column Generation) Methods

5.3.2.1. Single Subproblem

Problem (5.3), assuming for simplification that no multipliers $\boldsymbol{\rho}$ exist, can be rewritten as the following "LP" with an infinite number of constraints[2]:

$$\begin{aligned} \max \quad & z && && \text{(5.30a)} \\ \text{s.t.} \quad & z \leq L(\boldsymbol{\pi}') + (\boldsymbol{\pi} - \boldsymbol{\pi}')\boldsymbol{s}(\boldsymbol{\pi}') && \boldsymbol{\pi}' \in \mathbb{R}^{1\times m} && \text{(5.30b)} \\ & \boldsymbol{\pi} \in \mathbb{R}^{1\times m}, && && \text{(5.30c)} \end{aligned}$$

where $\boldsymbol{s}(\boldsymbol{\pi}')$ is a function that returns some supergradient of $L(\boldsymbol{\pi})$ at $\boldsymbol{\pi}'$. By the definition of supergradient, $L(\boldsymbol{\pi}) \leq L(\boldsymbol{\pi}') + (\boldsymbol{\pi} - \boldsymbol{\pi}')\boldsymbol{s}(\boldsymbol{\pi}')$ for all $\boldsymbol{\pi}, \boldsymbol{\pi}' \in \mathbb{R}^{1\times m}$, an inequality that becomes equality when $\boldsymbol{\pi} = \boldsymbol{\pi}'$. So, if $\boldsymbol{\pi}^*$ is an optimal solution of (5.3) with value z_{LD} then $(z_{\text{LD}}, \boldsymbol{\pi}^*)$ is an optimal solution of (5.30).

The cutting plane method by Cheney and Goldstein [1959] and Kelley, Jr [1960] converges to an optimal solution of (5.30) by solving a sequence of LPs having only a finite (and small) subset of the Constraints (5.30b). At each iteration, the method maintains an outer approximation of the hypograph of $L(\boldsymbol{\pi})$. An example of such an approximation is depicted in Figure 5.2 if we consider the polyhedral region below the green, red, and orange lines. Note that this region approximates the hypograph of (5.30), which is precisely the region shaded in blue. Algorithm 4 presents a pseudocode for the cutting plane method.

By considering the dual of (5.30) as a "Master LP" with an infinite number of variables, one may view the cutting plane method as a kind of dual CG that can be applied in any LR context, including the cases where the original problem is nonlinear. However, if the original problem is an LP or a MIP, like those considered in Chapter 2 and 4, the LDF is piecewise linear, meaning that there are only a finite number of distinct and non-redundant inequalities in (5.30b). In those cases, Algorithm 4 can be run with tolerance $\epsilon = 0$ and is guaranteed to finish with an optimal solution, becoming equivalent to the Column Generation Algorithm. This is why some authors in the Lagrangian community refer to the basic CGA as the

[2] "LP" because almost all authors assume that proper LPs should have finite size.

Algorithm 4 Cutting Plane Algorithm for Solving (5.30)

$t \leftarrow 0$
LP $\leftarrow \max z$ s.t. $\boldsymbol{\pi} \in \mathbb{R}^{1\times m}$ + artificial constraints to avoid unboundedness
repeat
  $t \leftarrow t+1$
  Solve LP obtaining solution $(z^t, \boldsymbol{\pi}^t)$
  $\boldsymbol{x}^t \leftarrow \arg\min_{\boldsymbol{x}\in X} L(\boldsymbol{x}, \boldsymbol{\pi}^t)$ ▷ Solve the LS
  $\boldsymbol{s} \leftarrow \boldsymbol{g}(\boldsymbol{x}^t)$ ▷ Get supergradient
  if $z^t > L(\boldsymbol{\pi}^t) + \epsilon$ **then** ▷ ϵ is a small tolerance
    Add constraint $z \leq L(\boldsymbol{\pi}^t) + (\boldsymbol{\pi} - \boldsymbol{\pi}^t)\boldsymbol{s}$ to LP
  end if
until no constraint was added to LP or other stopping condition is met
return π^t

Cheney-Goldstein-Kelley Algorithm (more often as only Kelley's Algorithm). We provide an example of the cutting plane/dual column generation method converging to the optimal solution of an LDP arising from non-linear optimization. Indeed, such an approach can be particularly interesting when the resulting LS can be decomposed into many independent subproblems.

5.3.2.2. Multiple Subproblems

Cheney-Goldstein-Kelley algorithm can be extended to handle multiple subproblems, some of them identical, through a development analogous to the one presented in Subsection 5.2.2. In this way, we show that one can still profit from the "magic of multiple subproblems" used in the CG approach to improve the convergence of non-linear problems that become separable when applying LR.

Consider a generic problem in the following format:

$$\min \quad z = c^1(\boldsymbol{x}^1) + \cdots + c^U(\boldsymbol{x}^U) \tag{5.31a}$$
$$\text{s.t.} \quad \boldsymbol{g}^1(\boldsymbol{x}^1) + \cdots + \boldsymbol{g}^U(\boldsymbol{x}^U) = \boldsymbol{b} \tag{5.31b}$$
$$\boldsymbol{x}^u \in \boldsymbol{X}^u \qquad u \in [U]. \tag{5.31c}$$

For each $u \in [U]$, $X^u \subseteq \mathbb{R}^{n^u}$ be a (possibly infinite) set of points, and let $c^u : X^u \mapsto \mathbb{R}$, and $\boldsymbol{g}^u : X^u \mapsto \mathbb{R}^m$ be functions defined over such points. The special structure in this problem is that (5.31c) consists of U independent sets of constraints, each

such set being expressed over a distinct subset of the variables.

As done in Section 2.3.2, we also assume that $[U]$ can be partitioned into K maximal groups of identical subproblems. Group $k \in [K]$ has U^k subproblems. The first K subproblems are distinct and the remaining $U-K$ subproblems are identical to some of those first K subproblems. Let $U(k) \subseteq [U]$ be the indices associated with group $k \in [K]$. Dualizing Constraints (5.31b), we obtain the following LS for any fixed $\boldsymbol{\pi} \in \mathbb{R}^{1\times m}$:

$$\begin{aligned}
\min \quad L(\boldsymbol{x},\boldsymbol{\pi}) &= \sum_{u\in[U]} c^u(\boldsymbol{x}^u) + \boldsymbol{\pi}\left(\boldsymbol{b} - \sum_{u\in[U]} \boldsymbol{g}^u(\boldsymbol{x}^u)\right) \\
&= \sum_{u\in[U]} (c^u(\boldsymbol{x}^u) - \boldsymbol{\pi}\boldsymbol{g}^u(\boldsymbol{x}^u)) + \boldsymbol{\pi}\boldsymbol{b} \\
&= \sum_{k\in[K]} \sum_{u\in U(k)} (c^k(\boldsymbol{x}^u) - \boldsymbol{\pi}\boldsymbol{g}^k(\boldsymbol{x}^u)) + \boldsymbol{\pi}\boldsymbol{b} \qquad (5.32a) \\
\text{s.t.} \quad \boldsymbol{x}^u \in X^u \qquad & u \in [U]. \qquad (5.32b)
\end{aligned}$$

By considering (5.32) as defining a single subproblem, all results in Section 5.2.1 remain valid. However, it is interesting to exploit the fact that (5.32) can be decomposed into K independent subproblems. For each $k \in [K]$, the subproblem to be solved is:

$$\begin{aligned}
\xi^k(\boldsymbol{\pi}) = \min \quad & \boldsymbol{c}^k(\boldsymbol{x}^k) - \boldsymbol{\pi}\boldsymbol{g}^k(\boldsymbol{x}^k) \qquad (5.33a) \\
\text{s.t.} \quad & \boldsymbol{x}^k \in X^k, \qquad (5.33b)
\end{aligned}$$

and $L(\boldsymbol{\pi}) = \sum_{k\in[K]} U^k \xi^k(\boldsymbol{\pi}) + \boldsymbol{\pi}\boldsymbol{b}$. Note that, just like $L(\boldsymbol{\pi})$, function $\xi^k(\boldsymbol{\pi})$, $k \in [K]$, is the pointwise minimum of a number (possibly infinite) of linear functions over $\boldsymbol{\pi}$. Thus, they are concave functions too. Moreover, when solving (5.33) for a given $\boldsymbol{\pi}' \in \mathbb{R}^{1\times m}$ to obtain the value of $\xi^k(\boldsymbol{\pi}')$, the optimal solution x^{k*} can be used to compute $\boldsymbol{g}^k(\boldsymbol{x}^{k*})$, which is a supergradient of $\xi^k(\boldsymbol{\pi})$ at $\boldsymbol{\pi}'$.

Theorem 5.7: *Let $\boldsymbol{x}^{k*} \in X^k$, and $\boldsymbol{\pi}' \in \mathbb{R}^{1\times m}$ such that $\xi^k(\boldsymbol{\pi}') = \boldsymbol{c}^k(\boldsymbol{x}^{k*}) - \boldsymbol{\pi}'\boldsymbol{g}^k(\boldsymbol{x}^{k*})$. Then, $\boldsymbol{s}^k = \boldsymbol{g}^k(\boldsymbol{x}^{k*})$ is a supergradient of function $\xi^k(\boldsymbol{\pi})$ at $\boldsymbol{\pi}'$, i.e., for any $\boldsymbol{\pi} \in \mathbb{R}^{1\times m}$,*

$$\xi^k(\boldsymbol{\pi}) \leq \xi^k(\boldsymbol{\pi}') + (\boldsymbol{\pi} - \boldsymbol{\pi}')\,\boldsymbol{s}^k.$$

Proof.

$$\begin{aligned}\xi^k(\boldsymbol{\pi}) &= \min_{\boldsymbol{x}^k \in X^k} \{\boldsymbol{c}^k(\boldsymbol{x}^k) - \boldsymbol{\pi}\boldsymbol{g}^k(\boldsymbol{x}^k)\} \leq \boldsymbol{c}^k(\boldsymbol{x}^{k*}) - \boldsymbol{\pi}\boldsymbol{g}^k(\boldsymbol{x}^{k*}) \\ &= c(\boldsymbol{x}^{k*}) + \boldsymbol{\pi}'\,\boldsymbol{g}(\boldsymbol{x}^{k*}) + (\boldsymbol{\pi} - \boldsymbol{\pi}')\,\boldsymbol{g}(\boldsymbol{x}^{k*}) \\ &= \xi^k(\boldsymbol{\pi}') + (\boldsymbol{\pi} - \boldsymbol{\pi}')\,\boldsymbol{s}^k.\end{aligned}$$

The first and the last equalities are true by definition of $\xi^k(\boldsymbol{\pi})$, and by the theorem statement, respectively. □

Now, let $\boldsymbol{s}^k(\boldsymbol{\pi}')$ denote a supergradient of $\xi^k(\boldsymbol{\pi})$ at $\boldsymbol{\pi}'$. Defining ν_k, $k \in [K]$, as auxiliary variables, the LDP of (5.31) can be written as:

$$z_{\text{LD}} = \max \sum_{k\in[K]} U^k \nu_k + \boldsymbol{\pi}\boldsymbol{b} \tag{5.34a}$$

$$\text{s.t.} \quad \nu_k \leq \xi^k(\boldsymbol{\pi}') + (\boldsymbol{\pi} - \boldsymbol{\pi}')\,\boldsymbol{s}^k(\boldsymbol{\pi}') \qquad k \in [K], \boldsymbol{\pi}' \in \mathbb{R}^{1\times m} \tag{5.34b}$$

$$\boldsymbol{\nu} \in \mathbb{R}^K \tag{5.34c}$$

$$\boldsymbol{\pi} \in \mathbb{R}^{1\times m} \tag{5.34d}$$

$$\tag{5.34e}$$

The LDP in the form (5.34) can also be solved using the standard cutting plane algorithm. Indeed, Kelley, Jr [1960] proposes to add only the most violated cut in each iteration. Later, Dinkel et al. [1977] observed that adding all violated cuts leads to a superior performance. This observation is consistent with the standard practices of the CG community. The modified cutting plane is presented as Algorithm 5. Note that (5.34b) is equivalent to

$$\nu_k \leq \boldsymbol{c}^k(\boldsymbol{x}^{k*}) - \boldsymbol{\pi}\boldsymbol{g}^k(\boldsymbol{x}^{k*}) \tag{5.35}$$

by the development of the proof of Theorem 5.7. This form has the advantage of not depending on $\boldsymbol{\pi}'$, and is still valid even if $\boldsymbol{x}^{k*}$ is not an optimal subproblem solution. This makes it suitable to initialize the LP in line 2 of Algorithm 5 when some feasible subproblem solutions are available, reducing the need for artificial constraints and also accelerating the convergence. We use that observation in the numerical example presented next.

Consider the problem over variables $\boldsymbol{x} = (x_1\, x_2\, x_3)$ with a non-linear objective

Algorithm 5 Cutting Plane Algorithm for Solving (5.34)

$t \leftarrow 0$
LP $\leftarrow \max \sum_{k\in[K]} U^k\nu_k + \boldsymbol{\pi b}$ s.t. $\boldsymbol{\pi} \in \mathbb{R}^{1\times m}$ + artificial constraints to avoid unboundedness
repeat
  $t \leftarrow t + 1$
  Solve LP obtaining solution $(\boldsymbol{\nu}^t, \boldsymbol{\pi}^t)$
  for each $k \in [K]$ **do**
    $\boldsymbol{x}^{kt} \leftarrow \arg\min_{\boldsymbol{x}^k\in X^k}\{\boldsymbol{c}^k(\boldsymbol{x}^k) - \boldsymbol{\pi}^t\boldsymbol{g}^k(\boldsymbol{x}^k)\}$ ▷ Solve the LS
    $\boldsymbol{s}^k \leftarrow \boldsymbol{g}^k(\boldsymbol{x}^{kt})$ ▷ Get supergradient
    if $\nu_k^t > \xi^k(\boldsymbol{\pi}^t) + \epsilon$ **then** ▷ ϵ is a small tolerance
      Add constraint $\nu_k \leq \xi^k(\boldsymbol{\pi}^t) + (\boldsymbol{\pi} - \boldsymbol{\pi}^t)\,\boldsymbol{s}^k$ to LP
    end if
  end for
until no constraint was added to LP or other stopping condition is met
return π^t

function and a non-linear quadratic constraint:

$$\min \quad \frac{x_1^2}{50} - \ln(x_2+1) - \ln(x_3+1) \tag{5.36a}$$
$$\text{s.t.} \quad x_1 + 3 \geq x_2^2 + x_3^2 \tag{5.36b}$$
$$\boldsymbol{x} \in \mathbb{R} \times \mathbb{Z}_+^2. \tag{5.36c}$$

Dualizing Constraint (5.36b), for any fixed $\rho \in \mathbb{R}_+$ we have the following Lagrangian Subproblem:

$$\min \quad L(\boldsymbol{x},\rho) = \frac{x_1^2}{50} - \ln(x_2+1) - \ln(x_3+1) + \rho(x_2^2 + x_3^2 - x_1 - 3) \tag{5.37a}$$
$$\text{s.t.} \quad \boldsymbol{x} \in \mathbb{R}_+ \times \mathbb{Z}_+^2. \tag{5.37b}$$

This LS can be decomposed into three independent problems. The first two define the following functions: $\xi^1(\rho) = \min \frac{x_1^2}{50} - \rho x_1$ s.t. $x_1 \in \mathbb{R}_+$, and $\xi^2(\rho) = \min \rho x_2^2 - \ln(x_2+1)$ s.t. $x_2 \in \mathbb{Z}_+$. The third subproblem is identical to the second. Then, Constraint (5.36b) can be written as $g^1(x_1) + g^2(x_2) + g^2(x_3) \leq 3$, where $g^1(x_1) = -x_1$, and $g^2(x_2) = x_2^2$. We can use elementary calculus to find an analytical expression to $\xi^1(\rho)$. Since the objective function of the corresponding

subproblem is strictly convex for fixed ρ, its minimum value is attained at the only point $x_1^* = 25\rho$ where the derivative with respect to x_1 is zero. Then, we obtain that $\xi^1(\rho) = -12.5\rho^2$. The same approach can be applied to the linear relaxation of the subproblem that defines $\xi^2(\rho)$, resulting in

$$\overline{x}_2^* = \frac{-1 + \sqrt{1 + 2/\rho}}{2},$$

for $\rho > 0$. Since this subproblem has a single decision variable and a strictly convex objective function, its optimal solution x_2^* is necessarily either $\lfloor \overline{x}_2^* \rfloor$ or $\lceil \overline{x}_2^* \rceil$, which can be computed by inspection.

We now solve its LDP using Algorithm 5. The LP is initialized in Line 2 with an artificial constraint bounding ν_1, and two constraints derived from the feasible solution (73 3 3), using (5.35). The initial LP is:

$$\begin{aligned}
\max \quad & \nu_1 + 2\nu_2 - 3\rho && \text{(5.38a)}\\
\text{s.t.} \quad & \nu_1 \leq 99 && \text{(5.38b)}\\
& \nu_1 \leq 106.58 - 73\rho && \text{(5.38c)}\\
& \nu_2 \leq -\ln 4 + 9\rho && \text{(5.38d)}\\
& \boldsymbol{\nu} \in \mathbb{R}^2 && \text{(5.38e)}\\
& \rho \in \mathbb{R}_+. && \text{(5.38f)}
\end{aligned}$$

So, at iteration $t = 1$, in Line 5 we obtain the optimal solution $\boldsymbol{\nu}^1 = (99 \ \ -0.4517738)$ and $\rho^1 = 0.1038356$. Solving the subproblems associated to $\xi^1(\rho^1)$ and $\xi^2(\rho^1)$ in Line 7, we obtain the objective values -0.1347729, for $x^{1,1} = 2.5958904$, and -0.6832698, for $x^{2,1} = 2$, respectively. The supergradients of $\xi^1(\rho)$ and $\xi^2(\rho)$ at ρ^1 computed in Line 8 are $-x^{1,1} = -2.5958904$ and $(x^{2,1})^2 = 4$, respectively. This leads to the violated constraints:

$$\begin{aligned}
\nu_1 &\leq -0.1347729 + (\rho - 0.1038356) \times (-2.5958904) && \text{(5.39a)}\\
\nu_2 &\leq -0.6832698 + (\rho - 0.1038356) \times 4, && \text{(5.39b)}
\end{aligned}$$

being added to the LP in Line 10. By repeating this procedure for four more itera-

tions, until $t = 5$, the following additional constraints are added:

$$\nu_1 \leq -28.5736932 + (\rho - 1.5119178) \times (-37.7979452) \tag{5.40a}$$
$$\nu_2 \leq 0 \tag{5.40b}$$
$$\nu_1 \leq -0.9429289 + (\rho - 0.2746531) \times (-6.8663268) \tag{5.40c}$$
$$\nu_2 \leq -0.4184941 + (\rho - 0.2746531) \times 1 \tag{5.40d}$$
$$\nu_1 \leq -0.228336 + (\rho - 0.135155) \times (-3.3788759), \tag{5.40e}$$

in this order. Two constraints were added in the second iteration, two in the third, and only one in the fourth. In the last iteration, no violated cut was found. The optimal solution for the final LP is $\boldsymbol{\nu}^5 = (-0.228336 \;\; -0.5579921)$ and $\rho^5 = 0.135155$, and the LR bound is $z_{\text{LD}} = L(\rho^5) = -1.7497854$. An optimal solution to the original mixed-integer non-linear problem (5.36) is $\boldsymbol{x} = (2 \;\; 1 \;\; 2)$, with objective value -1.7117595. By relaxing the integrality constraints in (5.36) and solving the resulting continuous non-linear problem, a lower bound of -1.8319819 would be obtained. The bound z_{LD} is stronger because the integrality is not relaxed in the subproblems.

Figure 5.6 depicts the LDF $L(\rho)$. A solution $\boldsymbol{x}$ of the LS (5.37) yields the following line:

$$z \leq \frac{x_1^2}{50} - x_1\,\rho - \ln(x_2 + 1) + x_2^2\,\rho - \ln(x_3 + 1) + x_3^2\,\rho - 3\,\rho, \tag{5.41}$$

which may be part of the outer approximation of the hypograph of $L(\rho)$. As done in the numerical example of Subsection 5.2.2, some of those lines are labeled by the solutions that define them. Observe in Figure 5.6 that the two lines that are active at the optimal solution are $z \leq -1.1579583 - 4.3788759 \times \rho$, and $z \leq -1.9688885 + 1.6211241 \times \rho$, associated with the LS solutions $(3.3788759 \;\; 1 \;\; 1)$, and $(3.3788759 \;\; 2 \;\; 2)$, respectively. Note that those two solutions are combinations of subproblem solutions found in different iterations. While $x_1 = 3.3788759$ has been found in the fourth iteration, $x_2 = x_3 = 1$ was found in the third, and $x_2 = x_3 = 2$ in the first. This illustrates again the "magic of multiple subproblems".

5.4. Two Examples

We illustrate the use of LR on exact algorithms for LCOPs with two examples.

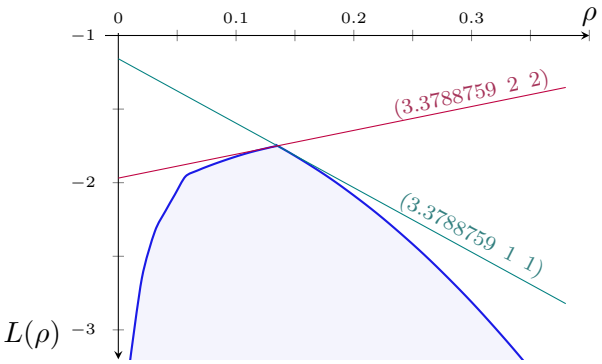


Figure 5.6: The LDF $L(\rho)$ corresponding to LS (5.37). The two lines (obtained by the cutting plane algorithm as combinations of partial solutions found in different iterations) that define the optimal solution are shown.

5.4.1. Traveling Salesperson Problem

Consider a TSP defined over a graph $G = (V, E)$ with vertex-set $V = [n]$, 1 representing an arbitrary vertex, and the following formulation, which is equivalent to (3.5):

$$\min \quad \sum_{e \in E} c_e x_e \tag{5.42a}$$

$$\text{s.t.} \quad \sum_{e \in \delta(i)} x_e = 2 \qquad i \in V \setminus \{1\} \tag{5.42b}$$

$$\sum_{e \in \delta(1)} x_e = 2 \tag{5.42c}$$

$$\sum_{e \in E(V \setminus \{1\})} x_e = n - 2 \tag{5.42d}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad S \subseteq V \setminus \{1\} \tag{5.42e}$$

$$\boldsymbol{x} \in \mathbb{B}^{|E|}. \tag{5.42f}$$

Equality (5.42d) is redundant, as it can be obtained by summing all constraints in (5.42b), subtracting the equality (5.42c), and then dividing the result by two. Subtour elimination inequalities in format (5.42e) are equivalent to inequalities (3.5c) (see Note 3.13). Held and Karp [1970] applied a DW decomposition to (5.42), keeping (5.42b) in the master. Let Q be the set of subproblem solutions, the resulting

MLP is:

$$z_{\mathrm{M}} = \min \quad \sum_{\boldsymbol{q}\in Q}\Big(\sum_{e\in E} c_e q_e\Big)\lambda_{\boldsymbol{q}} \tag{5.43a}$$

$$\text{s.t.} \quad \sum_{\boldsymbol{q}\in Q}\Big(\sum_{e\in\delta(i)} q_e\Big)\lambda_{\boldsymbol{q}} = 2 \qquad i \in V\setminus\{1\} \tag{5.43b}$$

$$\sum_{\boldsymbol{q}\in Q}\lambda_{\boldsymbol{q}} = 1 \tag{5.43c}$$

$$\boldsymbol{\lambda} \geq \mathbf{0}. \tag{5.43d}$$

Let $\boldsymbol{\pi}$ and ν be the dual variables corresponding to (5.43b) and (5.43c), the pricing subproblem has format:

$$\bar{c}^* = \min \quad \sum_{e=\{i,j\}\in E} (c_e - \pi_i^* - \pi_j^*)x_e - \nu^* \tag{5.44a}$$

$$\text{s.t.} \quad \sum_{e\in\delta(1)} x_e = 2 \tag{5.44b}$$

$$\sum_{e\in E(V\setminus\{1\})} x_e = n-2 \tag{5.44c}$$

$$\sum_{e\in E(S)} x_e \leq |S|-1 \qquad S\subseteq V\setminus\{1\} \tag{5.44d}$$

$$0 \leq x_e \leq 1 \qquad e\in E \tag{5.44e}$$

$$\boldsymbol{x} \in \mathbb{Z}^{|E|}. \tag{5.44f}$$

Held and Karp realized that the solutions of (5.44) correspond to what they called *1-trees*: a spanning tree of the subgraph induced by $V\setminus\{1\}$ plus two edges adjacent to vertex 1 (all tours are 1-trees but most 1-trees are not tours). Therefore, despite having an exponential number of constraints, the pricing subproblems can be easily solved! It suffices to compute a minimum cost spanning tree in $V\setminus\{1\}$ (implementations of either Borůvka, Prim-Jarník or Kruskal algorithms can run in $O(|E|.\log|V|)$ time) and then add the two cheapest edges adjacent to 1, in both cases using the modified costs in (5.44a).

That discovery had the potential to result in the first BPA ever. Let us consider that historical moment. The concepts of Branch-and-Cut and separation algorithms were not yet well-established. Dantzig et al. [1954] separated subtour elimination constraints in format (3.5c) manually, looking at fractional solutions drawn in a

paper. Solving (5.43) by CG would provide the first fully automated way of obtaining a strong bound z_{M} at least as good as the Dantzig-Fulkerson-Johnson Subtour Bound. By the way, after Edmonds [1971] proved that (5.44c)–(5.44e) is a perfect description of the spanning tree polyhedron, it can be shown that the subproblem has the integrality property, and therefore, by Theorem 4.1, those bounds are identical. However, the CGA implemented in Held and Karp [1970] to solve (5.43) yielded frankly disappointing results: *"The program was able to solve most problems with* $n = 12$ *and some problems with* $13 \leq n \leq 20$. *On larger problems, convergence was always too slow to permit optimal solutions to be reached. This slow convergence is consistent with the behavior of other column-generation techniques."* Remember that Dantzig et al. [1954] could solve an instance with $n = 49$ using much inferior computational resources.

The breakthrough in the exact solution of the TSP would only happen in Held and Karp [1971], when the authors dropped CG and replaced it with LR. For fixed multipliers $\boldsymbol{\pi} = (\pi_1 \dots \pi_n)$, they define the following Lagrangian Subproblem:

$$L(\boldsymbol{\pi}) = \min \quad \sum_{e=\{i,j\}\in E} (c_e - \pi_i - \pi_j)x_e - 2\sum_{i\in V} \pi_i \tag{5.45a}$$

$$\text{s.t.} \quad \boldsymbol{x} \in Q, \tag{5.45b}$$

where Q is the set containing the incidence vectors of all 1-trees. Multiplier π_1, corresponding to dualizing a constraint (5.42c) that is also kept in the subproblem, is not really necessary, as all 1-trees have degree two at vertex 1. However, the authors possibly preferred to have multipliers for all vertices to make things more symmetrical. The proposed method uses the Subgradient Algorithm with step size (5.29) for obtaining a near-optimal solution to the LDP $z_{\mathrm{LD}} = \max_{\boldsymbol{\pi}} L(\boldsymbol{\pi})$, reaching a bound z_{root} close to $z_{\mathrm{LD}} = z_{\mathrm{M}}$. The calculation of a supergradient in Line 5 of Algorithm 3 becomes $\boldsymbol{s} \leftarrow \boldsymbol{g}(\boldsymbol{x}^t) = ((\sum_{e\in\delta(1)} x_e^t - 2) \dots (\sum_{e\in\delta(n)} x_e^t - 2))$.

Finally, they devised a Branch-and-Bound using those strong Lagrangian bounds to obtain a complete TSP method. Remark that the "dual" Subgradient Algorithm does not readily provide a primal fractional solution $\boldsymbol{x}^*$ to the linear relaxation of (5.42) that can be used for choosing the branching variable (the issue of recovering primal fractional solutions in Lagrangian methods is discussed in Note 5.2). Their proposed branching scheme can be outlined as follows. Let $\boldsymbol{\pi}'$ be the best multiplier vector at the current node and $\boldsymbol{x}'$ be the corresponding 1-tree incidence vector.

For each still non-fixed edge e, they evaluate the conditional bound obtained with multipliers $\boldsymbol{\pi}'$ but assuming that $x_e = 0$. Due to the favorable properties of the minimum spanning tree problem, this can be efficiently done. Actually, for each edge e such that $x'_e = 0$ this is trivial, as no bound improvement is possible. Anyway, the edges are sorted in non-decreasing conditional bound order as $e_1, e_2, \ldots, e_p$. Then, the first child node imposes $x_{e_1} = 0$, the second imposes $x_{e_1} = 1, x_{e_2} = 0$, the third $x_{e_1} = 1, x_{e_2} = 1, x_{e_3} = 0$, and so on. The scheme worked fine in the tested instances because almost all those children nodes could be immediately pruned.

The authors of Held and Karp [1971] were quite enthusiastic, as indicated by their abstract: *"A branch-and-bound procedure based upon these considerations has easily produced proven optimum solutions to all traveling-salesman problems presented to it, ranging in size up to sixty-four cities. The bounds used are so sharp that the search trees are minuscule compared to those normally encountered in combinatorial problems of this type."* Indeed, that seminal article inspired the first wave of LR-based methods for LCOPs (and discouraged CG!) in the 1970s. Some remarks:

- As will be discussed in Section 5.5, now *we know* why the Subgradient Algorithm was much better than CG in solving essentially the same problem: approximating the Subtour Bound for the TSP by pricing 1-trees. In reality, that problem gathers all the known features favorable to subgradient methods and unfavorable to the standard CG (standard in the sense of not being enhanced by advanced stabilization methods, as described in Chapter 7).

- The Held-Karp method for the TSP (including its long sequence of enhanced versions, starting with Helbig Hansen and Krarup [1974]) was largely supplanted by BCAs that can not only obtain the exact Subtour Bound by separating those constraints in a very efficient way but indeed obtain almost incredible tighter bounds by also separating several other families of cuts, as described in Section 3.4.1 and in Note 3.15. Actually, the HK method survives in niche applications as a lightweight stand-alone code (not requiring an LP solver) that can quickly solve small TSP instances.

A very didactic presentation of the HK method can be found in Cook et al. [1998].

5.4.2. Generalized Assignment Problem

The GAP (Definition 4.3) was used in Section 4.4.1 to illustrate the BP and BCP algorithms. Note 4.8 is a short survey of existing GAP exact solvers. This section presents the LR-based method in Posta et al. [2012].

Consider the Formulation (4.16). By dualizing the assignment Constraints (4.16b), the LS problem can be decomposed into K subproblems. Given a fixed multiplier vector $\boldsymbol{\pi} = (\pi_1 \ \ldots \ \pi_J)$, the Binary Knapsack subproblem corresponding to machine $k \in [K]$ is:

$$\xi^{k*}(\boldsymbol{\pi}) = \min \sum_{j \in [J]} (c_j^k - \pi_j) x_j^k \tag{5.46a}$$

$$\text{s.t.} \sum_{j \in [J]} w_j^k x_j^k \leq W_k \tag{5.46b}$$

$$\boldsymbol{x}^k \in \{0,1\}^J. \tag{5.46c}$$

The LDP is $z_{\text{LD}} = \max \ L(\boldsymbol{\pi}) = \sum_{k \in [K]} \xi^{k*}(\boldsymbol{\pi}) + \sum_{j \in [J]} \pi_j$ s.t. $\boldsymbol{\pi} \in \mathbb{R}^{1 \times |J|}$. The optimal Lagrangian bound z_{LD} is identical to the strong bound z_{M} obtainable by solving (4.18) by CG. The optimal solutions $\boldsymbol{x}^{k*}$ of subproblems (5.46) provide a supergradient $\boldsymbol{s} = ((\sum_{k \in [k]} x_1^{k*} - 1) \ \ldots \ (\sum_{k \in [k]} x_J^{k*} - 1))$ of the LDF at point $\boldsymbol{\pi}$. Up to here, there is no novelty; Fisher [1981] already mentions three works that used such LR scheme for the GAP. However, some enhancements proposed in Posta et al. [2012] led to a method that still obtains some of the best results for this problem:

- At the root node, an LP solver quickly obtains the optimal dual solution of the linear relaxation of (4.16), which is used to hot-start the multiplier vector $\boldsymbol{\pi}$. Then, algorithm [B]TT/OBP [Frangioni, 1996] is used to obtain a near-optimal LDP solution $\boldsymbol{\pi}^{\text{r}}$ yielding a bound $z_{\text{root}} = L(\boldsymbol{\pi}^{\text{r}})$ very close to z_{LD}. This Bundle-type algorithm worked much better than the Subgradient algorithms employed by previous authors.

- All GAP instances from the literature have integer costs, so z_{IP} is known beforehand to be an integer. Moreover, those costs are relatively small numbers, meaning that the strong bound z_{root} implies an absolute gap $z_{\text{IP}} - z_{\text{root}}$ of only a few units. The method exploits that and proceeds by running a sequence of BBAs that try to find a solution with tentative integer values $\bar{z}$, starting with $\bar{z} = \lceil z_{\text{root}} \rceil$. In any of those runs, a node S with bound $z_S > \bar{z}$ is pruned. If a

BBA finds any integer solution with value $\bar{z}$, that should be an optimal GAP solution and the optimization finishes immediately. Otherwise, if a BBA run ends without finding a solution, it is proven that solutions with value $\bar{z}$ do not exist. In that case, a new BBA with $\bar{z}$ incremented by one unit is started. Of course, its root node already begins from $\boldsymbol{\pi}^{\mathrm{r}}$. For example, on instance D-5-100, $\lceil z_{\mathrm{root}} \rceil = 6350$, so the first three BBA runs prove that solutions with values 6350, 6351, and 6352 do not exist; and only the fourth BBA run finds the optimal solution with value $z_{\mathrm{IP}} = 6353$.

The approach might seem inefficient due to "wasted" runs where $\bar{z} < z_{\mathrm{IP}}$, but this is not the case. The number of nodes in a BBA run grows more than linearly (exponentially, indeed) with the difference $\bar{z} - z_{\mathrm{root}}$. Thus, if the run with $\bar{z} = z_{\mathrm{IP}}$ takes a reasonable time, the runs with $\bar{z} < z_{\mathrm{IP}}$ are certainly manageable. The key advantage is that the critical run with $\bar{z} = z_{\mathrm{IP}}$ (equivalent to using an external upper bound $UB = z_{\mathrm{IP}} + 1$) can be much faster than a BBA run without any external upper bound. This potential benefit is further amplified by the variable fixing technique described next.

- A crucial improvement was the fixing of variables by conditional Lagrangian bounds or Lagrangian reduced costs, as the authors called it. Let $\xi^{k*}(\boldsymbol{\pi}, j)$, $j \in [J]$, be the value of an optimal solution of (5.46) with the additional constraint $x_j^k = 1$. The difference $\bar{c}_j^k(\boldsymbol{\pi}) = \xi^{k*}(\boldsymbol{\pi}, j) - \xi^{k*}(\boldsymbol{\pi})$ is the "Lagrangian reduced cost" of variable x_j^k with respect to the multipliers $\boldsymbol{\pi}$. It can be used for fixing variables to zero. For example, consider the root node near-optimal multipliers $\boldsymbol{\pi}^{\mathrm{r}}$ which obtained $z_{\mathrm{root}} = L(\boldsymbol{\pi}^{\mathrm{r}})$. If $z_{\mathrm{root}} + \bar{c}_j^k(\boldsymbol{\pi}^{\mathrm{r}}) > \bar{z}$ then job j can not be assigned to machine k in any solution that costs $\bar{z}$ or less. A similar reasoning may be used for fixing variables to one. Fixing may be performed on all BBA nodes. Variables fixed at a certain node S remain fixed for the subtree rooted at S.

  The naive way of computing $\xi^{k*}(\boldsymbol{\pi}, j)$ values by solving Binary Knapsack problems, one for each $j \in [J]$, would be too time-consuming. Instead, the authors used the method proposed in Karabakal et al. [1992] for computing all those J values in $O(JW_k)$ time by Dynamic Programming. The standard Binary Knapsack DP is run twice, first using the forward recursion and a second time using the backward recursion. Then, the values $\xi^{k*}(\boldsymbol{\pi}, j)$ are readily obtained by concatenating partial forward and backward solutions. "Smart fixing tech-

niques", like the ones proposed in Tanaka et al. [2009], Irnich et al. [2010], Pessoa et al. [2010], Pecin et al. [2017b], Sadykov et al. [2021], Bianchessi et al. [2024], Yang [2024], described in depth in Chapter 11, play an important role in advanced BCPAs for problems like vehicle routing or machine scheduling.

- The LR-method in Posta et al. [2012] adopts a very simple branching scheme: select the job j with the largest multiplier π_j such that not all x_j^k variables are fixed, and branch on each unfixed machine, thus creating up to K children nodes. The Binary Knapsack subproblems (excepting the special DP runs for computing reduced costs) are solved using Pisinger's MINKNAP code (see Note 4.6). Algorithm [B]TT/OBP is restricted to 100 iterations on non-root nodes.

Some computational results in Pigatti et al. [2005], Avella et al. [2010], Posta et al. [2012], Pessoa et al. [2020] are reproduced in Table 5.1, the reported times (in seconds) are the original ones, obtained in different processors. Considering those differences and also results in many other instances not reproduced here, it can be concluded that the LR-based method in Posta et al. [2012] often obtains the best performance. However, it is interesting that the CG-based methods, even the simple BPA in Pigatti et al. [2005], can be better in instances with few jobs per machine, like D-20-100, for reasons that will be explained in the next section.

Table 5.1: Comparison of times (in seconds) on selected GAP instances

Instance	z_{LP}	z_{M}	z_{IP}	Pigatti (2005)	Avella (2010)	Posta (2012)	Pessoa (2020)
C-10-200	2795.41	2803.95	2806	266	2.90	2.28	28.6
C-20-200	2376.91	2390.17	2391	55.4	1.80	0.17	14.6
C-20-400	4774.15	4780.18	4782		21.9	9.79	299
D-5-100	6345.41	6349.92	6353	96.3	13.3	1.28	10.7
D-10-100	6323.46	6341.45	6347	818	385	13.4	23.6
D-20-100	6142.53	6176.14	6185	1043	27783	2425	239
D-25-90-e1	5566.11	5617.96	5627				49.7

Source: Pigatti et al. [2005], Avella et al. [2010], Posta et al. [2012], and Pessoa et al. [2020].

5.5. Assessment of Lagrangian Relaxation vs Column Generation

If LR and CG are closely related techniques, capable of obtaining the same bounds by solving the same subproblems, when should one be preferred over the other in an algorithm for a particular LCOP? By using LR we mean that multipliers are adjusted using "Lagrangian methods" [3], ranging from the basic Subgradient Algorithm to highly sophisticated algorithms for non-smooth convex/concave optimization. These methods typically terminate at near-optimality and do not provide an accurate primal fractional solution (Note 5.2). In contrast, by using CG we mean that multipliers/dual variables are obtained by solving RMLPs. CG is often performed until optimality. But even if it is stopped a bit earlier, an accurate primal fractional solution is a by-product.

During the 1970s and the 1980s, LR was far more popular than CG as a tool for obtaining strong lower bounds for LCOPs. One explanation may be that not everyone at that time had access to high-quality linear programming packages to solve RMLPs. In contrast, basic LR methods, like the Subgradient Algorithm, could be implemented in a few dozen lines of FORTRAN code. Another explanation may be the influence of the works by Held and Karp on the TSP, which obtained incomparably better results with LR. The reasons for that particular CG failure were poorly understood, which led to widespread skepticism over general CG.

Today, with the benefit of more than five decades of research on both techniques, we can identify three key factors to consider when deciding between LR and CG:

1. **Number of Subproblems**. Situations where the LS does not decompose into multiple (distinct or identical) subproblems are problematic for CG convergence and possibly much more suited to LR methods. More generally, having

[3] Lemaréchal [2001] laments that CG-based algorithms are not usually presented as LR. In his words: *"any formulation in terms of CG can be done in terms of LR (and conversely). We believe that this is important because LR – duality theory — mostly calls for fairly simple concepts. By contrast, CG has to call for the often fussy language of linear programming"*. Indeed, he views CG as nothing other than LDPs solved using the cutting plane algorithm, and therefore, as a LR method. While this observation is technically correct, CG has advanced so much and created so many unique characteristics that it is more productive to consider it a distinct technique. Nevertheless, research on modern Branch-Cut-and-Price algorithms continues to draw heavily from its Lagrangian origins.

only a few subproblems favors LR while having many subproblems is likely good for CG convergence. A partial explanation lies in what we have called the "magic of multiple subproblems" in Section 5.2.2: the subproblem solutions from past iterations stored in the RMLP as generated variables combine into the equivalent of many supergradients (which correspond to constraints in the RLDP), as in the example of Figure 5.4b. On the other hand, standard LR methods concatenate the subproblem solutions of the current iteration to obtain a single supergradient. The number of subproblems also plays a very important role in the next factor.

2. **The magnitude of the number of variables in the MLP.** The CGA is essentially a primal simplex. As mentioned in Note 1.8, the typical (not the worst-case) number of simplex iterations until convergence grows linearly with the number of rows, but much more slowly with the number of columns. Indeed, by solving the pricing step using auxiliary optimization subproblems, CG allows the solution of LPs with a huge number of columns. However, it is still quite expected that enormous differences in the magnitude of $p \equiv \sum_{k \in [K]} |Q^k|$ may lead to significant differences in performance; that the CGA will take less iterations for solving MLPs with "only" $p = 10^{10}$ columns than those with $p = 10^{80}$ [4].

   This is not a speculation. It is a fact observed by CG practitioners over a wide range of applications. For example, consider the CVRP. For instances where the capacities are small with respect to the demands, so routes can not visit, say, more than 10 customers, p is only "moderately huge". In those instances even a standard (i.e., not improved by stabilization techniques) CG has no convergence problems. However, if capacities are large enough to allow routes with a few dozen customers, the number of possible routes becomes "really huge" and the convergence of standard CG is much slower. On routing with time windows, as is known since the 1980s (see Note 4.19), when the time windows are narrow the number of possible routes is only "moderately huge" (even if there are still routes with many customers) and CG is fast; when they are wide CG can be slow.

   To make things more concrete we report an experiment in Pigatti et al. [2005]

[4] This figure matches the estimated number of atoms in the observable universe.

with a standard non-stabilized CGA (RMLP initialized with artificial variables, exact pricing in all iterations using MINKNAP) on finding the exact bound z_{M} on six GAP instances. Those D-class instances are very tight and each of the K machines should receive about the same fraction of the J jobs in a feasible solution. The number of possible columns having exactly J/K jobs is $K\binom{J}{J/K}$. We include that expression (which does not appear in Pigatti et al. [2005]) in Table 5.2 as a gross indicator of the relative differences in the magnitude of p in each MLP. As can be seen, the magnitude of $K\binom{J}{J/K}$ (or simply the number J/K) predicts well whether the unstabilized CG will have convergence issues or not. The number J/K is also an indicator of the average number of non-zeros in the RMLP columns. Sparse RMLPs are solved faster than dense ones, a fact that also has some influence on total CG time.

Table 5.2: The order of magnitude of the number of columns in the MLP helps to predict the number of iterations in a standard CGA for finding the exact z_{M}.

Instance	J/K	$K\binom{J}{J/K}$	z_{M}	Unstabilized CG		
				iters	cols	time (s)
D-20-100	5	1.5×10^{9}	6176.14	77	1469	0.79
D-10-100	10	1.7×10^{14}	6341.45	168	1587	1.98
D-20-200	10	4.4×10^{17}	12229.66	224	4260	26.12
D-5-100	20	2.7×10^{21}	6349.92	678	3156	18.71
D-10-200	20	1.6×10^{28}	12425.61	839	7239	181.25
D-5-200	40	1.0×10^{43}	12740.04	4139	19007	6610.31

Source: Pigatti et al. [2005].

In contrast, LR methods do not seem to be affected by a "really huge" number of LS solutions. Keep in mind that Lagrangian multipliers were first devised for non-linear differentiable problems where the number of LS solutions is infinite! Let us consider the LR TSP method in Held and Karp [1971]. The largest solved instance had $n = 64$ cities, leading to $63!/2 = 10^{87}$ possible tours and even more 1-trees (all tours are 1-trees). Of course, the vast majority of those 1-trees are dominated, in the sense of not being optimal solutions of the LS for any multiplier $\boldsymbol{\pi}$. However, the non-dominated solutions are still so numerous that they define a piecewise linear LDF that is "almost differentiable", something similar to what is schematically depicted in Figure 5.7a. In such

situations, LR methods, in the Held-Karp case even the basic Subgradient Algorithm, can work much better than CG. On the other hand, situations where the LSs are defined by a not-so-huge number of non-dominated solutions, as schematically depicted in Figure 5.7b, are better for CG. Note that we do not know how to even estimate the number of non-dominated solutions (indeed, many columns in a MLP can never appear in a RMLP, unless heuristic pricing is used, because the associated solutions are never optimal in the pricing). This is why we use p as a proxy in our guidelines.

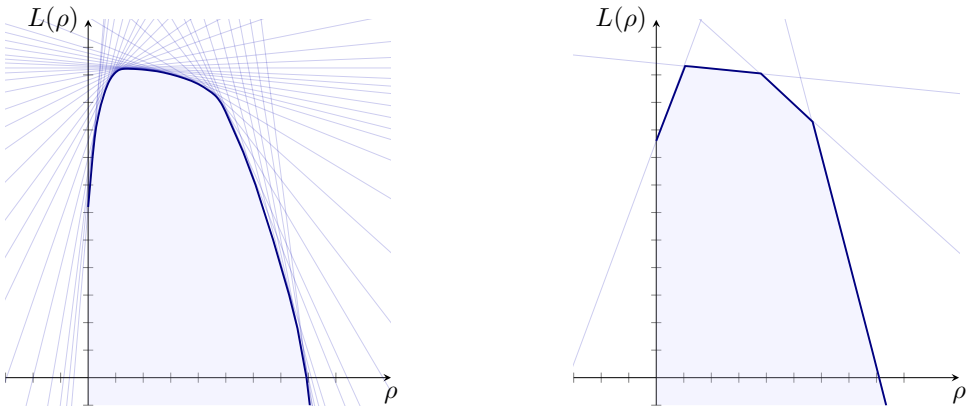
By the way, the fact that CG approaches suffer when the number of columns is enormous is consistent with the consensus that on typical non-linear differentiable problems, the cutting plane algorithms (which in those cases are equivalent to a CG over an MLP having infinite columns) are usually not the best option. The exceptions are cases where the subproblem can be decomposed into many parts, so the cutting plane method may profit from the "magic of multiple subproblems". An example of that situation can be found in Aroztegui and Pessoa [2024].

3. **Strength of the bound.** Having fewer subproblems and LSs formed by an enormous number of linear pieces makes CG slow. The convergence problem can be mitigated by dual stabilization techniques, but not eliminated. That would lead to preferring LR methods in many situations, perhaps even in the majority of cases. However, there is a third factor that may tip the scales in favor of CG: the strength of the bound $z_{\text{LD}} = z_{\text{M}}$.

   Suppose that those bounds are excellent, so the gap $z_{\text{IP}} - z_{LD}$ can be closed by a LR-based BBA (possibly together with some Lagrangian reduced cost fixing scheme) with a tree search of moderate size. In such situations, LR is often the best choice. As examples, we have the TSP method by Held and Karp [1971] *on not-so-large instances* and the GAP method by Posta et al. [2012] on most instances.

   But what to do if the gap is larger and can not be closed in a reasonable time? The most standard way of improving a bound is by adding cuts. This is complicated in a LR context. Let us first consider robust cuts defined over the original variables. Separating such cuts requires good approximations of a primal fractional solution, something that is not so easy to obtain in a LR context (Note 5.2). The alternative of separating over integral LS solutions,

in the so-called Relax-and-Cut methods, has its limitations (Note 5.3). But what about non-robust cutting over the generated variables? As far as we know, there is nothing equivalent to that in LR. For example, consider the CVRP bounds z_{M} obtained by CG from solving (4.29). Those bounds are not so good, yielding typical gaps of around 3-5%. LR methods would be faster than CG in approximating those bounds on instances with long routes, but the resulting LR-based BBA algorithm (similar to the one in Christofides et al. [1981]) would not be competitive. Advanced BCPAs for the CVRP use both robust and non-robust cuts to obtain much smaller root gaps to solve much larger instances.



(a) Good for LR.

(b) Good for CG.

Figure 5.7: Impact of the number of linear pieces in a LDF.

5.6. Case Study: Parallel Machine Scheduling

Scheduling refers to problems where the goal is to allocate resources (like machines) over time in order to execute tasks in the most efficient way. These problems are prevalent in various industries and disciplines, including manufacturing, project management, and computer science. This section describes some progress brought

by both CG-based and LR-based methods in the exact algorithms for solving the following family of scheduling problems:

Definition 5.1: Parallel Machine Scheduling Problem (PMSP). Instance: J jobs and K identical machines; integer processing times p_j and cost functions $f_j(C_j)$ over job completion time, $j \in [J]$. Solutions: scheduling of jobs in the machines, perhaps introducing idle times between jobs. Each machine can process without preemption at most one job at a time. Goal: minimize total scheduling costs.

The generality of the cost function permits viewing many classical scheduling problems as particular cases of the PMSP. In fact, it includes any single machine or parallel identical machines problem where the cost function is based on penalties for job earliness or lateness (possibly including an infinity penalty for a job started before its release date or finished after its deadline). For a typical example, in the weighted-tardiness scheduling problem, each job $j \in [J]$ has a due date d_j and a weight w_j, and the cost function of job j is defined as $f(C_j) = w_j \max\{0, C_j - d_j\}$. In the three-field notation [Graham et al., 1979], this problem is referred to as $1||\sum w_j T_j$ for the single machine case (already strongly $\mathcal{NP}$-hard) and as $P||\sum w_j T_j$ for the parallel identical machines case. Compact formulations for the PMSP use binary variables to indicate job sequence and continuous variables to represent completion times. The big-M constraints linking those variables make them notoriously weak. Many instances with only 20 jobs may not be solved by a general MIP solver.

Much stronger pseudo-polynomial formulations do exist. Assume that there is an optimal solution where all jobs are processed in a time horizon ranging from 0 to T. If the cost functions are monotonically non-decreasing (later job completion is never better), there are optimal solutions without idle times between jobs in the same machine. In those cases, which include weighted tardiness functions, T can be set as $\lfloor(\sum_{j\in[J]} p_j - p_{max})/K\rfloor + p_{max}$, where p_{max} is the maximum processing time of a job. The time-indexed formulation proposed in Dyer and Wolsey [1990] for the single machine case (other time-indexed formulations for other scheduling problems exist since Bowman [1959]) has $O(JT)$ binary variables corresponding to all possible completion times of each job. We present here the version of the time-indexed formulation where its variables can be interpreted as arc flows. Let

$G = (V, A)$ be a directed multigraph having vertex-set $V = \{0, \ldots, T\}$ and arc-set $A = \cup_{j \in [J]} A_j \cup A_0 \cup (T, 0, 0)$, where $A_j = \{(t - p_j, t, j) \,|\, p_j \leq t \leq T\}$ and $A_0 = \{(t - 1, t, 0) \,|\, 1 \leq t \leq T\}$. This interpretation makes the time-indexed formulation similar to Valério de Carvalho's CSP formulation. Indeed, the graph G for an scheduling instance with $J = 2$, $p_1 = 2$, $p_2 = 3$, and $T = 5$ is isomorphic to the graph in Figure 4.11. However, if different jobs have the same processing time then G will have parallel arcs. The formulation is:

$$\min \quad z = \sum_{a=(t-p_j,t,j) \in A,\, j \neq 0} f_j(t) x_a \tag{5.47a}$$

$$\text{s.t.} \quad \sum_{a \in \delta^-(v)} x_a - \sum_{a \in \delta^+(v)} x_a = 0 \qquad v \in V \tag{5.47b}$$

$$\sum_{a \in A_j} x_a = 1 \qquad j \in [J] \tag{5.47c}$$

$$x_{(T,0,0)} = K \tag{5.47d}$$

$$\boldsymbol{x} \in \mathbb{Z}_+^{|A|}. \tag{5.47e}$$

A variable x_a where $a = (t - p_j, t, j) \in A_j$ can only assume values in $\mathbb{B}$ and indicates whether job j finishes at time t. A variable x_a where $a = (t - 1, t, 0) \in A_0$ represents how many machines are idle between time $t - 1$ and t. The return flow variable $x_{(T,0,0)}$ is fixed to ensure that the solution can be decomposed into K paths between vertices 0 and T, and each such path corresponds to the schedule of a machine.

van den Akker et al. [2000] proposed a CG approach for single machine scheduling. A DW decomposition of (5.47) that keeps (5.47c) and (5.47d) in the master yields a subproblem that has a network flow structure. Then, its path+cycle decomposition results in a formulation where there is a variable for each $0-T$-path in G. Those paths were called *pseudo-schedules* because, as Constraints (5.47c) were relaxed, some jobs may appear more than once. Let Q be the set of all pseudo-schedules, represented as J-dimensional vectors. This means that q_j indicates how many times job j appears in the pseudo-schedule corresponding to $\boldsymbol{q} \in Q$. The

formulation is:

$$\min \quad \sum_{\boldsymbol{q} \in Q} \lambda_{\boldsymbol{q}} \tag{5.48a}$$

$$\text{s.t.} \quad \sum_{\boldsymbol{q} \in Q} q_j \lambda_{\boldsymbol{q}} = 1 \qquad j \in J \tag{5.48b}$$

$$\boldsymbol{\lambda} \in \mathbb{Z}_+^{|Q|}. \tag{5.48c}$$

Its linear relaxation can be solved by the CGA, the pricing subproblem being a Shortest Path Problem over an acyclic graph. Tests on $1|r_j|\sum w_j C_j$ (a variant that includes release dates and uses the weighted completion time objective function) instances with 20 and 30 jobs have shown that, despite convergence difficulties, when processing times were larger, the CGA could be much better than simplex or interior-point algorithms at solving the linear relaxation of the time-indexed formulation.

This work is also one of the pioneers (see Note 4.20) on combining CG with the separation of robust cuts defined over the original formulation, fully describing the mechanism presented in Section 4.3.2. In their case, the cuts to reinforce (5.47) were those already used in van den Akker et al. [1999]. However, the authors were not satisfied with the outcome: *"These results are obviously disappointing. We have to pay a high price for improving the quality of the lower bounds: the computation times increase significantly (much more so than in standard cutting plane algorithms). Reoptimizing the linear program after a set of cuts has been added seems to be almost as hard as solving the original LP (...) If this observation holds in other contexts as well, this constitutes a major computational drawback of combined column and cut generation approaches."* As discussed in Section 5.5, nowadays we have a much better understanding of why that particular CG was so prone to slow convergence and know ways of mitigating the problem.

Subsequent works also based on the time-indexed formulation produced important advances. Avella et al. [2005] applied LR to it, obtaining better results. Later, Pan and Shi [2007] showed that the time-indexed bound can be exactly computed by solving a cleverly crafted transportation problem, while Bigras et al. [2008] proposed a CG scheme that improves upon van den Akker et al. [2000] by the use of a temporal decomposition to improve convergence. The first use of the time-indexed formulation in an exact algorithm for parallel machines was in Tanaka and Araki

[2008], where a LR-based BBA for $P||\sum T_j$ was proposed. Note that the time-indexed bound may still leave a significant gap and all exact algorithms based on it may need to explore large enumeration trees. At that time, single machine instances with 100 jobs and multiple machine instances with 25 jobs could still be challenging.

However, the breakthrough was soon to be obtained. Sourd [2009], Tanaka et al. [2009] and Pessoa et al. [2010] independently proposed an even larger pseudo-polynomial formulation having $O(J^2T)$ variables. Define a directed simple graph $G = (V, A)$, where $V = \cup_{j\in[J]} V_j \cup V_0$ and $A = (\cup_{i,j\in[J],i\neq j} A_{ij}) \cup (\cup_{j\in[J]} A_{0j}) \cup (\cup_{i\in[J]} A_{i0}) \cup A_{00} \cup \{((0,T),(0,-1))\}$. For each $j \in [J]$, $V_j = \{(j,t) \,|\, 0 \leq t \leq T - p_j\}$, while $V_0 = \{(0,t) \,|\, -1 \leq t \leq T\}$. For each $i, j \in J$ such that $i \neq j$, $A_{ij} = \{((i, t - p_i), (j,t)) \,|\, p_i \leq t \leq T - p_j\}$; for $j \in [J]$, $A_{0j} = \{((0, t-1), (j,t)) \,|\, 0 \leq t \leq T - p_j\}$; for $i \in [J]$, $A_{i0} = \{((i, t - p_i), (0,t)) \,|\, p_i \leq t \leq T\}$; $A_{00} = \{((0, t-1), (0,t)) \,|\, 0 \leq t \leq T\}$. The formulation, referred to as the arc-time formulation, is the following:

$$\min \quad z = \sum_{a=((i,t-p_i),(j,t))\in A,\, i\neq 0} f_i(t) x_a \tag{5.49a}$$

$$\text{s.t.} \quad \sum_{a\in\delta^-(v)} x_a - \sum_{a\in\delta^+(v)} x_a = 0 \qquad v \in V \tag{5.49b}$$

$$\sum_{a\in\delta^+(V_i)} x_a = 1 \qquad i \in [J] \tag{5.49c}$$

$$x_{((0,T),(0,-1))} = K \tag{5.49d}$$

$$\boldsymbol{x} \in \mathbb{Z}_+^{|A|}. \tag{5.49e}$$

A variable x_a where $a = ((i, t - p_i), (j,t)) \in A_{ij}$ indicates whether job i finishes at time t and job j immediately follows it in the same machine. Variables corresponding to arcs in A_{0j}, for some $j \in [J]$, indicate machine transitions from idle to executing job j; variables corresponding to arcs in A_{i0}, for some $i \in [J]$, indicate machine transitions executing job i to idle; while variables corresponding to arcs in A_{00} count how many machines that were already idle remained idle. An illustration of the arc-time formulation for an instance with $J = 4$, $K = 2$, $\boldsymbol{p} = (2\ 1\ 2\ 4)$, and $T = 6$ appears in Figure 5.8. The arcs shown correspond to a solution having machine schedules 3-2-0-1 and 4-0-0 (units of idle time are represented by 0).

The above-presented formulation is only slightly stronger than the time-indexed formulation: if arcs of format $((i, t - p_i), (i,t))$ were added to it, they would become equivalent. For example, consider the $1||\sum w_j T_j$ instance with $J = 3$, $\boldsymbol{p} =$

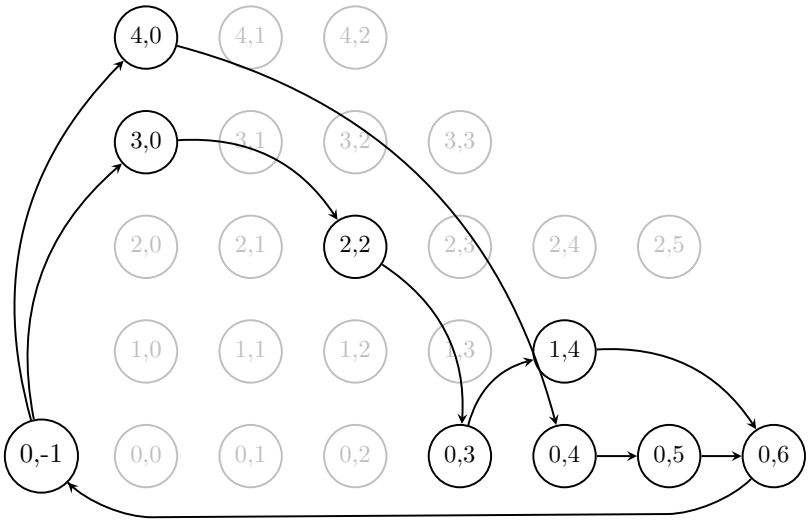


Figure 5.8: Graph for the Arc-Time formulation.

(100 300 200), $\boldsymbol{d}$ = (200 300 400), and $\boldsymbol{w}$ = (6 3 2). Its optimal solution is the schedule 1-2-3 with cost 700. The linear relaxation of the time-indexed formulation yields a fractional solution which is the half-half linear combination of pseudo-schedules 1-1-3-3 and 2-2, with a cost of 650. Pseudo-schedules that repeat the same job immediately are not possible in the arc-time formulation, which would yield the optimal integer solution for that instance. However, consider including additional jobs 4 and 5 with zero weights, arbitrary deadlines, and unit processing times in that instance. The linear relaxation of the arc-time formulation would yield a fractional solution which is the half-half linear combination of pseudo-schedules 1-4-1-3-4-3 and 2-5-2-5, with a cost of 657.5 (the short jobs are being used to circumvent the impossibility of re-executing the same job consecutively). What makes the arc-time formulation worthy of the very significant increase in size are the following results:

Theorem 5.8: *Let i and j be jobs in $[J]$ such that $i < j$ and let $\Delta = (f_i(t) + f_j(t + p_j)) - (f_j(t - p_i + p_j) + f_i(t + p_j))$. If $\Delta > 0$ arc $((i, t - p_i), (j, t))$ can be removed from A; otherwise arc $((j, t - p_i), (i, t - p_i + p_j))$ can be removed.*

Proof. For any given schedule, swapping two consecutive jobs in the same machine does not affect the rest of the scheduling. Therefore, a schedule where j follows

i at time t can be compared with the schedule where i and j are swapped (which means that i follows j at time $t - p_i + p_j$). If $\Delta > 0$ the first schedule is worse than the second, so $((i, t - p_i), (j, t))$ never appears in an optimal solution. Otherwise, the first schedule is at least as good as the second, so there is an optimal solution not using $((j, t - p_i), (i, t - p_i + p_j))$. □

A similar reasoning proves the following:

Theorem 5.9: *Let j be a job in J and let $\Delta = f_j(t) - f_j(t+1)$. If $\Delta > 0$, arc $((j, t - p_j), (0, t))$ can be removed from A; otherwise $((0, t - p_j), (j, t - p_j + 1))$ can be removed.*

Those reductions remove half of the arcs from graph G. However, *the really important gain is drastically reducing the number of paths in G that correspond to pseudo-schedules that are not proper schedules, making the arc-time formulation much stronger*. In the previous example with 5 jobs, the lack of arc $((4, 100)(1, 101))$ eliminates pseudo-schedule 1-4-1-3-4-3 and the lack of arc $((5, 300)(2, 301))$ eliminates 2-5-2-5. Table 5.3 compares the time-indexed and the arc-time gaps over the weighted-tardiness OR-Lib instances, for $K \in \{1, 2, 4\}$ and each $J \in \{40, 50, 100\}$. Each row in that table shows average results over 125 instances having integer processing times uniformly distributed between 1 and 100.

Table 5.3: Comparison of gaps ($1||\sum w_j T_j$ and $P||\sum w_j T_j$)

J	K	Time-indexed	Arc-time	Arc-time + cuts
		Avg. gap (%)	Avg. gap (%)	Avg. gap (%)
40	2	1.533	1.243	0.042
	4	0.544	0.406	0.200
50	2	0.535	0.487	0.090
	4	0.529	0.489	0.274
100	1	0.540	0.023	0.000
	2	1.801	0.688	0.422
	4	0.505	0.493	0.362

Source: Pessoa et al. [2010]

- Looking at the line corresponding to $J = 100$ and $K = 1$, we see something

amazing. The time-indexed formulation produces an average gap of 0.54%, which is quite good but may still lead to big search trees in a BBA. However, the arc-time formulation yields an average gap of only 0.023%! This can be explained. When $K = 1$, the time-indexed formulation only has a positive gap due to the existence of the non-proper pseudo-schedules. When those are nearly eliminated in the arc-time formulation the gap drops to almost zero. To make an analogy with another single subproblem case, the Held-Karp TSP bound only has a positive gap because there are 1-trees that are not proper tours. If one could somehow forbid most of those non-proper 1-trees in the subproblem the gap would drop to almost zero too.

- The gap improvements obtained by the arc-time formulation on instances with $K = 2$ or $K = 4$ are significant but not so dramatic. Using an analogy with another case where there are multiple identical subproblems, this happens for the same reason that makes the CVRP CG bounds not drop to zero if only elementary routes are priced. In multi-machine scheduling, the partial single-machine schedules priced in the arc-time formulation, even if they do not repeat jobs, may combine into fractional solutions with a significant gap. However, the very extended arc-time formulation provides "richer" variables that make it easier to find good robust cuts (Note 3.17). The algorithm in Pessoa et al. [2010] takes advantage of that and separates effective cuts, as also seen in Table 5.3.

Despite the good bounds, the practical use of the very large arc-time formulation depends a lot on efficient fixing by Lagrangian reduced costs, to make its size more tractable. Tanaka et al. [2009], Pessoa et al. [2010] (and also Irnich et al. [2010] in a different context) independently rediscovered the principle that can be found in Karabakal et al. [1992] that such reduced costs can be computed by running the DP recursion twice, first using the forward recursion and a second time using the backward recursion. Then, the reduced costs are readily obtained by concatenating partial forward and backward solutions. In the arc-time formulation, this is particularly efficient: one can compute in $O(J^2T)$ time Lagrangian reduced costs for all the $O(J^2T)$ variables!

Those advances led to two families of algorithms that are still the dominant

exact approaches for the PMSP [5]:

- **Single machine case.** The algorithm in Tanaka et al. [2009] for single machine scheduling without idle times dualizes (5.49c) and solves the resulting LDP by a specially tailored Subgradient Algorithm that has six parameters in its step size calculation. The LSs are Shortest Path Problems over graph G, which can be solved in $O(J^2T)$ by Dynamic Programming. However, the procedure is combined with effective primal heuristics and, due to the extraordinary strength of the arc-time formulation, manages to quickly fix most variables by Lagrangian reduced costs, reducing the time to solve subsequent LSs, which become easier after each iteration. The algorithm does not perform branching. Instead, it closes the very small possible gaps by Successive Sublimation Dynamic Programming (SSDP) [Ibaraki and Nakamura, 1994]. The method detects that some jobs are appearing more than once in some LS solutions and changes the subproblem in order to forbid that. Of course, this creates additional states in the Dynamic Programming, making it harder. However, those increases in complexity turn out to be very manageable because the fixings already made the underlying graph G very sparse. Indeed, stronger subproblems lead to stronger bounds that lead to more fixing that allows even stronger subproblems, and so on.

  Table 5.4 presents the results in Tanaka et al. [2009] for $1||\sum w_jT_j$ instances with up to 300 jobs. All rows are averages over 125 machines. The table provides comparisons with Pan and Shi [2007] and Pessoa et al. [2008] (the technical report version of Pessoa et al. [2010] that includes single machine results obtained by its BCP over the arc-time formulation). The times are in seconds and are taken directly from the sources, without any corrections due to machine speed (as those sources are nearly contemporaneous, those differences are not significant). The results of Tanaka et al. [2009] are spectacular. Note that Pan and Shi [2007] was already a major improvement upon previous exact algorithms. This big "LR success" happened in a context that gathers all the conditions listed in Section 5.5 as favorable to that technique: single sub-

[5] Other families of scheduling problems, in particular those where each job has to be processed at a sequence of machines, like the notoriously hard job-shop scheduling, are better handled by other techniques.

Table 5.4: Single machine results ($1||\sum w_j T_j$)

J	Pan (2007)			Pessoa (2008)			Tanaka (2009)
	Avg. time (s)	Avg. nodes	Root gap (%)	Avg. time (s)	Avg. nodes	Root gap (%)	Avg. time (s)
40	69.0	141	0.68	12.1	1	0	0.19
50	142.8	416	0.74	28.1	1	0	0.39
100	1811	18.8K	0.52	648.5	2.03	0.0013	6.42
150							26.12
200							74.24
250							170.36
300							353.61

Source: Pan and Shi [2007], Pessoa et al. [2008], Tanaka et al. [2009]

problem, enormous number of LS solutions, and very tight Lagrangian bounds (no need for cutting). The approach was extended in Tanaka and Fujikuma [2012] for single machine scheduling with idle times. The authors replace the Subgradient Algorithm with a more sophisticated Conjugate Subgradient. The possibility of idle times makes the problems harder but the resulting LR-based method is still very consistent in solving instances with up 200 jobs in a short time. The source codes of those methods are available at Tanaka [2016].

Tanaka and Araki [2013] deals with single machine scheduling with sequence-dependent setup times. Those problems are significantly harder and the authors introduce the possibility of branching (instead of only doing SSDP) for solving some instances.

- **Multiple machine case.** The BCPA in Pessoa et al. [2010] also works over the arc-time formulation. CG convergence is helped by dual smoothing stabilization [Wentges, 1997] (the authors were not aware of that article and claimed in Pessoa et al. [2008] that they were using an original technique. The mistake was corrected in Pessoa et al. [2010]). Only robust cuts are separated: the Extended Capacity Cuts first proposed in Uchoa et al. [2008]. Those cuts take advantage of the extended arc-time formulation and can not be expressed over the variables of the time-indexed formulation. Fixing by Lagrangian reduced costs is essential to the BCPA performance. However, as the gaps of the arc-time formulation for multiple machines are not as tight, fixing is not so fast

and not so powerful as happens in Tanaka et al. [2009]. Indeed, the proposed method has an alternative that, if at the end of the root node there remains no more than $200K$ arc-time variables, the residual formulation (5.49), plus the separated Extended Capacity Cuts, is given to a MIP solver that finishes the optimization. Sometimes this is faster than performing the full BCPA. The algorithm in Pessoa et al. [2010] could solve all the tested $P||\sum w_j T_j$ and $P||\sum T_j$ instances with up to 50 jobs and having from 2 to 6 machines. It could also solve the majority of the tested $P||\sum w_j T_j$ instances with 100 jobs.

More recently, Oliveira and Pessoa [2020] proposed an improved BCP algorithm that separates additional cuts and is also able to project the arc-time formulation (with most of its variables fixed by reduced costs) onto the time-indexed variable space, to generate more compact IPs to be given to the MIP solver. Bulhões et al. [2020] generalized the arc-time approach and created a BCPA that can handle a wide variety of parallel machine scheduling problems, including situations where job processing times depend on the machine, sequence-dependent setup times, among others. That BCPA also uses non-robust cuts. In all those multiple machine scheduling problems, the current superiority of CG-based approaches over LR-based approaches (assuming that both would solve the same subproblems) is explained by the fact that the known base formulations only provide reasonably good bounds and cutting (robust or non-robust) is needed.

The connection between Lagrangian Relaxation and Column Generation is essential for a deeper understanding of the later technique. In fact, many advanced techniques in CG for improving convergence and for fixing variables draw heavily from that connection. It is also important to be able to recognize the situations where LR methods are likely to outperform CG and may completely replace it.

Notes

5.1. Lagrange multipliers, introduced in the 19th century [Lagrange, 1806], have long been a standard technique for constrained optimization of differentiable functions in calculus. Their application to non-differentiable (or non-smooth) functions is more modern. Naum Z. Shor proposed a subgradient algorithm for solving large-scale dual network transportation problems by reducing them to maximizing piecewise linear concave functions [Shor, 1962]. He and colleagues at the V.M. Glushkov Institute of Cybernetics in Kyiv, Ukraine, applied similar methods with decomposition schemes to various planning problems. Polyak [1978] and Rubinov [2002] describe some of those early Soviet-era developments. Everett III [1963] realized that Lagrangian multipliers could be applied to many LCOPs. The approach gained prominence following Held and Karp [1971] influential work on the TSP. Fisher [1981] already reports numerous successful applications soon after. Interestingly, the term *Lagrangian Relaxation* is quite recent and was coined in Geoffrion [1974]. Beasley [1993] offers an excellent tutorial for beginners, while Guignard [2003] provides a more advanced treatment. The technique is also covered in integer programming textbooks like Wolsey [2020]. There is a vast literature on methods for solving Lagrangian Dual Problems and, more broadly, optimizing non-smooth convex/concave functions. We have no hope of even naming all the *families* of relevant methods and direct readers to surveys and compared studies such as Lemaréchal [2001], Briant et al. [2008], Oliveira and Sagastizábal [2014], Frangioni et al. [2017], and Bragin [2024].

5.2. **Recovering a primal fractional solution**. When DW reformulation and CG are applied to an IP, an optimal fractional solution $\boldsymbol{x}^*$ of its MLP relaxation is readily obtained using (2.9). For example, in problem (5.18) the optimal fractional solution is $\boldsymbol{x}^* = 1/3\,(\,3\;0\,) + 2/3\,(\,1\;1\,) = (\,5/3\;2/3\,)$. In general, an optimal fractional solution may not be unique. Anyway, the knowledge of one such solution is assumed in standard BP and BCP algorithms (as in standard LP-based BB and BC algorithms) for choosing a branching variable and separating cuts.

However, if one solves the equivalent LDP using typical Lagrangian methods the situation is different. Indeed, the vast majority of successful works that applied LR to the exact solution of LCOPs, including Held and Karp [1971], Posta et al. [2012], and all works cited in Fisher [1981] and Beasley [1993], ignore the primal fractional solutions (in the sense of simply not trying to find them) and instead propose branch rules based on $(\boldsymbol{\pi}' \boldsymbol{\rho}')$ and $\boldsymbol{x}'$, which are the multipliers that obtained the best Lagrangian bound of the node and the integer solution of the corresponding LS, respectively. We remark that those multipliers are an approximate LDP solution with $L(\boldsymbol{\pi}', \boldsymbol{\rho}')$ being a bit smaller than the theoretical node bound z_{LD} and therefore $\boldsymbol{x}'$ is a somehow arbitrary solution (it may not even participate in a convex combination of integer solutions that would produce an exact $\boldsymbol{x}^*$) that depends on the chosen stopping criterion. So, there is no guarantee that a branching using only that information will indeed cut an optimal fractional solution in all children nodes. If this does not happen in a certain child node, its final bound can not be larger than the parent theoretical bound z_{LD}. Actually, some of those branch rules include look-ahead mechanisms where several candidate variables are roughly evaluated to avoid too bad choices.

The Volume Algorithm [Barahona and Anbil, 2000] is an advanced Lagrangian method (see Bahiense et al. [2002] for an analysis of it). One of the Volume Algorithm features, already announced in the title of Barahona and Anbil [2000], is that it can produce primal fractional solutions. Some authors like Lemaréchal [2001] and Anstreicher and Wolsey [2009] clarified that it was known for a long time (in "community folklore" but also in Shor [1985] and in Larsson et al. [1999]) that other Lagrangian methods, even the basic Subgradient Algorithm, can produce primal fractional solutions. Indeed, in the the main loop of Algorithm 3, at iteration t, we may compute a weighted average:

$$\hat{\boldsymbol{x}} = \frac{\sum_{k=1}^{t} w_k \boldsymbol{x}^t}{\sum_{k=1}^{t} w_k}, \tag{5.50}$$

where the last iterations receive more weight. It can be proved that even if weights are unitary, under the typical conditions that make the algorithm converge to an optimal LDP solution, $\hat{\boldsymbol{x}}$ converges to an optimal primal solution. This result should be looked at with some care. Even when the current

LDP solution is already near-optimal, $\hat{\boldsymbol{x}}$ may still be a poor approximation of an optimal primal solution. One of the reasons is that the primal integer solutions generated in the early iterations only introduce noise in the estimate (5.50). But even if one disregards the earlier solutions, convergence can still be slow. Consider problem (5.18) and Figure 5.3a. The Subgradient Algorithm will need to evaluate many values of ρ in the vicinity of the optimal $\rho^* = 1/3$ until it slowly "realizes" that point $(1\ 1)$ appears two times more often than point $(3\ 0)$ and so that (5.50) converges to a value very close to $\boldsymbol{x}^* = 1/3\,(3\ 0) + 2/3\,(1\ 1) = (5/3\ 2/3)$. Imagine for a moment how many iterations the Subgradient Algorithm would take until $\hat{\boldsymbol{x}}$ is a good approximation in a large problem where $\boldsymbol{x}^*$ is a combination of dozens of integer solutions! More sophisticated Lagrangian methods, like the Bundle methods or the Volume algorithm itself, sample the vicinity of the optimal LDP solutions more systematically and can produce better primal estimates more quickly. Yet, those methods are usually stopped before those primal estimates have become very precise.

The question is: are precise primal fractional solutions really necessary in an exact LR-based algorithm?

- For the sake of branching the answer is "no". As discussed in Note 3.10, the actual fractional value of a variable (as long as the variable is indeed fractional!) is a very poor predictor of its performance as a branching candidate. Suppose that one branches over a variable x_j such that $\hat{x}_j = 0.5$ but $x_j^* = 0.4$. That difference is likely to be irrelevant due to the two-sided nature of branching: the improvement in the $\leq$ child node may be smaller than expected but this is compensated by a larger than expected improvement in the $\geq$ child node. To improve branch quality it is much better to invest in look-ahead or even full strong branching mechanisms than in obtaining accurate primal solutions.
- For the sake of cutting the answer is "probably". Cut separation is a more critical and one-sided procedure: estimation errors in $\hat{\boldsymbol{x}}$ may lead to the separation of cuts that are not violated and there is no compensation for it. So, in situations where significant duality gaps need to be reduced by cutting it may be advisable to use a hybrid approach, applying a LR method to get a near-optimal LDP solution and then switching to a CG

(hot-started with the columns corresponding to the last generated points and also using some tricks described in Chapter 7) to obtain an exact $\boldsymbol{x}^*$, even if this takes more time. A discussion on that can be found in Section 3.3 of Pessoa et al. [2010].

5.3. Relax-and-Cut. Consider the IP (5.9), and let $(\boldsymbol{\pi}'\,\boldsymbol{\rho}')$ and $\boldsymbol{x}'$, be near-optimal Lagrangian multipliers and the corresponding LS integer solution, respectively (so, $L(\boldsymbol{\pi}', \boldsymbol{\rho}') = L(\boldsymbol{x}', \boldsymbol{\pi}', \boldsymbol{\rho}')$). To obtain better bounds, *Relax-and-Cut* separates valid inequalities cutting $\boldsymbol{x}'$ and immediately dualizes them, so they do not change the structure of the LSs.

The approach has its roots in the LR algorithm by Balas and Christofides [1981] for the asymmetric TSP, where the degree constraints are kept in the LS (which can then be solved as an easy Assignment Problem) and the Subtour Elimination constraints are dualized. As there is an exponential number of such constraints, this has to be done dynamically: given the current assignment solution $\boldsymbol{x}'$, its subcycles are identified and the corresponding Subtour Elimination Constraints receive positive multipliers, so those subcycles are less likely to appear in the next iterations.

The Relax-and-Cut technique was advanced in Lucena [1992] and Escudero et al. [1994], the latter work proposing its name. Discussions on it can be found in Guignard [2003], Lucena [2005] and Ralphs and Galati [2005]. As observed by them, for a typical $\mathcal{NP}$-hard LCOP, the separation problems for most known families of valid inequalities are also $\mathcal{NP}$-hard (usually, only the simplest families of inequalities have polynomial exact separation). This means that the problem of finding an inequality in one of those families that cuts a given arbitrary fractional point $\boldsymbol{x}^*$ often has to be handled by separation heuristics. However, the separation problem restricted to integer points $\boldsymbol{x}'$ may become much easier, even polynomial. This is a feature of Relax-and-Cut schemes.

We view Relax-and-Cut as a potentially effective but *heuristic* separation procedure, not only in the sense that cuts violated by a primal fractional solution $\boldsymbol{x}^*$ may be missed but also in the sense that there is no guarantee that a cut that is violated by the integer point $\boldsymbol{x}'$ also cuts $\boldsymbol{x}^*$. This is not suprising after one realizes that turning $\mathcal{NP}$-hard separation problems into polynomial

ones is perhaps "too good to be true". There are some other caveats:

- It is only possible to cut an integer point $\boldsymbol{x}'$ if it is not a feasible solution of (5.9) (or equivalently, if $\boldsymbol{x}'$ would correspond to a non-proper variable in a CG scheme). If so, $\boldsymbol{x}'$ violates some original constraints that were dualized. One should be careful to avoid "cuts that are falsely violated". Consider for example the Held-Karp LR method for the TSP. The $\boldsymbol{x}'$ solutions correspond to 1-trees, and therefore never violate Subtour Elimination constraints in format (5.42e). However, they often violate many Subtour Elimination constraints in format (3.5c)! This happens because, as explained in Note 3.13, those two formats are only equivalent for solutions that satisfy the degree constraints (5.42b), which is not the case, since the HK method dualizes (5.42b). The point is that the separation of Subtour Elimination constraints, in any format is useless, in the sense of not leading to bounds stronger than the original bound z_{LD}. In order to do that, one should try to separate more complex families of cuts like 2-matchings, combs, etc (see Section 3.4.1). *In general, in a LR over the IP* (5.9)*, a cut can only improve the bound* z_{LD} *if it cuts the polyhedron given by* $\boldsymbol{Ax} = \boldsymbol{b}$, $\boldsymbol{Dx} \geq \boldsymbol{d}, \boldsymbol{x} \in Conv(Int(P))$.
- Relax-and-Cut can not be applied to cases where identical subproblems are aggregated and only proper variables are priced. Consider the CVRP bound z_{M} obtained by CG from solving (4.29), assuming that only elementary routes are priced. A bound very close to the theoretical $z_{\text{LD}} = z_{\text{M}}$ can be obtained by LR (similarly to what is done in Christofides et al. [1981] using q-routes). Even if z_{LD} has a fairly large gap with respect to z_{IP} (which usually happens), all integer $\boldsymbol{x}'$ vectors that are LSs solutions correspond to feasible routes. It is not possible to separate cuts over them! Looking at the situation in more depth, $\boldsymbol{x}'$ can be converted into an infeasible integer solution to the original formulation (4.27) having U identical copies of the same route. That solution is infeasible because it violates the dualized degree constraints (4.27b). Anyway, such an original solution is still not useful for separating cuts. For example, any set $S \subset V_+$ that does not include vertices in the route corresponding to $\boldsymbol{x}'$ leads to a maximally violated (with zero LHS) Rounded Capacity Cut (3.6c). Which one should be chosen? The separation algo-

rithm would be blind and would indeed separate a "random cut" in that family.

The only alternative for cutting in the above mentioned context would be to accumulate many successive vectors $\boldsymbol{x}'$ in a formula similar to (5.50) to obtain an approximated primal fractional solution. But this has the difficulties pointed out in the previous note and loses the potential advantages of doing separation over integer solutions.

Exercises

E 5.1. Let UB be the value of the best known solution for the IP $\min \boldsymbol{cx}$ subject to $\boldsymbol{Ax} \geq \boldsymbol{b}$, $\boldsymbol{x} \in \mathbb{Z}^n_+$ and let $\boldsymbol{\rho}' \in \mathbb{R}^{1\times m}_+$ be a feasible solution to the dual of its linear relaxation. Use LR to prove that every integer variable x_j, $j \in [n]$, such that $\boldsymbol{\rho}'\boldsymbol{b} + c_j - \boldsymbol{\rho}'\boldsymbol{a}_j \geq UB$ should have value zero in any improving IP solution.

E 5.2. Relax the integrality constraints in (5.36) and consider the resulting continuous non-linear problem. Apply LR to it, dualizing (5.36b). Solve the resulting LDP by the cutting plane method, starting from LP (5.38). Use the fact that subproblems can be solved by closed-form expressions.

E 5.3. Project exercise. Implement the Held-Karp 1-tree LR method for the TSP. Use the Subradient Algorithm with the step sizes suggested in Held et al. [1974]. Plot the 1-trees from successive iterations and observe if they can get close to being tours. However, to find proven optimal integer solutions use an estimated primal fractional solution to choose the branching variables in a binary search tree. Test on small instances from TSPLIB.

E 5.4. Project exercise. Implement the Held-Karp 1-tree CG bounding method for the TSP, as proposed in Held and Karp [1970]. Be patient when running the CG for larger instances.

E 5.5. Open project exercise. Use LR and the Subgradient Algorithm (or other more sophisticated Lagrangian method) for implementing a stand-alone code for solving LPs in format $\min z = \boldsymbol{cx}$ subject to $\boldsymbol{Ax} \geq \boldsymbol{b}$, $\boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{u}$, where $\boldsymbol{A}$ has dimension $m \times n$. Assume that vector $\boldsymbol{u}$ provides finite bounds to all variables and $\boldsymbol{b} \geq \boldsymbol{0}$, so unboundedness or infeasibility is not possible. How can you use the resulting near-optimal dual solution to compute a near-optimal primal solution? Discuss possibilities for "crossing-over" to a primal basic feasible solution. Compare your code with simplex and interior-point LP solvers for different instance sizes, starting with small ones. Try instances with relatively few rows and many columns, like those arising as linear relaxations of typical set covering problems. Be ready for defeat, but still have fun!

Part II

Topics in Column Generation

Chapter 6

Column Generation Based Heuristics

Chapter 7

Column Generation Convergence

7.1. Managing the Restricted Master LP

7.1.1. RMLP initialization

7.1.2. Pricing policies

7.1.3. RMLP clean-up

7.2. Dual stabilization

7.2.1. Dual feasible cuts

7.2.2. Stabilization by dual smoothing

7.2.3. Stabilization by penalty functions

7.2.4. Master constraint aggregation

7.2.5. Lagrangian hot-start

7.2.6. Solving RMLPs by interior-point methods

Chapter 8

Advanced Branching

8.1. Advanced Branching Schemes

8.2. Strong Branching for CG

Chapter 9

The Dynamic Programming Labeling Algorithm for the RCSP

Chapter 10

Non-Robust Cuts

10.1. General non-robust cuts

10.2. Limited-memory Rank 1 Cuts

10.2.1. Packing Sets

Chapter 11

Reduced Cost Fixing and Related Techniques

Chapter 12

Additional Case Studies

Chapter 13

Software for Column Generation

References

Hernán Abeledo, Ricardo Fukasawa, Artur Pessoa, and Eduardo Uchoa. The time dependent traveling salesman problem: polyhedra and algorithm. *Mathematical Programming Computation*, 5(1):27–55, 2013. doi: 10.1007/s12532-012-0047-y. 196

Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technischen Universität Berlin, 2007. doi: 10.14279/depositonce-1634. 90

Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005. doi: 10.1016/j.orl.2004.04.002. 97, 98

Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2):473–506, 2020. doi: 10.1287/ijoc.2018.0857. 89

Narasimaha R Achuthan, Louis Caccetta, and Stephen P Hill. An improved branch-and-cut algorithm for the capacitated vehicle routing problem. *Transportation Science*, 37 (2):153–169, 2003. doi: 10.1287/trsc.37.2.153.15243. 87

Tommaso Adamo, Gianpaolo Ghiani, Pierpaolo Greco, and Emanuela Guerriero. Properties and bounds for the single-vehicle capacitated routing problem with time-dependent travel times and multiple trips. In Greg H Parlier, Federico Liberatore, and Marc Demange, editors, *Proceedings of the 10th International Conference on Operations Research and Enterprise Systems (ICORES 2021)*, pages 82–87, 2021. doi: 10.5220/0010322500820087. 181

Yogesh Agarwal, Kamlesh Mathur, and Harvey M Salkin. A set-partitioning-based exact algorithm for the vehicle routing problem. *Networks*, 19(7):731–749, 1989. doi: 10.1002/net.3230190702. 198

Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993. ISBN 978-0136175490. 37, 40

Cláudio Alves, François Clautiaux, José M Valério de Carvalho, and Jürgen Rietz. *Dual-Feasible Functions for Integer Programming and Combinatorial Optimization: Basics, Extensions and Applications*. EURO Advanced Tutorials on Operational Research. Springer, 2016. doi: 10.1007/978-3-319-27604-5. 178

Adriana C F Alvim, Celso C Ribeiro, Fred Glover, and Dario J Aloise. A hybrid improvement heuristic for the one-dimensional bin packing problem. *Journal of Heuristics*, 10 (2):205–229, 2004. doi: 10.1023/B:HEUR.0000026267.44673.ed. 178

Yash P Aneja. An integer linear programming approach to the Steiner problem in graphs. *Networks*, 10(2):167–178, 1980. ISSN 1097-0037. doi: 10.1002/net.3230100207. 106

Kurt M Anstreicher and Laurence A Wolsey. Two "well-known" properties of subgradient optimization. *Mathematical Programming*, 120:213–220, 2009. doi: 10.1007/s10107-007-0148-y. 263

Leif H Appelgren. A column generation algorithm for a ship scheduling problem. *Transportation Science*, 3(1):53–68, 1969. doi: 10.1287/trsc.3.1.53. 49, 196

Leif H Appelgren. Integer programming methods for a vessel scheduling problem. *Transportation Science*, 5(1):64–78, 1971. doi: 10.1287/trsc.5.1.64. 196

David L Applegate, Robert E Bixby, Vašek Chvátal, and William J Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2007. doi: 10.1515/9781400841103. 85, 104

J P Arabeyre, J Fearnley, F C Steiger, and W Teather. The airline crew scheduling problem: A survey. *Transportation Science*, 3(2):140–163, 1969. doi: 10.1287/trsc.3.2.140. 96

Jesús R Araque, Gautham Kudva, Thomas L Morin, and Joseph F Pekny. A branch-and-cut algorithm for vehicle routing problems. *Annals of Operations Research*, 50:37–59, 1994. doi: 10.1007/BF02085634. 87

José Miguel Aroztegui and Artur Pessoa. A cutting plane approach to maximization of fundamental frequency in truss topology optimization. *Structural and Multidisciplinary Optimization*, 67(4):54, 2024. doi: 10.1007/s00158-024-03778-y. 250

Philippe Augerat, José Manuel Belenguer, Enrique Benavent, Angel Corberán, Denis Naddef, and Giovanni Rinaldi. Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical Report 949-M, Université Joseph Fourier, Grenoble, France, 1995. https://www.osti.gov/etdeweb/servlets/purl/289002 (accessed on August 2024). 87

Pasquale Avella, Maurizio Boccia, and Bernardo D'Auria. Near-optimal solutions of large-scale single-machine scheduling problems. *INFORMS Journal on Computing*, 17(2):183–191, 2005. doi: 10.1287/ijoc.1040.0069. 254

Pasquale Avella, Maurizio Boccia, and Igor Vasilyev. A computational study of exact knapsack separation for the generalized assignment problem. *Computational Optimization*

and Applications, 45(3):543–555, 2010. doi: 10.1007/s10589-008-9183-8. 170, 188, 189, 246

Laura Bahiense, Nelson Maculan, and Claudia Sagastizábal. The volume algorithm revisited: relation with bundle methods. *Mathematical Programming*, 94:41–69, 2002. doi: 10.1007/s10107-002-0357-3. 263

Egon Balas and Nicos Christofides. A restricted Lagrangean approach to the traveling salesman problem. *Mathematical Programming*, 21(1):19–46, 1981. doi: 10.1007/BF01584228. 265

Roberto Baldacci, Nicos Christofides, and Aristide Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115:351–385, 2008. doi: 10.1007/s10107-007-0178-5. 178, 200

Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283, 2011. doi: 10.1287/opre.1110.0975. 149, 178, 180, 200

Roberto Baldacci, Stefano Coniglio, Jean-François Cordeau, and Fabio Furini. A numerically exact algorithm for the bin-packing problem. *INFORMS Journal on Computing*, 36(1): 141–162, 2023. doi: 10.1287/ijoc.2022.0257. 178

Isaac Balster, Teobaldo Bulhões, Pedro Munari, Artur Pessoa, and Ruslan Sadykov. A new family of route formulations for split delivery vehicle routing problems. *Transportation Science*, 57(5):1359–1378, 2023. doi: 10.1287/trsc.2022.0085. 181

Francisco Barahona and Ranga Anbil. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, 87(3):385–399, 2000. doi: 10.1007/s101070050002. 263

Francisco Barahona and Ali Ridha Mahjoub. On the cut polytope. *Mathematical programming*, 36:157–173, 1986. doi: 10.1007/BF02592023. 85

Jonathan F Bard, George Kontoravdis, and Gang Yu. A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science*, 36(2):250–269, 2002. doi: 10.1287/trsc.36.2.250.565. 180

Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin W P Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998. doi: 10.1287/opre.46.3.316. 163, 198

Cynthia Barnhart, Christopher A Hane, and Pamela H Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48(2):318–326, 2000. doi: 10.1287/opre.48.2.318.12378. 198

Saverio Basso and Alberto Ceselli. Distributed asynchronous column generation. *Computers & Operations Research*, 146:105894, 2022. doi: 10.1016/j.cor.2022.105894. 160

Saverio Basso and Alberto Ceselli. A data driven Dantzig–Wolfe decomposition framework. *Mathematical Programming Computation*, 15(1):153–194, 2023. doi: 10.1007/s12532-022-00230-4. 160

Mokhtar S Bazaraa, John J Jarvis, and Hanif D Sherali. *Linear programming and network flows*. John Wiley & Sons, 4th edition, 2010. doi: 10.1002/9780471703778. 3, 12, 49

John E Beasley. Lagrangian relaxation. In Colin R Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, Advanced Topics in Computer Science, pages 243–303. Blackwell Scientific Publications, 1993. 262, 263

Gleb Belov and Guntram Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research*, 171(1):85–106, 2006. doi: 10.1016/j.ejor.2004.08.036. 178

Gleb Belov, Adam N Letchford, and Eduardo Uchoa. A node-flow model for 1D stock cutting: Robust branch-cut-and-price. Technical Report 7, Universidade Federal Fluminense, Engenharia de Produção, 2005. https://www.producao.uff.br/conteudo/rpep/volume52005/RelPesq_V5_2005_07.pdf (accessed on August 2024). 178

Jacobus F Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962. doi: 10.1007/BF01386316. 183

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. doi: 10.1016/j.ejor.2020.07.063. 98

Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and Olivier Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1:76–94, 1971. doi: 10.1007/BF01584074. 97

Martin Bergner, Alberto Caprara, Alberto Ceselli, Fabio Furini, Marco E Lübbecke, Enrico Malaguti, and Emiliano Traversi. Automatic Dantzig–Wolfe reformulation of mixed integer programs. *Mathematical Programming*, 149(1):391–424, 2015. doi: 10.1007/s10107-014-0761-5. 56, 160

Dimitris Bertsimas and John N Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997. 3, 11, 59

Nicola Bianchessi, Timo Gschwind, and Stefan Irnich. Resource-window reduction by reduced costs in path-based formulations for routing and scheduling problems. *INFORMS Journal on Computing*, 36(1):224–244, 2024. doi: 10.1287/ijoc.2022.0214. 246

Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, Second edition, 2021. https://www.iospress.com/catalog/books/handbook-of-satisfiability-2 (accessed on August 2024). 91

Louis-Philippe Bigras, Michel Gamache, and Gilles Savard. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization*, 5(4):685–699, 2008. doi: 10.1016/j.disopt.2008.04.001. 254

Robert E Bixby. A brief history of linear and mixed-integer programming computation. In Martin Grötschel, editor, *Optimization Stories*, volume 21th ISMP of *Documenta Mathematica*, pages 107–121. EMS Press, 2012. doi: 10.4171/DMS/6/16. 88

Robert E Bixby. LP & MIP solving, 2022. Presentation at Bonn Hausdorff School: Computational Combinatorial Optimization. 90

Ulrich Blasum and Winfried Hochstättler. Application of the branch and cut method to the vehicle routing problem. Technical Report zpr2000-386, Universität zu Köln, Zentrum für Angewandte Informatik, 2000. 87

Maurizio Boccia, Antonio Sforza, Claudio Sterle, and Igor Vasilyev. A cut and branch approach for the capacitated p-median problem based on Fenchel cutting planes. *Journal of Mathematical Modeling and Algorithms*, 7:43–58, 2008. doi: 10.1007/s10852-007-9074-5. 189

Ivan Boldyrev and Till Düppe. Programming the USSR: Leonid V. Kantorovich in context. *The British Journal for the History of Science*, 53(2):255–278, 2020. doi: 10.1017/S0007087420000059. 172

Alan Bollard. *Economists at War: How a Handful of Economists Helped Win and Lose the World Wars*, chapter The Calculating Iceman: Leonid Kantorovich in the USRR, 1941-42. Oxford University Press, 2020. doi: 10.1093/oso/9780198846000.001.0001. 172

Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher

Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. The SCIP Optimization Suite 9.0. ZIB-Report 24-02-29, Zuse Institute Berlin, 2024. https://nbn-resolving.org/urn:nbn:de:0297-zib-95528 (accessed on August 2024). 91

Pierre Bonami, Domenico Salvagnin, and Andrea Tramontani. Implementing automatic Benders decomposition in a modern MIP solver. In Daniel Bienstock and Giacomo Zambelli, editors, *International Conference on Integer Programming and Combinatorial Optimization, IPCO 2020, Proceedings*, volume 12125 of *Lecture Notes on Computer Science*, pages 78–90. Springer, 2020. doi: 10.1007/978-3-030-45771-6_7. 184, 185

Quentin Botton, Bernard Fortz, Luis Gouveia, and Michael Poss. Benders decomposition for the hop-constrained survivable network design problem. *INFORMS Journal on Computing*, 25(1):13–26, 2013. doi: 10.1287/ijoc.1110.0472. 184

Edward H Bowman. The schedule-sequencing problem. *Operations Research*, 7(5):621–624, 1959. doi: 10.1287/opre.7.5.621. 252

E Andrew Boyd. Generating Fenchel cutting planes for knapsack polyhedra. *SIAM Journal on Optimization*, 3(4):734–750, 1993. doi: 10.1137/0803038. 187

E Andrew Boyd. Fenchel cutting planes for integer programs. *Operations Research*, 42(1): 53–64, 1994. doi: 10.1287/opre.42.1.53. 187

Mikhail A Bragin. Survey on Lagrangian relaxation for MILP: importance, challenges, historical review, recent advancements, and opportunities. *Annals of Operations Research*, 333(1):29–45, 2024. doi: 10.1007/s10479-023-05499-9. 262

Olivier Briant, Claude Lemaréchal, Philippe Meurdesoif, Sophie Michel, Nancy Perrot, and François Vanderbeck. Comparison of bundle and classical column generation. *Mathematical programming*, 113:299–344, 2008. doi: 10.1007/s10107-006-0079-z. 262

Teobaldo Bulhões, Ruslan Sadykov, and Eduardo Uchoa. A branch-and-price algorithm for the minimum latency problem. *Computers & Operations Research*, 93:66–78, 2018. doi: 10.1016/j.cor.2018.01.016. 196

Teobaldo Bulhões, Ruslan Sadykov, Anand Subramanian, and Eduardo Uchoa. On the exact solution of a large class of parallel machine scheduling problems. *Journal of Scheduling*, 23:411–429, 2020. doi: 10.1007/s10951-020-00640-z. 261

Der-San Chen, Robert G Batson, and Yu Dang. *Applied integer programming: modeling and solution*. John Wiley & Sons, 2010. doi: 10.1002/9781118166000. 71, 100

Rui Chen and James Luedtke. On generating lagrangian cuts for two-stage stochastic integer programs. *INFORMS Journal on Computing*, 34(4):2332–2349, 2022. doi: 10.1287/ijoc.2022.1185. 185

Rui Chen, Oktay Günlük, and Andrea Lodi. Recovering Dantzig–Wolfe bounds by cutting planes. *Operations Research*, 2024. doi: 10.1287/opre.2023.0048. 56, 189, 190

Elliot Ward Cheney and Allen A Goldstein. Newton's method for convex programming and Tchebycheff approximation. *Numerische Mathematik*, 1:253–268, 1959. doi: 10.1007/BF01386389. 49, 233

Sunil Chopra and Mendu Rammohan Rao. The Steiner tree problem I: Formulations, compositions and extension of facets. Technical Report 88-82, New York University, 1988a. Later published in *Mathematical Programming*, 64: 209–229, 1994, doi: 10.1007/BF01582573. 107

Sunil Chopra and Mendu Rammohan Rao. The Steiner tree problem II: Properties and classes of facets. Technical Report 88-83, New York University, 1988b. Later published in *Mathematical Programming*, 64: 231–246, 1994, doi: 10.1007/BF01582574. 107

Nicos Christofides, Aristide Mingozzi, and Paolo Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical programming*, 20(1):255–282, 1981. doi: 10.1007/BF01589353. 149, 251, 266

William Chung. Dantzig–Wolfe decomposition. In James J Cochran, Louis A Cox, Pinar Keskinocak, Jeffrey P Kharoufeh, and J Cole Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, 2011. doi: 10.1002/9780470400531.eorms0220. 62

Vašek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete mathematics*, 4(4):305–337, 1973. doi: 10.1016/0012-365X(73)90167-2. 93

Vašek Chvátal. *Linear programming*. W.H. Freeman, 1983. 3, 12

Armin Claus and Nelson Maculan. Une nouvelle formulation du probleme de Steiner sur un graphe. Technical Report 280, Université de Montréal, Centre de recherche sur les transports, 1983. 107

François Clautiaux, Cláudio Alves, and José M Valério de Carvalho. A survey of dual-feasible and superadditive functions. *Annals of Operations Research*, 179:317–342, 2010. doi: 10.1007/s10479-008-0453-8. 178

Jean-Maurice Clochard and Denis Naddef. Using path inequalities in a branch and cut code for the symmetric traveling salesman problem. In Giovanni Rinaldi and Laurence A

Wolsey, editors, *Proceedings of the 3rd Integer Programming and Combinatorial Optimization, IPCO 1993*, pages 291–311, 1993. 100

Concorde. Concorde TSP Solver, 2020. https://www.math.uwaterloo.ca/tsp/concorde (accessed on August 2024). 104

Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. Extended formulations in combinatorial optimization. *4OR*, 8(1):1–48, 2010. doi: 10.1007/s10288-010-0122-z. 105

Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer programming*, volume 271 of *Graduate Texts in Mathematics*. Springer, 2014. doi: 10.1007/978-3-319-11008-0. 71, 163

Stefano Coniglio, Fabio Furini, and Pablo San Segundo. A new combinatorial branch-and-bound algorithm for the knapsack problem with conflicts. *European Journal of Operational Research*, 289(2):435–455, 2021. doi: 10.1016/j.ejor.2020.07.023. 153

Claudio Contardo and Rafael Martinelli. A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization*, 12: 129–146, 2014. doi: 10.1016/j.disopt.2014.03.001. 178

William J Cook, William H Cunningham, William R Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. Wiley Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1998. doi: 10.1002/9781118627372. 243

COPT. Cardinal Optimizer. https://www.copt.de (accessed on August 2024). 91

Alysson M Costa. A survey on benders decomposition applied to fixed-charge network design problems. *Computers & Operations Research*, 32(6):1429–1450, 2005. doi: 10.1016/j.cor.2003.11.012. 183

Luciano Costa, Claudio Contardo, and Guy Desaulniers. Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, 53(4):946–985, 2019. doi: 10.1287/trsc.2018.0878. 181

Richard Cottle. *The Basic George B. Dantzig*. Stanford University Press, 2003. ISBN 9780804748346. http://www.sup.org/books/title/?id=5507(accessed on August 2024). 9

Cplex. IBM ILOG CPLEX Optimizer. https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer (accessed on August 2024). 91

Renan F F da Silva and Rafael Schouery. A branch-and-cut-and-price algorithm for cutting stock and related problems. *Revista Eletrônica de Iniciação Científica em Computação*, 22(1):31–40, 2024. doi: 10.5753/reic.2024.4646. 178

Robert J Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965. doi: 10.1093/comjnl/8.3.250. 88

Caio Marinho Damião, João Marcos Pereira Silva, and Eduardo Uchoa. A branch-cut-and-price algorithm for the cumulative capacitated vehicle routing problem. *4OR*, 21:41–71, 2023. doi: 10.1007/s10288-021-00498-7. 181

George B Dantzig. Programming in a linear structure. In *54th Summer Meeting and 30th Colloquium of the American Mathematical Society, Bulletin of the American Mathematical Society*, volume 54(11), pages 1074–1074, 1948. doi: 10.1090/S0002-9904-1948-09093-0. 9

George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. In Tjalling C Koopmans, editor, *Activity analysis of production and allocation: Proceedings of a conference*, volume 13 of *Cowles Comission Monographs*, pages 339–347. John Wiley & Sons, 1951. https://cowles.yale.edu/research/cfm-13-activity-analysis-production-and-allocation(accessed on August 2024). 9

George B Dantzig. Notes on Linear Programming, Part III: Computational algorithm of the revised simplex method. Technical Report RM-1266, Rand Corporation, 1953. https://www.rand.org/pubs/research_memoranda/RM1266.html(accessed on August 2024). 9, 174

George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959. doi: 10.1287/mnsc.6.1.80. 85

George B Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960. doi: 10.1287/opre.8.1.101. 49, 50

George B Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954. doi: 10.1287/opre.2.4.393. 84, 241, 242

George B Dantzig, Alexander Orden, and Philip Wolfe. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5(2):183–195, 1955. doi: 10.2140/pjm.1955.5.183. 7

Vinícius Loti de Lima, Cláudio Alves, François Clautiaux, Manuel Iori, and José M Valério de Carvalho. Arc flow formulations based on dynamic programming: Theoretical foundations and applications. *European Journal of Operational Research*, 296(1):3–21, 2022. doi: 10.1016/j.ejor.2021.04.024. 109, 178

Vinícius Loti de Lima, Manuel Iori, and Flávio Keidi Miyazawa. Exact solution of network flow models with strong relaxations. *Mathematical Programming*, 197(2):813–846, 2023. doi: 10.1007/s10107-022-01785-9. 109

Claudio N de Meneses and Cid C de Souza. Exact solutions of rectangular partitions via integer programming. *International Journal of Computational Geometry & Applications*, 10(05):477–522, 2000. doi: 10.1142/S0218195900000280. 110

Caterina De Simone and Giovanni Rinaldi. A cutting plane algorithm for the max-cut problem. *Optimization Methods and Software*, 3(1-3):195–214, 1994. doi: 10.1080/10556789408805564. 85

Maxence Delorme and Manuel Iori. Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. *INFORMS Journal on Computing*, 32(1):101–119, 2020. doi: 10.1287/ijoc.2018.0880. 178

Guy Desaulniers. Branch-and-price-and-cut for the split-delivery vehicle routing problem with time windows. *Operations Research*, 58(1):179–192, 2010. doi: 10.1287/opre.1090.0713. 164

Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon, editors. *Column Generation*, volume 5. Springer Science & Business Media, 2005. doi: 10.1007/b135457. 163

Guy Desaulniers, François Lessard, and Ahmed Hadjar. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, 42(3):387–404, 2008. doi: 10.1287/trsc.1070.0223. 180

Guy Desaulniers, Jørgen G Rakke, and Leandro C Coelho. A branch-price-and-cut algorithm for the inventory-routing problem. *Transportation Science*, 50(3):1060–1076, 2016. doi: 10.1287/trsc.2015.0635. 164

Martin Desrochers and François Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation science*, 23(1):1–13, 1989. doi: 10.1287/trsc.23.1.1. 198

Martin Desrochers, Jacques Desrosiers, and Marius M Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2): 342–354, 1992. doi: 10.1287/opre.40.2.342. 198

Jacques Desrosiers and Marco E Lübbecke. A primer in column generation. In Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon, editors, *Column generation*, pages 1–32. Springer, 2005. doi: 10.1007/0-387-25486-2_1. 163

Jacques Desrosiers and Marco E Lübbecke. Branch-price-and-cut algorithms. In James J Cochran, Louis A Cox, Pinar Keskinocak, Jeffrey P Kharoufeh, and J Cole Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, 2011. doi: 10.1002/9780470400531.eorms0118. 200

Jacques Desrosiers, François Soumis, and Martin Desrochers. Routing with time windows by column generation. *Networks*, 14(4):545–565, 1984. doi: 10.1002/net.3230140406. 197

Jacques Desrosiers, Yvan Dumas, Marius M Solomon, and François Soumis. Time constrained routing and scheduling. In Michael Ball, Thomas Magnanti, Clyde L Monma, and George L Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, pages 35–139. Elsevier, 1995. doi: 10.1016/S0927-0507(05)80106-9. 198

Jacques Desrosiers, Marco Lübbecke, Guy Desaulniers, and Jean-Bertrand Gauthier. *Branch-and-Price*, volume G-2024-36 of *Les Cahiers du GERAD*. Groupe d'études et de recherche en analyse des décisions (GERAD), June 2024. https://www.gerad.ca/en/papers/G-2024-36 (accessed on August 2024). 163

John J Dinkel, William H Elliott, and Gary A Kochenberger. Computational aspects of cutting-plane algorithms for geometric programming problems. *Mathematical Programming*, 13:200–220, 1977. doi: 10.1007/BF01584337. 236

Robert Dorfman, Paul Anthony Samuelson, and Robert M Solow. *Linear programming and economic analysis*. McGraw-Hill, 1958. 172

Yvan Dumas, Jacques Desrosiers, and François Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, 1991. doi: 10.1016/0377-2217(91)90319-Q. 198

Martin E Dyer and Laurence A Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26(2-3): 255–270, 1990. doi: 10.1016/0166-218X(90)90104-K. 252

Bernard P Dzielinski and Ralph E Gomory. Optimal programming of lot sizes, inventory and labor allocations. *Management Science*, 11(9):874–890, 1965. doi: 10.1287/mnsc.11.9.874. 196

Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69B(1–2):125–130, 1965. doi: 10.6028/jres.069B.013. 103

Jack Edmonds. Matroids and the greedy algorithm. *Mathematical programming*, 1:127–136, 1971. doi: 10.1007/BF01584082. 242

Friedrich Eisenbrand. Note - on the membership problem for the elementary closure of a polyhedron. *Combinatorica*, 19(2):297–300, 1999. doi: 10.1007/s004930050057. 94

Michael Ellman. Leonid Kantorovich. In Vladimir Avtonomov and Harald Hagemann, editors, *Russian and Western Economic Thought: Mutual Influences and Transfer of Ideas*, pages 427–447. Springer, 2022. doi: 10.1007/978-3-030-99052-7_20. 172

Robert Entriken. Parallel decomposition: Results for staircase linear programs. *SIAM Journal on Optimization*, 6(4):961–977, 1996. doi: 10.1137/S1052623494253286. 61

Najib Errami, Eduardo Queiroga, Ruslan Sadykov, and Eduardo Uchoa. VRPSolverEasy: a Python library for the exact solution of a rich vehicle routing problem. *INFORMS Journal on Computing*, 2023. doi: 10.1287/ijoc.2023.0103. 181

Laureano F Escudero, Monique Guignard, and Kavindra Malik. A Lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships. *Annals of Operations Research*, 50:219–237, 1994. doi: 10.1007/BF02085641. 265

Hugh Everett III. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research*, 11(3):399–417, 1963. doi: 10.1287/opre.11.3.399. 262

Ashkan Fakhri, Mehdi Ghatee, Antonios Fragkogios, and Georgios K.D. Saharidis. Benders decomposition with integer subproblem. *Expert Systems with Applications*, 89:20–30, 2017. doi: 10.1016/j.eswa.2017.07.017. 184

Alan A Farley. A note on bounding a class of linear programming problems, including cutting stock problems. *Operations Research*, 38(5):922–923, 1990. doi: 10.1287/opre.38.5.922. 182

Sándor P Fekete and Jörg Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical programming*, 91:11–31, 2001. doi: 10.1007/s101070100243. 178

Giovanni Felici, Claudio Gentile, and Giovanni Rinaldi. Solving large MIP models in supply chain management by branch & cut. Technical Report 522, Consiglio Nazionale delle Ricerche, Istituto di Analisi dei Sistemi ed Informatica, 2000. 198

Matteo Fischetti and Andrea Lodi. Optimizing over the first Chvátal closure. *Mathematical Programming*, 110(1):3–20, 2007. doi: 10.1007/s10107-006-0054-8. 94, 201

Marshall L Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management science*, 27(1):1–18, 1981. doi: 10.1287/mnsc.27.1.1. 244, 262, 263

Lester Randolph Ford Jr and Delbert R Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958. doi: 10.1287/mnsc.5.1.97. 40, 49, 50

John J Forrest and Donald Goldfarb. Steepest-edge simplex algorithms for linear programming. *Mathematical programming*, 57:341–374, 1992. doi: 10.1007/BF01581089. 12

Brian A Foster and David M Ryan. An integer programming approach to the vehicle scheduling problem. *Journal of the Operational Research Society*, 27(2):367–384, 1976. doi: 10.1057/jors.1976.63. 197

Antonio Frangioni. Solving semidefinite quadratic problems within nonsmooth optimization algorithms. *Computers & Operations Research*, 23(11):1099–1118, 1996. doi: 10.1016/0305-0548(96)00006-8. 244

Antonio Frangioni, Bernard Gendron, and Enrico Gorgone. On the computational efficiency of subgradient methods: a case study with Lagrangian bounds. *Mathematical Programming Computation*, 9:573–604, 2017. doi: 10.1007/s12532-017-0120-7. 262

Matheus Freitas, João Marcos Pereira Silva, and Eduardo Uchoa. A unified exact approach for clustered and generalized vehicle routing problems. *Computers & Operations Research*, page 106040, 2022. doi: 10.1016/j.cor.2022.106040. 181

Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming*, 106(3):491–511, 2006. doi: 10.1007/s10107-005-0644-x. 178, 179, 199

Gerald Gamrath. Generic branch-cut-and-price. Master's thesis, Technischen Universität Berlin, 2010. urn:nbn:de:0297-zib-57543 https://opus4.kobv.de/opus4-zib/frontdoor/index/index/docId/5754 (accessed on August 2024). 51, 163, 164

Roy Gardner. LV Kantorovich: the price implications of optimal planning. *Journal of Economic Literature*, 28(2):638–648, 1990. https://www.jstor.org/stable/2727266 (accessed on August 2024). 172

Michael R Garey and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, volume 174. W.H.Freeman, 1979. ISBN 0-7167-1044-7. xx, 77

Arthur M Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972. doi: 10.1007/BF00934810. 184

Arthur M Geoffrion. Lagrangean relaxation for integer programming. In Michel L Balinski, editor, *Approaches to Integer Programming*, volume 2 of *Mathematical Programming Studies*, pages 82–114. Springer, 1974. doi: 10.1007/BFb0120690. 163, 216, 262

Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961. doi: 10.1287/opre.9.6.849. 143, 144, 170, 175, 196, 203

Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting stock problem — Part II. *Operations Research*, 11(6):863–888, 1963. doi: 10.1287/opre.11.6.863. 49, 176, 196

Paul C Gilmore and Ralph E Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13(1):94–120, 1965. doi: 10.1287/opre.13.1.94. 176

Michel X Goemans. The Steiner tree polytope and related polyhedra. *Mathematical programming*, 63:157–182, 1994. doi: 10.1007/BF01582064. 108

Ralph E Gomory. Outline of an algorithm for integer solution to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958. doi: 10.1090/S0002-9904-1958-10224-4. 88, 93

Ralph E Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597-PR, RAND Corporation, 1960. https://www.rand.org/pubs/research_memoranda/RM2597.html (accessed on August 2024). 88

Luis Gouveia, Markus Leitner, and Mario Ruthmair. Layered graph approaches for combinatorial optimization problems. *Computers & Operations Research*, 102:22–38, 2019. doi: 10.1016/j.cor.2018.09.007. 108

Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and Alexander H G Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In Peter L Hammer, Ellis Lane Johnson, and Bernhard H Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. North Holland, 1979. doi: 10.1016/S0167-5060(08)70356-X. 252

Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981. doi: 10.1007/BF02579273. 103, 187

Monique Guignard. Lagrangean relaxation. *TOP - Transactions in Operations Research*, 11(2):151–200, 2003. doi: 10.1007/BF02579036. 262, 265

Monique Guignard and Siwhan Kim. Lagrangean decomposition: A model yielding stronger Lagrangean bounds. *Mathematical programming*, 39(2):215–228, 1987. doi: 10.1007/BF02592954. 55

Gurobi. https://www.gurobi.com/solutions/gurobi-optimizer/ (accessed on August 2024). 91

Keld Helbig Hansen and Jakob Krarup. Improvements of the Held—Karp algorithm for the symmetric traveling-salesman problem. *Mathematical Programming*, 7:87–96, 1974. doi: 10.1007/BF01585505. 243

Michael Held and Richard M Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970. doi: 10.1287/opre.18.6.1138. 240, 242, 267

Michael Held and Richard M Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical programming*, 1(1):6–25, 1971. doi: 10.1007/BF01584070. 230, 242, 243, 249, 250, 262, 263

Michael Held, Philip Wolfe, and Harlan P Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88, 1974. doi: 10.1007/BF01580223. 230, 267

Stephan Held. exactcolors, 2022. https://github.com/heldstephan/exactcolors (accessed on August 2024). 169

Stephan Held, William Cook, and Edward C Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4(4):363–381, 2012. doi: 10.1007/s12532-012-0042-3. 169

James K Ho and Etienne Loute. An advanced implementation of the Dantzig—Wolfe decomposition algorithm for linear programming. *Mathematical Programming*, 20(1): 303–326, 1981. doi: 10.1007/BF01589355. 61

Stefan Hougardy and Xianghui Zhong. Hard to solve instances of the Euclidean traveling salesman problem. *Mathematical Programming Computation*, 13(1):51–74, 2021. doi: 10.1007/s12532-020-00184-5. 85

Toshihide Ibaraki and Yuichi Nakamura. A dynamic programming method for single machine scheduling. *European Journal of Operational Research*, 76(1):72–82, 1994. doi: 10.1016/0377-2217(94)90007-8. 259

Irina Ioachim, Jacques Desrosiers, Yvan Dumas, Marius M Solomon, and Daniel Villeneuve. A request clustering algorithm for door-to-door handicapped transportation. *Transportation Science*, 29(1):63–78, 1995. doi: 10.1287/trsc.29.1.63. 198

Stefan Irnich and Daniel Villeneuve. The shortest-path problem with resource constraints and k-cycle elimination for k $\geq$ 3. *INFORMS Journal on Computing*, 18(3):391–406, 2006. doi: 10.1287/ijoc.1040.0117. 149

Stefan Irnich, Guy Desaulniers, Jacques Desrosiers, and Ahmed Hadjar. Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing*, 22(2): 297–313, 2010. doi: 10.1287/ijoc.1090.0341. 246, 258

Brigitte Jaumard, Pierre Hansen, and Marcus Poggi de Aragão. Column generation methods for probabilistic logic. *ORSA Journal on Computing*, 3(2):135–148, 1991. doi: 10.1287/ijoc.3.2.135. 62

Mads Jepsen, Bjørn Petersen, Simon Spoorendonk, and David Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56 (2):497–511, 2008. doi: 10.1287/opre.1070.0449. 180, 200

Hua Jiang and Chu-Min Li. Tsm-mwc, 2017. https://home.mis.u-picardie.fr/~cli/EnglishPage.html (accessed on August 2024). 169

Hua Jiang, Chu-Min Li, Yanli Liu, and Felip Manyà. A two-stage MaxSAT reasoning approach for the maximum weight clique problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, pages 1338–1346, 2018. doi: 10.1609/aaai.v32i1.11527. 169

Ellis L Johnson, Anuj Mehrotra, and George L Nemhauser. Min-cut clustering. *Mathematical programming*, 62:133–151, 1993. doi: 10.1007/BF01585164. 198

Kim L Jones, Irvin J Lustig, Judith M Farvolden, and Warren B Powell. Multicommodity network flows: The impact of formulation on decomposition. *Mathematical Programming*, 62:95–117, 1993. doi: 10.1007/BF01585162. 59

Michael Jünger, Thomas M Liebling, Denis Naddef, George L Nemhauser, William R Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A Wolsey, editors. *50 Years of Integer Programming 1958-2008: From the early years to the state-of-the-art*. Springer, 2010. doi: 10.1007/978-3-540-68279-0. 88

Brian Kallehauge, Jesper Larsen, and Oli B G Madsen. Lagrangian duality applied to the vehicle routing problem with time windows. *Computers & Operations Research*, 33(5): 1464–1487, 2006. doi: 10.1016/j.cor.2004.11.002. 180

Leonid V Kantorovich. *Matematicheskie metody organizatsii i planirovaniya proizvodstva [Mathematical methods of production organization and planning]*. Lenizdat, Leningrad, 1939. 142, 143, 170, 172, 173, 176, 203

Leonid V Kantorovich. Mathematical methods of organizing and planning production. *Management science*, 6(4):366–422, 1960. doi: 10.1287/mnsc.6.4.366. 170, 176

Leonid V Kantorovich and Victor A Zalgaller. Calculation of rational cutting of stock. *Lenizdat, Leningrad*, 5:11–14, 1951. 172, 173, 175

Nejat Karabakal, James Bean, and Jack R Lohmann. A steepest descent multiplier adjustment method for the generalized assignment problem. Technical Report 92-11, University of Michigan, 1992. https://deepblue.lib.umich.edu/bitstream/handle/2027.42/5870/ban1158.0001.001.pdf (accessed in August 2024). 245, 258

Vadim M Kartak, Artem V Ripatti, Guntram Scheithauer, and Sascha Kurz. Minimal proper non-IRUP instances of the one-dimensional cutting stock problem. *Discrete Applied Mathematics*, 187:120–129, 2015. doi: 10.1016/j.dam.2015.02.020. 144

Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004. doi: 10.1007/978-3-540-24777-7. 167

James E Kelley, Jr. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960. doi: 10.1137/0108053. 49, 233, 236

Leonid Genrikhovich Khachiyan. A polynomial algorithm in linear programming. In *Doklady Akademii Nauk*, volume 244(5), pages 1093–1096. Russian Academy of Sciences, 1979. 103

Taghi Khaniyev, Samir Elhedhli, and Fatih Safa Erenay. Structure detection in mixed-integer programs. *INFORMS Journal on Computing*, 30(3):570–587, 2018. doi: 10.1287/ijoc.2017.0797. 160

Daeki Kim, Cynthia Barnhart, Keith Ware, and Gregory Reinhardt. Multimodal express package delivery: A service network design application. *Transportation science*, 33(4): 391–407, 1999. doi: 10.1287/trsc.33.4.391. 198

Victor Klee and George J Minty. How good is the simplex algorithm? In Oved Shisha, editor, *Inequalities*, volume 3(3), pages 159–175. Academic Press, 1972. 13

Thorsten Koch and Alexander Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32(3):207–232, 1998. doi: 10.1002/(SICI)1097-0037(199810)32:3¡207::AID-NET5¿3.0.CO;2-O. 108

Thorsten Koch, Timo Berthold, Jaap Pedersen, and Charlie Vanaret. Progress in mathematical programming solvers from 2001 to 2020. *EURO Journal on Computational Optimization*, 10:100031, 2022. doi: 10.1016/j.ejco.2022.100031. 90

Niklas Kohl, Jacques Desrosiers, Oli B G Madsen, Marius M Solomon, and François Soumis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116, 1999. doi: 10.1287/trsc.33.1.101. 180, 198

Viviane Köhler, Marcia Fampa, and Olinto Araújo. Mixed-integer linear programming formulations for the software clustering problem. *Computational Optimization and Applications*, 55:113–135, 2013. doi: 10.1007/s10589-012-9512-9. 156, 157, 158

Péter Kovács. Minimum-cost flow algorithms: an experimental evaluation. *Optimization Methods and Software*, 30(1):94–127, 2015. doi: 10.1080/10556788.2014.895828. 37

Hugo Harry Kramer, Eduardo Uchoa, Marcia Fampa, Viviane Köhler, and François Vanderbeck. Column generation approaches for the software clustering problem. *Computational Optimization and Applications*, 64:843–864, 2016. doi: 10.1007/s10589-015-9822-9. 157, 158, 159

Markus Kruber, Marco E Lübbecke, and Axel Parmentier. Learning when to use a decomposition. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming*, volume 10335 of *Lecture Notes in Computer Science*, pages 202–210. Springer, 2017. doi: 10.1007/978-3-319-59776-8_16. 160

Simge Küçükyavuz and Suvrajeet Sen. An introduction to two-stage stochastic mixed-integer programming. In *Leading Developments from INFORMS Communities*, INFORMS TutORials in Operations Research, pages 1–27. INFORMS, 2017. doi: 10.1287/educ.2017.0171. 185

Joseph-Louis Lagrange. Leçons sur le calcul des fonctions. In *Journal de l'École Polytechnique*, volume 12. Conseil d'Instrution et administration de l'École Polytechnique, 1806. https://gallica.bnf.fr/ark:/12148/bpt6k86261t/ (accessed on August 2024). 207, 262

François Lamothe, Alain Haït, Emmanuel Rachelson, Claudio Contardo, and Bernard Gendron. On the integration of Dantzig–Wolfe and Fenchel decompositions via directional normalizations. arXiv, 2023. doi: 10.48550/arXiv.2303.15573. 189

Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960. doi: 10.2307/1910129. 72, 88

Gilbert Laporte and Yves Nobert. A branch and bound algorithm for the capacitated vehicle routing problem. *Operations-Research-Spektrum*, 5(2):77–85, 1983. doi: 10.1007/BF01720015. 86

Torbjörn Larsson, Michael Patriksson, and Ann-Brith Strömberg. Ergodic, primal convergence in dual subgradient schemes for convex programming. *Mathematical programming*, 86:283–312, 1999. doi: 10.1007/s101070050090. 263

Claude Lemaréchal. An extension of Davidon methods to non differentiable problems. In Michel L. Balinski and Philip Wolfe, editors, *Nondifferentiable Optimization*, volume 3 of *Mathematical Programming Studies*, pages 95–109. Springer Berlin Heidelberg, 1975. ISBN 978-3-642-00764-4. doi: 10.1007/BFb0120700. 231

Claude Lemaréchal. Lagrangian relaxation. In Michael Jünger and Denis Naddef, editors, *Computational Combinatorial Optimization: optimal or provably near-optimal solutions*, volume 2241 of *Lecture Notes in Computer Science*, pages 112–156. Springer, 2001. doi: 10.1007/3-540-45586-8_4. 247, 262, 263

Claude Lemaréchal, Arkadi Nemirovski, and Yurii Nesterov. New variants of bundle methods. *Mathematical programming*, 69:111–147, 1995. doi: 10.1007/BF01585555. 190

Adam N Letchford. Personal communication, 2005. 200

Adam N Letchford and Juan-José Salazar-González. Projection results for vehicle routing. *Mathematical Programming*, 105:251–274, 2006. doi: 10.1007/s10107-005-0652-x. 199

Amos Levin. Some fleet routing and scheduling problems for air transportation systems. Technical Report 5, Massachusetts Institute of Technology, Flight Transportation Laboratory, 1968. https://dspace.mit.edu/handle/1721.1/68120 (accessed on August 2024). 196

Benedikt Lienkamp and Maximilian Schiffer. Column generation for solving large scale multi-commodity flow problems for passenger transportation. *European Journal of Operational Research*, 314(2):703–717, 2024. doi: 10.1016/j.ejor.2023.09.019. 46, 47, 48

Pedro Henrique Liguori, Ali Ridha Mahjoub, Guillaume Marques, Ruslan Sadykov, and Eduardo Uchoa. Nonrobust strong knapsack cuts for capacitated location routing and related problems. *Operations Research*, 71(5):1577–1595, 2023. doi: 10.1287/opre.2023.2458. 181

John DC Little, Katta G Murty, Dura W Sweeney, and Caroline Karel. An algorithm for the traveling salesman problem. *Operations Research*, 11(6):972–989, 1963. doi: 10.1287/opre.11.6.972. 88

Ivana Ljubić. Solving Steiner trees: Recent advances, challenges, and perspectives. *Networks*, 77(2):177–204, 2021. doi: 10.1002/net.22005. 108

Andrea Lodi. Mixed integer programming computation. In Michael Jünger, Thomas M Liebling, Denis Naddef, George L Nemhauser, William R Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A Wolsey, editors, *50 Years of Integer Programming 1958-2008: From the early years to the state-of-the-art*, pages 619–645. Springer, 2010. doi: 10.1007/978-3-540-68279-0_16. 90

Kok-Hua Loh, Bruce Golden, and Edward Wasil. Solving the one-dimensional bin packing problem with a weight annealing heuristic. *Computers & Operations Research*, 35(7): 2283–2291, 2008. doi: 10.1016/j.cor.2006.10.021. 178

Humberto Longo. *Técnicas para programação inteira e aplicações em problemas de roteamento de veículos*. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2004. https://www.maxwell.vrac.puc-rio.br/6029/6029_1.PDF (accessed on August 2024). 193

Marco E Lübbecke. Column generation. In James J Cochran, Louis A Cox, Pinar Keskinocak, Jeffrey P Kharoufeh, and J Cole Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. Wiley New York, 2011. doi: 10.1002/9780470400531.eorms0158. 163

Abilio Lucena. Steiner problem in graphs: Lagrangean relaxation and cutting planes. In *COAL Committee on Algorithms Bulletin*, volume 21, pages 2–7. Mathematical Optimization Society, 1992. https://www.mathopt.org/COAL-Issues/COAL21.pdf (accessed on August 2024). 107, 265

Abilio Lucena. Non delayed relax-and-cut algorithms. *Annals of Operations Research*, 140: 375–410, 2005. doi: 10.1007/s10479-005-3977-1. 265

Jens Lysgaard. *CVRPSEP: A package of separation routines for the capacitated vehicle routing problem*. Aarhus School of Business, Department of Management Science and Logistics, 2003. https://github.com/sassoftware/cvrpsep (accessed on August 2024). 86

Jens Lysgaard, Adam N Letchford, and Richard W Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100:423–445, 2004. doi: 10.1007/s10107-003-0481-8. 87, 100, 179, 199

Thomas L Magnanti, Jeremy F Shapiro, and Michael H Wagner. Generalized linear programming solves the dual. *Management science*, 22(11):1195–1203, 1976. doi: 10.1287/mnsc.22.11.1195. 163, 216

Stephen J Maher. Implementing the branch-and-cut approach for a general purpose Benders' decomposition framework. *European Journal of Operational Research*, 290(2):479–498, 2021. doi: 10.1016/j.ejor.2020.08.037. 184, 185

Odile Marcotte. The cutting stock problem and integer rounding. *Mathematical Programming*, 33(1):82–92, 1985. doi: 10.1007/BF01582013. 144

Guillaume Marques, Ruslan Sadykov, Jean-Christophe Deschamps, and Rémy Dupas. An improved branch-cut-and-price algorithm for the two-echelon capacitated vehicle routing problem. *Computers & Operations Research*, 114:104833, 2020. doi: 10.1016/j.cor.2019.104833. 181

Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., 1990. 167

Richard Kipp Martin. Generating alternative mixed-integer programming models using variable redefinition. *Operations Research*, 35(6):820–831, 1987. doi: 10.1287/opre.35.6.820. 109

Anuj Mehrotra and Michael A Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996. doi: 10.1287/ijoc.8.4.344. 150, 198, 204

Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960. doi: 10.1145/321043.321046. 100

Brian Scott Mitchell. *A heuristic search approach to solving the software clustering problem*. PhD thesis, Drexel University, 2002. https://www.proquest.com/openview/36ec98a6c9e484c32c9f9b6c293e93c7 (accessed on August 2024). 156

Irmen Ben Mohamed, Walid Klibi, Ruslan Sadykov, Halil Şen, and François Vanderbeck. The two-echelon stochastic multi-period capacitated location-routing problem. *European Journal of Operational Research*, 306(2):645–667, 2023. doi: 10.1016/j.ejor.2022.07.022. 181

Denis Naddef and Giovanni Rinaldi. Branch-and-cut algorithms for the capacitated VRP. In Paolo Toth and Daniele Vigo, editors, *The vehicle routing problem*, pages 53–84. SIAM, 2002. doi: 10.1137/1.9780898718515.ch3. 86

Robert M Nauss. Solving the generalized assignment problem: An optimizing and heuristic approach. *INFORMS Journal on Computing*, 15(3):249–266, 2003. doi: 10.1287/ijoc.15.3.249.16075. 170, 204

George L Nemhauser and Sungsoo Park. A polyhedral approach to edge coloring. *Operations Research Letters*, 10(6):315–322, 1991. doi: 10.1016/0167-6377(91)90003-8. 198

George L Nemhauser and Laurence A Wolsey. *Integer and Combinatorial Optimization*. Wiley Interscience Series in Discrete Mathematics and Optimization. Wiley, 1988. ISBN 978-0-471-82819-8. doi: 10.1002/9781118627372. 71, 102

Nils J Nilsson. Probabilistic logic. *Artificial intelligence*, 28(1):71–87, 1986. doi: 10.1016/0004-3702(86)90031-7. 62

Sampo Niskanen and Patric R J Östergård. Cliquer - routines for clique searching, 2010. https://users.aalto.fi/~pat/cliquer.html (accessed on August 2024). 169

Daniel Oliveira and Artur Pessoa. An improved branch-cut-and-price algorithm for parallel machine scheduling problems. *INFORMS Journal on Computing*, 32(1):90–100, 2020. doi: 10.1287/ijoc.2018.0854. 261

Welington de Oliveira and Claudia Sagastizábal. Bundle methods in the XXIst century: A bird's-eye view. *Pesquisa Operacional*, 34(3):647–670, 2014. doi: 10.1590/0101-7438.2014.034.03.0647. 262

Patric R J Östergård. A new algorithm for the maximum-weight clique problem. *Nordic Journal of Computing*, 8(4):424–436, 2001. 169

Manfred W Padberg. On the facial structure of set packing polyhedra. *Mathematical programming*, 5:199–215, 1973. doi: 10.1007/BF01580121. 85

Yunpeng Pan and Leyuan Shi. On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems. *Mathematical Programming*, 110:543–559, 2007. doi: 10.1007/s10107-006-0013-4. 254, 259, 260

Mark Parker and Jennifer Ryan. A column generation algorithm for bandwidth packing. *Telecommunication Systems*, 2:185–195, 1993. doi: 10.1007/BF02109857. 198

David Lorge Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972. doi: 10.1145/361598.361623. 155

Diego Pecin, Claudio Contardo, Guy Desaulniers, and Eduardo Uchoa. New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 29(3):489–502, 2017a. doi: 10.1287/ijoc.2016.0744. 180

Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9:61–100, 2017b. doi: 10.1007/s12532-016-0108-8. 178, 246

Artur Pessoa, Eduardo Uchoa, Marcus Poggi de Aragão, and Rosiane Rodrigues. Algorithms over arc-time indexed formulations for single and parallel machine scheduling problems. Technical Report 8, Universidade Federal Fluminense, 2008. https://optimization-online.org/wp-content/uploads/2008/06/2022.pdf (accessed on August 2024). 259, 260

Artur Pessoa, Eduardo Uchoa, Marcus Poggi de Aragão, and Rosiane Rodrigues. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation*, 2:259–290, 2010. doi: 10.1007/s12532-010-0019-z. 200, 246, 255, 257, 258, 259, 260, 261, 265

Artur Pessoa, Ruslan Sadykov, and Eduardo Uchoa. Enhanced branch-cut-and-price algorithm for heterogeneous fleet vehicle routing problems. *European Journal of Operational Research*, 270:530–543, 2018. doi: 10.1016/j.ejor.2018.04.009. 181

Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, 183(1):483–523, 2020. doi: 10.1007/s10107-020-01523-z. 170, 180, 181, 186, 246

Artur Pessoa, Michael Poss, Ruslan Sadykov, and François Vanderbeck. Branch-cut-and-price for the robust capacitated vehicle routing problem with knapsack uncertainty. *Operations Research*, 69(3):739–754, 2021a. doi: 10.1287/opre.2020.2035. 181

Artur Pessoa, Ruslan Sadykov, and Eduardo Uchoa. Solving bin packing problems using VRPSolver models. In *Operations Research Forum*, volume 2(20), pages 1–25. Springer, 2021b. doi: 10.1007/s43069-020-00047-8. 178

Alexandre Pigatti, Marcus Poggi de Aragão, and Eduardo Uchoa. Stabilized branch-and-cut-and-price for the generalized assignment problem. *Electronic Notes in Discrete Mathematics*, 19:389–395, 2005. doi: 10.1016/j.endm.2005.05.052. 170, 246, 248, 249

David Pisinger. Knapsack problems. http://hjemmesider.diku.dk/~pisinger/codes.html (accessed on August 2024). 167

David Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284, 2005. doi: 10.1016/j.cor.2004.03.002. 169

Marcus Poggi and Eduardo Uchoa. New exact algorithms for the capacitated vehicle routing problem. In Paolo Toth and Daniele Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, MOS-SIAM Series on Optimization, pages 59–86. SIAM, 2014. doi: 10.1137/1.9781611973594.ch3. 181

Marcus Poggi de Aragão and Eduardo Uchoa. Integer program reformulation for robust branch-and-cut-and-price algorithms. In Laurence Wolsey, editor, *Proceedings of Mathematical Programming in Rio: a conference in honour of Nelson Maculan*, pages 56–61. 2003. https://www.inf.puc-rio.br/~uchoa/doc/rbcp-a.pdf (accessed on August 2024). 56, 193, 199

Marcus Poggi de Aragão, Eduardo Uchoa, and Renato F Werneck. Dual heuristics on the exact solution of large Steiner problems. *Electronic Notes in Discrete Mathematics*, 7: 150–153, 2001. doi: 10.1016/S1571-0653(04)00247-1. 108

Lukas Polten and Simon Emde. Multi-shuttle crane scheduling in automated storage and retrieval systems. *European Journal of Operational Research*, 302(3):892–908, 2022. doi: 10.1016/j.ejor.2022.01.043. 181

Boris T Polyak. Subgradient methods: a survey of Soviet research. In Claude Lemaréchal and Robert Mifflin, editors, *Nonsmooth optimization, Proceedings of a IIASA Workshop*, volume 3, pages 5–29. Pergamon Press, 1978. https://pure.iiasa.ac.at/id/eprint/819/1/XB-78-504.pdf (accessed on August 2024). 262

Boris T Polyak. History of mathematical programming in the USSR: analyzing the phenomenon. *Mathematical Programming*, 91(3):401–416, 2002. doi: 10.1007/s101070100258. 172

Tobias Polzin and Siavash Vahdati Daneshmand. Improved algorithms for the Steiner problem in networks. *Discrete Applied Mathematics*, 112:263–300, 2001. ISSN 0166-218X. doi: 10.1016/S0166-218X(00)00319-X. 108

Tobias Polzin and Siavash Vahdati Daneshmand. Approaches to the Steiner problem in networks. In Jürgen Lerner, Dorothea Wagner, and Katharina A Zweig, editors, *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 81–103. Springer, 2009. doi: 10.1007/978-3-642-02094-0_5. 108

Marius Posta, Jacques A Ferland, and Philippe Michelon. An exact method with variable fixing for solving the generalized assignment problem. *Computational Optimization and Applications*, 52(3):629–644, 2012. doi: 10.1007/s10589-011-9432-0. 170, 244, 246, 250, 263

Rafael Praxedes, Teobaldo Bulhões, Anand Subramanian, and Eduardo Uchoa. A unified exact approach for a broad class of vehicle routing problems with simultaneous pickup and delivery. *Computers & Operations Research*, 162:106467, 2024. doi: 10.1016/j.cor.2023.106467. 181

Eduardo Queiroga, Yuri Frota, Ruslan Sadykov, Anand Subramanian, Eduardo Uchoa, and Thibaut Vidal. On the exact solution of vehicle routing problems with backhauls. *European Journal of Operational Research*, 287(1):76–89, 2020. doi: 10.1016/j.ejor.2020.04.047. 181

Marcela Quiroz-Castellanos, Laura Cruz-Reyes, Jose Torres-Jimenez, Claudia Gómez-Santillan, Héctor J Fraire Huacuja, and Adriana C F Alvim. A grouping genetic algorithm with controlled gene transmission for the bin packing problem. *Computers & Operations Research*, 55:52–64, 2015. doi: 10.1016/j.cor.2014.10.010. 178

Samuel Raff. Routing and scheduling of vehicles and crews: The state of the art. *Computers & Operations Research*, 10(2):63–211, 1983. doi: 10.1016/0305-0548(83)90030-8. 197

Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817, 2017. doi: 10.1016/j.ejor.2016.12.005. 184

Ted K Ralphs. Parallel branch and cut for capacitated vehicle routing. *Parallel Computing*, 29(5):607–629, 2003. doi: 10.1016/S0167-8191(03)00045-0. 87

Ted K Ralphs and Matthew V Galati. Decomposition in integer linear programming. In John K Karlof, editor, *Integer Programming: Theory and Practice*, pages 73–126. CRC Press, 1st edition, 2005. doi: 10.1201/9781420039597. 189, 265

Ted K Ralphs, Leonid Kopman, William R Pulleyblank, and Leslie Earl Trotter Jr. On the capacitated vehicle routing problem. *Mathematical Programming*, 94:343–359, 2003. doi: 10.1007/s10107-002-0323-0. 87

Mendu Rammohan Rao and Stanle Zionts. Allocation of transportation units to alternative trips — a column generation scheme with out-of-kilter subproblems. *Operations Research*, 16(1):52–63, 1968. doi: 10.1287/opre.16.1.52. 196

Steffen Rebennack, Marcus Oswald, Dirk Oliver Theis, Hanna Seitz, Gerhard Reinelt, and Panos M Pardalos. A branch and cut solver for the maximum stable set problem. *Journal of combinatorial optimization*, 21:434–457, 2011. doi: 10.1007/s10878-009-9264-3. 85

Daniel Rehfeldt and Thorsten Koch. Implications, conflicts, and reductions for Steiner trees. *Mathematical Programming*, 197:903–966, 2021. doi: 10.1007/s10107-021-01757-5. 108

Gerhard Reinelt. TSPLIB — a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991. doi: 10.1287/ijoc.3.4.376. 104

Celso C Ribeiro and François Soumis. A column generation approach to the multiple-depot vehicle scheduling problem. *Operations Research*, 42(1):41–52, 1994. doi: 10.1287/opre.42.1.41. 198

Joseph Rios. Algorithm 928: A general, parallel implementation of Dantzig–Wolfe decomposition. *ACM Transactions on Mathematical Software*, 39(3), 2013. doi: 10.1145/2450153.2450159. 62

Joseph Rios and Kevin Ross. Massively parallel Dantzig-Wolfe decomposition applied to traffic flow scheduling. *Journal of Aerospace Computing, Information, and Communication*, 7(1):32–45, 2010. doi: 10.2514/1.45606. 62

Roberto Roberti and Aristide Mingozzi. Dynamic ng-path relaxation for the delivery man problem. *Transportation Science*, 48(3):413–424, 2014. doi: 10.1287/trsc.2013.0474. 196

Marcos Costa Roboredo, Ruslan Sadykov, and Eduardo Uchoa. Solving vehicle routing problems with intermediate stops using VRPSolver models. *Networks*, 81(3):399–416, 2023. doi: 10.1002/net.22137. 181

Stefan Røpke. Branching decisions in branch-and-cut-and-price algorithms for vehicle routing problems. Presentation at International Workshop on Column Generation 2012, 2012. https://www.gerad.ca/colloques/ColumnGeneration2012/presentations/session7/Ropke.pdf (accessed on August 2024). 178

Judah Ben Rosen and Robert S Maier. Parallel solution of large-scale, block-angular linear programs. *Annals of Operations Research*, 22:23–41, 1990. doi: 10.1007/BF02023046. 61

Fabrizio Rossi and Stefano Smriglio. A branch-and-cut algorithm for the maximum cardinality stable set problem. *Operations Research Letters*, 28(2):63–74, 2001. doi: 10.1016/S0167-6377(00)00060-2. 85

Francesca Rossi, Peter Van Beek, and Toby Walsh, editors. *Handbook of constraint programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. https://shop.elsevier.com/books/handbook-of-constraint-programming/rossi/978-0-444-52726-4 (accessed on August 2024). 91

Thomas Rothvoß. The matching polytope has exponential extension complexity. *Journal of the ACM*, 64(6):1–19, 2017. doi: 10.1145/3127497. 104

Alexander M. Rubinov, editor. *Congratulations to Naum Shor on his 65th birthday. Journal of Global Optimization*, volume 24. Springer, 2002. doi: 10.1023/A:1020215832722. 262

David M Ryan and Brian A Foster. An integer programming approach to scheduling. In Anthony Wren, editor, *Computer scheduling of public transport: urban passenger vehicle and crew scheduling*, pages 269–280. North-Holland, Amsterdam, 1981. 99, 151

Ruslan Sadykov. *Modern branch-cut-and-price*. HDR thesis, Université de Bordeaux, 2019. https://inria.hal.science/tel-02410101/ (accessed on August 2024). 163

Ruslan Sadykov and François Vanderbeck. Column generation for extended formulations. *EURO Journal on Computational Optimization*, 1(1–2):81–115, 2013. doi: 10.1007/s13675-013-0009-9. 109

Ruslan Sadykov and François Vanderbeck. BaPCod - a generic branch-and-price code. Technical Report HAL-03340548, Inria Bordeaux Sud-Ouest, 2021. https://inria.hal.science/hal-03340548/ (accessed on August 2024). 181

Ruslan Sadykov, Eduardo Uchoa, and Artur Pessoa. A bucket graph–based labeling algorithm with application to vehicle routing. *Transportation Science*, 55(1):4–28, 2021. doi: 10.1287/trsc.2020.0985. 180, 181, 246

Khodakaram Salimifard and Sara Bigharaz. The multicommodity network flow problem: state of the art classification, applications, and solution methods. *Operational Research*, 22:1–47, 2022. doi: 10.1007/s12351-020-00564-8. 45

Pablo San Segundo, Fabio Furini, and Jorge Artieda. A new branch-and-bound algorithm for the maximum weighted clique problem. *Computers & Operations Research*, 110:18–33, 2019. doi: 10.1016/j.cor.2019.05.017. 169

Haroldo Gambini Santos, Eduardo Uchoa, Luiz Satoru Ochi, and Nelson Maculan. Strong bounds with cut and column generation for class-teacher timetabling. *Annals of Operations Research*, 194:399–412, 2012. doi: 10.1007/s10479-010-0709-y. 188

Martin W P Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45(6):831–841, 1997. doi: 10.1287/opre.45.6.831. 170, 198

Lara Scavuzzo, Karen Aardal, Andrea Lodi, and Neil Yorke-Smith. Machine learning augmented branch and bound for mixed integer linear programming. *Mathematical Programming*, (to appear), 2024. doi: 10.1007/s10107-024-02130-y. 98

Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1986. ISBN 978-0-471-98232-6. 51, 71, 105

Ron Shamir. The efficiency of the simplex method: a survey. *Management science*, 33(3): 301–334, 1987. doi: 10.1287/mnsc.33.3.301. 13

Naum Zuselevich Shor. Application of the gradient descent method to network transportation problem *(in Russian)*. In *Economic Cybernetics and Operations Research Seminar*, volume 1 of *Materials of scientific seminars on theoretical and applied issues of cybernetics*, pages 9–17. Scientific Council on Cybernetics of the Academy of Sciences of the Ukrainian, 1962. National Library of Ukraine (accessed on August 2024). 262

Naum Zuselevich Shor. *Minimization methods for non-differentiable functions*, volume 3 of *Springer Series in Computational Mathematics*. Springer Verlag, 1985. doi: 10.1007/978-3-642-82118-9. 263

João Marcos Pereira Silva, Eduardo Uchoa, and Anand Subramanian. Cluster branching for vehicle routing problems. Technical Report L-2024-2, Cadernos do LOGIS-UFF, Universidade Federal Fluminense, Engenharia de Produção, 2024. https://optimization-online.org/2024/07/cluster-branching-for-vehicle-routing-problems/ (accessed on August 2024). 100

Luidi Simonetti, Nelson Maculan, Ana Flávia Uzeda Macambira, and Pedro Henrique González. Geração de colunas em programação inteira. In Ana Flávia Uzeda Macambira, Luigi Simonetti, Rosiane de Freitas Rodrigues, and Nelson Maculan, editors, *Tópicos em Otimização Inteira*. Editora UFRJ - Universidade Federal do Rio de Janeiro, 2022. http://hdl.handle.net/11422/19343 (accessed on August 2024). 163

Robert W Simpson. Scheduling and routing models for airline systems. Technical Report R68-3, Massachusetts Institute of Technology, Flight Transportation Laboratory, 1969. https://dspace.mit.edu/handle/1721.1/67999 (accessed on August 2024). 196

Vinícius Carvalho Soares and Marcos Costa Roboredo. On the exact solution of the multi-depot open vehicle routing problem. *Optimization Letters*, 18(4):1053–1069, 2024. doi: 10.1007/s11590-023-02072-y. 181

Francis Sourd. New exact algorithms for one-machine earliness-tardiness scheduling. *INFORMS Journal on Computing*, 21(1):167–175, 2009. doi: 10.1287/ijoc.1080.0287. 255

Anirudh Subramanyam, Taner Cokyasar, Jeffrey Larson, and Monique Stinson. Joint routing of conventional and range-extended electric vehicles in a large metropolitan network. *Transportation Research Part C: Emerging Technologies*, 144:103830, 2022. doi: 10.1016/j.trc.2022.103830. 181

Shunji Tanaka. SiPS - single-machine scheduling problem solver / SiPSi - single-machine scheduling problem solver with idle time, 2016. https://sites.google.com/site/shunjitanaka/sips (accessed on August 2024). 260

Shunji Tanaka and Mituhiko Araki. A branch-and-bound algorithm with lagrangian relaxation to minimize total tardiness on identical parallel machines. *International Journal of Production Economics*, 113(1):446–458, 2008. doi: 10.1016/j.ijpe.2007.10.006. 254

Shunji Tanaka and Mituhiko Araki. An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times. *Computers & Operations Research*, 40(1):344–352, 2013. doi: 10.1016/j.cor.2012.07.004. 260

Shunji Tanaka and Shuji Fujikuma. A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time. *Journal of Scheduling*, 15: 347–361, 2012. doi: 10.1007/s10951-011-0242-0. 260

Shunji Tanaka, Shuji Fujikuma, and Mituhiko Araki. An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling*, 12:575–593, 2009. doi: 10.1007/s10951-008-0093-5. 246, 255, 258, 259, 260, 261

James Richard Tebboth. *A computational study of Dantzig-Wolfe decomposition*. PhD thesis, University of Buckingham, 2001. 62

Eduardo Uchoa. Cuts over extended formulations by flow discretization. In Ali Ridha Mahjoub, editor, *Progress in Combinatorial Optimization: Recent Progress*, pages 255–282. ISTE-Wiley, 2011. https://www.wiley.com/en-br/Progress+in+Combinatorial+Optimization%3A+Recent+Progress-p-9781848212060 (accessed on August 2024). 109

Eduardo Uchoa and Ruslan Sadykov. Kantorovich and Zalgaller (1951): the 0-th column generation algorithm. Technical Report L-2024-1, Cadernos do LOGIS-UFF, Niterói, Brazil, January 2024. https://optimization-online.org/2024/01/kantorovich-and-zalgaller-1951-the-0-th-column-generation-algorithm/ (accessed on August 2024). 172, 175

Eduardo Uchoa, Marcus Poggi de Aragão, and Celso C Ribeiro. Preprocessing Steiner problems from VLSI layout. *Networks*, 40(1):38–50, 2002. doi: 10.1002/net.10035. 108

Eduardo Uchoa, Ricardo Fukasawa, Jens Lysgaard, Artur Pessoa, Marcus Poggi de Aragão, and Diogo Andrade. Robust branch-cut-and-price for the capacitated minimum spanning tree problem over a large extended formulation. *Mathematical Programming*, 112:443–472, 2008. doi: 10.1007/s10107-006-0043-y. 200, 260

Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, Anand Subramanian, Ivan Xavier Lima, Daniel Oliveira, and Eduardo Queiroga. CVRPLib - Capacitated Vehicle Routing Problem Library, 2014. http://vrp.galgos.inf.puc-rio.br/index.php/en/ (accessed on August 2024). 181

José M Valério de Carvalho. Exact solution of cutting stock problems using column generation and branch-and-bound. *International Transactions in Operational Research*, 5(1): 35–44, 1998. doi: 10.1016/S0969-6016(97)00044-0. 154

José M Valério de Carvalho. Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86:629–659, 1999. doi: 10.1023/A:1018952112615. 178

José M Valério de Carvalho. LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253–273, 2002. doi: 10.1016/S0377-2217(02)00124-8. 109

Janna Magrietje van den Akker, Constantinus P M van Hoesel, and Martin W P Savelsbergh. A polyhedral approach to single-machine scheduling problems. *Mathematical Programming*, 85:541–572, 1999. doi: 10.1007/s10107990047a. 254

Janna Magrietje van den Akker, Cor AJ Hurkens, and Martin W P Savelsbergh. Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12(2):111–124, 2000. doi: 10.1287/ijoc.12.2.111.11896. 198, 253, 254

Pamela H Vance. Branch-and-price algorithms for the one-dimensional cutting stock problem. *Computational optimization and applications*, 9(3):211–228, 1998. doi: 10.1023/A:1018346107246. 178, 198

Pamela H Vance, Cynthia Barnhart, Ellis L Johnson, and George L Nemhauser. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational optimization and applications*, 3(2):111–130, 1994. doi: 10.1007/BF01300970. 152, 153, 178, 198

François Vanderbeck. Lot-sizing with start-up times. *Management Science*, 44(10):1409–1425, 1998. doi: 10.1287/mnsc.44.10.1409. 198

François Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86(3):565–594, 1999. doi: 10.1007/s101070050105. 178

François Vanderbeck and Martin W P Savelsbergh. A generic view of Dantzig–Wolfe decomposition in mixed integer programming. *Operations Research Letters*, 34(3):296–306, 2006. doi: 10.1016/j.orl.2005.05.009. 163, 164

François Vanderbeck and Laurence A Wolsey. Reformulation and decomposition of integer programs. In Michael Jünger, Thomas M Liebling, Denis Naddef, George L Nemhauser, William R Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A Wolsey, editors, *50 Years of Integer Programming 1958-2008: From the early years to the state-of-the-art*, pages 431–502. Springer, 2010. doi: 10.1007/978-3-540-68279-0_13. 163

Robert J Vanderbei. *Linear Programming: Foundations and Extensions*. International Series in Operations Research & Management Science. Springer, 2014. doi: 10.1007/978-3-030-39415-8. 3

Eleonora Vercesi, Stefano Gualandi, Monaldo Mastrolilli, and Luca Maria Gambardella. On the generation of metric TSP instances with a large integrality gap by branch-and-cut. *Mathematical Programming Computation*, 15:389—-416, 2023. doi: 10.1007/s12532-023-00235-7. 85

Anatoly Vershik. L. V. Kantorovich and linear programming, 2007. doi: 10.48550/ARXIV.0707.0491. 172

Gabriel Volte, Eric Bourreau, Rodolphe Giroudeau, and Olivier Naud. Using VRPSolver to efficiently solve the differential harvest problem. *Computers & Operations Research*, 149:106029, 2023. doi: 10.1016/j.cor.2022.106029. 181

VRPSolver. Generic exact solver for vehicle routing and related problems. https://vrpsolver.math.u-bordeaux.fr (accessed on August 2024). 181

VRPSolverEasy. https://vrpsolvereasy.readthedocs.io (accessed on August 2024). 181

Lijun Wei, Zhixing Luo, Roberto Baldacci, and Andrew Lim. A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. *INFORMS Journal on Computing*, 32(2):428–443, 2020. doi: 10.1287/ijoc.2018.0867. 178

Peng Wei, Yi Cao, and Dengfeng Sun. Total unimodularity and decomposition method for large-scale air traffic cell transmission model. *Transportation Research Part B: Methodological*, 53:1–16, 2013. doi: 10.1016/j.trb.2013.03.004. 62

Klaus Wenger. *Generic Cut Generation Methods for Routing Problems*. PhD thesis, University of Heidelberg, Institute of Computer Science, 2003. https://www.shaker.de/de/content/catalogue/index.asp?ISBN=978-3-8322-2545-2 (accessed on August 2024). 87

Paul Wentges. Weighted Dantzig-Wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research*, 4(2):151–162, 1997. doi: 10.1016/S0969-6016(97)00001-4. 260

Wilbert E Wilhelm. A technical review of column generation in integer programming. *Optimization and Engineering*, 2:159–200, 2001. doi: 10.1023/A:1013141227104. 163

H Paul Williams. *Model building in mathematical programming*. John Wiley & Sons, 1st edition, 1978. ISBN 978-0471995418. 100

H Paul Williams. *Model building in mathematical programming*. John Wiley & Sons, 5th edition, 2013. ISBN 978-1-118-50617-2. 100

Philip Wolfe. A method of conjugate subgradients for minimizing nondifferentiable functions. In Michel L. Balinski and Philip Wolfe, editors, *Nondifferentiable Optimization*, volume 3 of *Mathematical Programming Studies*, pages 145–173. Springer Berlin Heidelberg, 1975. ISBN 978-3-642-00764-4. doi: 10.1007/BFb0120703. 231

Philip Wolfe. Finding the nearest point in a polytope. *Mathematical Programming*, 11: 128–149, 1976. doi: 10.1007/BF01580381. 232

Laurence A Wolsey. *Integer programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1st edition, 1998. ISBN 978-0471283669. 71

Laurence A Wolsey. *Integer programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 2nd edition, 2020. ISBN 9781119606536. doi: 10.1002/9781119606475. 71, 163, 170, 184, 262

Richard T Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28:271–287, 1984. doi: 10.1007/BF02612335. 107

Xpress. FICO Xpress Optimization. https://www.fico.com/en/products/fico-xpress-optimization (accessed on August 2024). 91

Takeo Yamada, Seija Kataoka, and Kohtaro Watanabe. Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *Journal of the Information Processing Society, Japan*, 43(9):2864–2870, 2002. http://id.nii.ac.jp/1001/00011486/ (accessed on August 2024). 153

Yu Yang. Deluxing: Deep lagrangian underestimate fixing for column-generation-based exact methods. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4585724 (accessed on August 2024), 2024. 246

Mihalis Yannakakis. Expressing combinatorial optimization problems by linear programs. *Journal of Computer and System Sciences*, 43(3):441–466, 1991. doi: 10.1016/0022-0000(91)90024-Y. 104

Tallys Yunes, Ionuţ D Aron, and John N Hooker. An integrated solver for optimization problems. *Operations Research*, 58(2):342–356, 2010. doi: 10.1287/opre.1090.0733. 92