REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
07/31/2011	Final report	05/01/2008 to 07/31/2011

4. TITLE AND SUBTITLE
Stochastic Pseudo-Boolean Optimization

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-08-1-0268

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Dr. Oleg Prokopyev

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Pittsburgh
Office of Research
123 University Place
Pittsburgh, PA 15213-2303

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research
875 N. Randolph Street, Room 3112
Arlington, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-OSR-VA-TR-2012-0871

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution A

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Pseudo-boolean (and general nonlinear integer) functions provide an extremely powerful modeling and solution tool in operations research and related areas. A large number of practical as well as purely theoretical decision problems can be easily represented and solved as optimization of a pseudo-boolean or general nonlinear integer function. In the framework of this project we have considered several stochastic extensions of classical combinatorial optimization problems that involve some type of nonlinearity, typically in the objective function. We have provided respective theoretical analysis and developed advanced solution approaches. In particular, we have investigated the following topics: (i) exact solution algorithms for broad classes of two-stage stochastic quadratic binary and general integer programming problems; (ii) approximation algorithms for solving a class of two-stage stochastic assignment problems; (iii) theoretical analysis of two-stage stochastic minimum s-t cut problems; (iv) exact solution algorithm for a class of stochastic bilevel knapsack problems; (v) exact solution algorithms for a class multiple-ratio fractional programming problems; and (vi) integer programming approach for solving a polyomino tiling problem with application in antenna design.

15. SUBJECT TERMS
Pseudo-boolean optimization, nonlinear optimization, integer optimization, stochastic optimization, combinatorial optimization

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Oleg Prokopyev
Unclassified	Unclassified	Unclassified	UU	122	19b. TELEPHONE NUMBER *(Include area code)* 412-624-9833

Contents

1 Report Summary and Organization **3**

2 Two-Stage Stochastic Quadratic Integer Programs **5**
2.1 Introduction . . . 5
2.2 Contribution . . . 6

3 Two-Stage Stochastic Quadratic Binary Programs **7**
3.1 Introduction . . . 7
3.2 Alternative Dual Decomposition Schemes . . . 9
3.3 Preliminary Numerical Study . . . 13
3.4 A Lagrangian Decomposition Approach for QBPs . . . 14
3.4.1 Introduction . . . 14
3.4.2 A Lagrangian Decomposition Method . . . 16
3.4.3 Structure Preserving Decomposition . . . 26
3.4.4 Computational Considerations in the B&B Framework . . . 29
3.4.5 Computational Experiments . . . 30
3.5 Concluding Remarks and Future Work . . . 31

4 Two-Stage Stochastic Minimum $s-t$ Problem **35**
4.1 Introduction . . . 35
4.2 Mathematical Programming Formulation . . . 36
4.2.1 Total Unimodularity . . . 39
4.2.2 General Case: Total Unimodularity is Lost . . . 40
4.2.3 Trees: Total Unimodularity is Preserved . . . 42
4.3 Computational Complexity . . . 50
4.4 Linear Running Time Algorithm for Trees . . . 53
4.5 Node-Based Version . . . 54
4.6 Concluding Remarks . . . 57

5 Bilevel Knapsack Problems with Stochastic Right-Hand Sides **58**

6 Two-Stage Stochastic Assignment Problems **59**
6.1 Introduction . . . 59
6.2 Greedy Approximation Algorithms . . . 60
6.2.1 Basic Greedy Approach . . . 60
6.2.2 Greedy Approach of Escoffier et al. . . . 61
6.3 Necessary Optimality Condition . . . 62
6.4 Enhanced Greedy Approach . . . 63
6.4.1 Improving the first-stage assignment (EGA-I) . . . 64
6.4.2 Improving the second-stage assignment (`EGA-II`) . . . 68
6.4.3 Improving EGA with Local Search . . . 71
6.4.4 Analytical Observations . . . 73
6.5 Computational Experiments . . . 75
6.5.1 Setup . . . 75
6.5.2 Results and Discussion . . . 77
6.6 Concluding Remarks . . . 81

7 Multiple-Ratio Fractional Programming Problems **84**
7.1 Introduction . . . 84
7.2 A Global Optimization Approach . . . 86
7.2.1 Description and convergence of the main algorithm . . . 86
7.2.2 Solving the subproblem . . . 89
7.2.3 Partitioning the feasible region . . . 91
7.3 Computational Experiments . . . 93
7.3.1 Setup . . . 93
7.3.2 Results and Discussion . . . 93
7.4 Concluding Remarks . . . 96

8 Irregular Polyomino Tiling via Integer Programming **97**
8.1 Introduction . . . 97
8.2 Information Theoretic Entropy as a Measure of "Irregularity" . . . 98
8.3 Mathematical Programming Formulations . . . 99
8.3.1 Formal Setup . . . 99
8.3.2 Nonlinear Set Partitioning Formulation . . . 101
8.3.3 Linear Set Partitioning Formulation . . . 101
8.4 Heuristics . . . 103
8.4.1 Heuristic Procedure: Zoom-in . . . 103
8.4.2 Heuristic Procedure: Magnify . . . 104
8.4.3 Heuristic Procedure: Retile . . . 105
8.4.4 Heuristic Procedure: Smoothen . . . 106
8.4.5 Heuristic Procedure: Randomize . . . 106
8.5 Current Work and Concluding Remarks . . . 110

9 Participants **111**

10 Publications **111**

11 Presentations **112**

1 Report Summary and Organization

Nonlinear functions of binary (and general integer) variables naturally arise when modeling selections and interactions within a unified optimization framework. For example, consider a set of n objects $\{1, \ldots, n\}$. For each pair of objects (i, j) we associate a weight q_{ij} measuring the interaction between objects i and j. Let $x_i = 1$ if object i is selected (or $x_i = k$, $k \in \mathbb{Z}^1_{++}$ if several copies of the object may be selected), and $x_i = 0$, otherwise. If the global interaction is the sum of all interactions between the selected objects, then their global interaction can be formulated as a quadratic function $\sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j$.

Pseudo-boolean (and general nonlinear integer) functions provide an extremely powerful modeling and solution tool in operations research and related areas. A large number of practical as well as purely theoretical decision problems can be easily represented and solved as optimization of a pseudo-boolean (or general nonlinear integer) function.

Given the recent advances in computational stochastic discrete optimization and continuing pursuit of cost-effectiveness when making complex decisions in noticeable stochastic dynamic environments, we have witnessed many applications of stochastic programming to real-world problems such as capacity planning, facility location, and production control. Stochastic programming provides a simple optimization model of decision making under uncertainty to overcome many limitations of classic deterministic approaches. However, many stochastic combinatorial optimization problems are notoriously difficult to solve. This fact has greatly hindered the further application of stochastic programming to many problems where discrete decisions are involved.

We have also seen the increasing number of nonlinear models for combinatorial optimization problems, e.g, different types of nonlinear assignment problems, application of various quadratic and fractional binary programming models in medicine, chemistry, computational biology and data mining, etc. These models are much more suitable for modeling real-world problems full of nonlinearity and they are more capable for modeling interactions among involved entities. However, very little work has been done to extend these wonderful results in stochastic dynamic environments.

Therefore, the major goal of the project was investigating stochastic pseudo-boolean (and general nonlinear integer) optimization problems. The remainder of this report is organized as follows:

- Section 2 provides a novel solution approach for solving a broad class of two-stage stochastic quadratic integer programs. The proposed approach is based on the value function reformulation of the original problem. We show that our approach can solve instances whose extensive forms are hundreds of orders of magnitude larger than the largest quadratic integer programming instances solved in the literature.

- Section 3 describes new Lagrangian based approaches for solving two-stage stochastic and deterministic quadratic binary programs.

- Section 4 is focused on two-stage stochastic extensions of the classical minimum $s - t$ cut problem. The deterministic minimum $s - t$ cut problem has two equivalent formulations that are motivated by two different interpretations of the problem: (i) linear 0–1 program (arc based interpretation) and (ii) quadratic 0–1 program (node partitioning based interpretation). We show that stochastic extensions of these equivalent deterministic models result in two different stochastic optimization problems. We discuss the corresponding mathematical programming formulations and related computational complexity issues.

- Section 5 considers a specific stochastic extension of the bilevel knapsack problem. Bilevel and multilevel optimization is extremely important in military and law enforcement appli-

cations since it naturally models the adversarial relationship between the upper- (the attacker/defender) and lower-level (the defender/attacker) decision makers.

- Section 6 describes greedy approximation algorithms for solving a class of two-stage stochastic assignment problems.
- Section 7 provides a global optimization algorithm for solving a class of multiple-ratio fractional combinatorial optimization problems. Multiple-ratio problems often arise when one considers stochastic extensions of single-ratio fractional problems.
- Section 8 describes a nonlinear integer optimization model for the irregular polyomino tiling problem. It is motivated by an antenna design application.

In Section 9 we list the participants of the projects. Sections 10 and 11 summarize the most important refereed journal publications and research conference presentations, respectively.

We should emphasize that according to the performance report requirements we **do not** provide copies of already published articles in this report.

2 Two-Stage Stochastic Quadratic Integer Programs

The details of the work in this chapter can be found in:

- O.Y. Ozaltin, O.A. Prokopyev, A.J. Schaefer, "Two-Stage Quadratic Integer Programs with Stochastic Right-Hand Sides," *Mathematical Programming*, accepted for publication, 2010.

2.1 Introduction

We consider the following class of two-stage quadratic integer programs with stochastic right-hand sides:

$$(P1): \quad \max \quad \frac{1}{2}x^T\Lambda x + c^T x + \mathbb{E}_\omega \mathbf{Q}(x,\omega) \tag{1a}$$

$$\text{subject to} \quad x \in \mathbb{X}, \tag{1b}$$

where $\mathbb{X} = \{x \in \mathbb{Z}_+^{n_1} \mid Ax \leq b\}$ and,

$$\mathbf{Q}(x,\omega) = \max \quad \frac{1}{2}y^T\Gamma y + d^T y \tag{2a}$$

$$\text{subject to} \quad Wy \leq h(\omega) - Tx, \tag{2b}$$

$$y \in \mathbb{Z}_+^{n_2}. \tag{2c}$$

The random variable ω from probability space $(\Omega, \mathcal{F}, \mathcal{P})$ describes the realizations of uncertain parameters, known as *scenarios*. The numbers of constraints and decision variables in stage i are m_i and n_i, respectively, for $i = 1, 2$. The first-stage objective vector $c \in \mathbb{R}^{n_1}$, right-hand side vector $b \in \mathbb{R}^{m_1}$ and the second-stage objective vector $d \in \mathbb{R}^{n_2}$ are known column vectors. The first-stage constraint matrix $A \in \mathbb{R}^{m_1 \times n_1}$, technology matrix $T \in \mathbb{R}^{m_2 \times n_1}$ and recourse matrix $W \in \mathbb{R}^{m_2 \times n_2}$ are all deterministic. Furthermore, $\Lambda \in \mathbb{R}^{n_1 \times n_1}$ and $\Gamma \in \mathbb{R}^{n_2 \times n_2}$ are known, and possibly indefinite, symmetric matrices. The stochastic component consists of only $h(\omega) \in \mathbb{R}^{m_2}$ $\forall \omega \in \Omega$.

The *extensive form* formulation of $(P1)$ is given by:

$$\max \quad \frac{1}{2}x^T\Lambda x + c^T x + \mathbb{E}_\omega \left[\frac{1}{2}y(\omega)^T\Gamma y(\omega) + d^T y(\omega)\right] \tag{3a}$$

$$\text{subject to} \quad x \in \mathbb{X}, \tag{3b}$$

$$Wy(\omega) \leq h(\omega) - Tx \qquad \forall \omega \in \Omega, \tag{3c}$$

$$y(\omega) \in \mathbb{Z}_+^{n_2} \qquad \forall \omega \in \Omega. \tag{3d}$$

In this work we make the following assumptions:

A1 The random variable ω follows a discrete distribution with finite support.

A2 The first-stage feasibility set $\mathbb{X} = \{x \in \mathbb{Z}_+^{n_1} \mid Ax \leq b\}$ is nonempty and bounded.

A3 $\mathbf{Q}(x,\omega)$ is finite for all $x \in \mathbb{X}$ and $\omega \in \Omega$.

A4 The first-stage constraint matrix A, technology matrix T and recourse matrix W are all integral, i.e. $A \in \mathbb{Z}^{m_1 \times n_1}$, $T \in \mathbb{Z}^{m_2 \times n_1}$, $W \in \mathbb{Z}^{m_2 \times n_2}$.

Assumption **A1** is justified by Schultz [136], who showed that the optimal solution to any stochastic program with continuously distributed ω can be approximated within any desired accuracy using a discrete distribution. Assumption **A2** and integrality restrictions in the first stage ensure that $\mathbb{X}$ is a finite set. Assumption **A3** ensures that $\mathbf{Q}(x, \omega)$ is feasible for all $x \in \mathbb{X}$ and $\omega \in \Omega$, i.e. relatively complete recourse [148]. Assumption **A4** is not too restrictive in a sense, as any rational matrix can be converted to an integral one. Most of the stochastic programming studies in the literature make assumptions similar to **A1**-**A3** [7, 34, 89, 137] and **A4** [89]. Without loss of generality, we also assume that $b \in \mathbb{Z}^{m_1}$ and $h(\omega) \in \mathbb{Z}^{m_2}$ $\forall \omega \in \Omega$, as A, T and W are all integer matrices. Note that all of the undesirable properties of stochastic integer programs, e.g. discontinuity and nonconvexity of $\mathbf{Q}(x, \omega)$, still exist in $(P1)$.

We reformulate $(P1)$ using the value functions of the first- and second-stage quadratic integer programs. The advantage of this reformulation is that it is relatively insensitive to the number of variables and scenarios. In the first phase of our solution approach, we construct the value functions in both stages. In the second phase, we use a global branch-and-bound algorithm or a level-set approach to optimize $(P1)$ over the set of feasible first-stage right-hand sides.

Our approach can solve very large instances of $(P1)$ as measured by the size of the extensive form. However, it is sensitive to the number of constraints in each stage and the magnitude of $h(\omega)$. Note that the number of quadratic integer programs that must be solved when constructing the value function grows exponentially in the number of constraints. A major contribution of this work is to propose algorithms that can mitigate the effect of this exponential growth to some extent by exploiting the properties of value functions. Specifically, our approach can handle instances of $(P1)$ that have up to seven constraints in each stage.

2.2 Contribution

We develop an algorithmic framework for a class of two-stage stochastic quadratic integer programs where the uncertainty only appears in the second-stage right-hand sides. The main contribution of the work is twofold. First, we derive some theoretical properties of QIP value functions. These properties may be useful in sensitivity analysis of quadratic integer programs [43, 73]. Second, we use these properties as well as superadditivity to develop efficient algorithms for computing value functions of QIPs. We then apply a dual reformulation and use a generic global branch-and-bound algorithm and a level-set approach to find an optimal tender.

This work represents an important first step towards more general two-stage stochastic quadratic integer programs where uncertainty appears in the second-stage objective and constraint matrix, as well as the right-hand side. We note that our approach is amenable to solve general two-stage stochastic quadratic integer programs as long as the scenarios may be divided into relatively few groups that share the same objective functions and constraint matrices. For such instances, the value function must be found for the first stage and each group of scenarios.

The major limitation of our two-phase solution approach is the explicit storage of value functions in computer memory. This is why our computations are based on instances that have large number of columns and scenarios but relatively few rows. One approach to overcome this limitation is to seek more efficient ways to store value functions, such as using generating functions [96]. Another approach is to modify the global branch-and-bound algorithm to calculate the solution on a subset of right-hand sides so that only a portion of the value function needs to be stored at any time.

3 Two-Stage Stochastic Quadratic Binary Programs

This chapter is mostly based on the results from:

- Z. Zhu, N. Kong, O.A. Prokopyev, "A New Lagrangian Decomposition Based Approach for Quadratic Binary Programs," Technical Report, 2011.
- Z. Zhu, N. Kong, O.A. Prokopyev, "Two-Stage Stochastic Quadratic Binary Program with Recourse: A Dual Decomposition Approach," Working paper, 2011.

3.1 Introduction

A two-stage stochastic quadratic binary program with fixed resource (SQBP) can be presented as:

$$\begin{aligned} (\text{SQBP}): \max_{x} \ & c^T x + x^T C x + \sum_{k=1}^{K} p^k Q(x,k) \\ \text{s.t.} \quad & Ax \ \leq \ b; \\ & x \ \in \ \{0,1\}^{n_1}, \end{aligned}$$

and for each $k = 1, \ldots, K$,

$$\begin{aligned} Q(x,k) \ &= \ \max_{y} \ (d^k)^T y + y^T D^k y \\ \text{s.t.} \quad Wy \ &\leq \ h^k \ - \ T^k x; \\ y \ &\in \ \{0,1\}^{n_2}. \end{aligned}$$

In Problem (SQBP), matrices $C := \{c_{ij}\} \in \mathbb{R}^{n_1^2}$ and $D^k = \{d_{ij}\} \in \mathbb{R}^{n_2^2}$ for $k = 1, \ldots, K$, contain first-stage and second-stage objective coefficients for the quadratic terms of $x_i x_j$ and $y_i y_j$, respectively, and vectors $c := \{c_i\} \in \mathbb{R}^{n_1}$ and $d^k := \{d_i^k\} \in \mathbb{R}^{n_2}$ for $k = 1, \ldots, K$, contain first-stage and second-stage objective coefficients for the linear terms x_i and y_i, respectively. In addition, matrices $A \in \mathbb{R}^{m_1 \times n_1}$, $W \in \mathbb{R}^{m_2 \times n_2}$, $T \in \mathbb{R}^{m_2 \times n_1}$, for $k = 1, \ldots, K$, are known real matrices, and vectors $b \in \mathbb{R}^{m_1}$, $h^k \in \mathbb{R}^{m_2}$, for $k = 1, \ldots, K$, are known real vectors. With the problem, a decision maker takes the first-stage decisions and takes the second-stage recourse decision based on some realization of the uncertainty, which is not exogenous with respect to first-stage decisions. The objective is to maximize the sum of the first-stage benefit and the expected second-stage benefit. To avoid complications when computing the expectation, we assume that we only have a finite number K of scenarios. Hence, each scenario $k = 1, \ldots, K$, having probability p^k, is represented by (d^k, D^k, h^k, T^k).

The problem (SQBP) is equivalent to a large, dual block-angular quadratic binary program. For $k = 1, \ldots, K$, we define the set

$$\mathcal{F}^k := \{(x, y^k) : Ax \leq b, x \in \mathbb{B}^{n_1}, T^k x + W y^k \leq h^k, y^k \in \mathbb{B}^{n_2}\}.$$

Then the deterministic equivalent of (SQBP) can be written as

$$z = \max \left\{ c^T x + x^T C x + \sum_{k=1}^{K} p^k \left((d^k)^T y^k + (y^k)^T D^k y^k \right) \ \middle| \ (x, y^k) \in \mathcal{F}^k, k = 1, \ldots, K \right\}. \quad (4)$$

It is important to develop efficient solution methods for SQBPs for the following two reasons. First, deterministic QBPs have been extensively studied in scheduling [10], computer-aided design

[17, 84], computational biology [60], among others. Many graph-theoretic problems can be naturally formulated with quadratic binary programs (QBPs) [2, 115]. Second, optimization under uncertainty has been shown critical in many decision problems as stochastic linear binary programs (SLBPs), the linear counterpart of SQBPs (i.e., $C = D = 0$), have been applied in many fields such as energy planning [86], manufacturing [47], logistics [92], etc.

It is not surprise that solving SQBPs is even more computationally prohibitive as SQBPs inherit computational challenges from both stochastic integer programming and deterministic quadratic binary optimization. Very little work has been done to develop efficient algorithms for SQBPs. In this chapter, we adapt *dual decomposition* (or termed *scenario decomposition*), an approach originally developed in Carøe and Schultz [33] for solving SIPs. The idea of scenario decomposition is to introduce copies $x^1, \ldots, x^K$ of the first-stage variable x and then rewrite Problem (4) in the form

$$\max_{x^1,\ldots,x^K,y^1,\ldots,y^K} \left\{ \sum_{k=1}^{K} p^k \left(c^T x^k + (x^k)^T C x^k + (d^k)^T y^k + (y^k)^T D^k y^k \right) \,\middle|\, (x^k, y^k) \in \mathcal{F}^k, \right.$$
$$\left. k = 1, \ldots, K, x^1 = \cdots = x^K \right\}. \tag{5}$$

Here the non-anticipativity condition $x^1 = \cdots = x^K$ states that the first-stage decision should not depend on the scenario that is realized in the second stage. To solve (5), one can apply Lagrangian relaxation with respect to the non-anticipativity condition and recovery the identity among the first-stage variables with branch and bound. A major advantage of this solution approach is that it splits into separate subproblems for different scenarios. Note that the idea of dual decomposition can be related to existing techniques in both combinatorial optimization (i.e., [74, 83]) and stochastic programming (i.e., [131, 132]).

For SIP, it is clear that there exist several equivalent representations of the non-anticipativity condition. In general, it can be represented by the equality $\sum_{k=1}^{K} \Gamma^k x^k = 0$ where $\Gamma = (\Gamma^1, \ldots, \Gamma^K)$ is a suitable matrix. One notable representation in SLBP is via the single constraint

$$\left(\sum_{k=2}^{K} \gamma_k \right) x^1 = \gamma_2 x^2 + \ldots + \gamma_K x^K, \tag{6}$$

where $\gamma_2, \ldots, \gamma_K$ are positive weights.

When extending the idea of dual decomposition to SQBP, we enjoy more flexibility on imposing the non-anticipativity condition. Feasible representations may involve quadratic constraints, i.e., $\sum_{k=1}^{K} \Lambda^k x^k (x^k)^T = 0$ where $\Lambda = (\Lambda^1, \ldots, \Lambda^K)$ is a matrix. Each feasible representation leads to a Lagrangian decomposition scheme. It is well known that there exists a tradeoff between the solution efficiency and bounding quality of Lagrangian duals. In this chapter, we explore a few representations that involve quadratic constraints and investigate corresponding dual decomposition schemes. For each deterministic quadratic binary Lagrangian dual subproblem, we propose an innovative Lagrangian decomposition based branch-and-bound method that is inspired by the idea of variable splitting. *In fact, the proposed Lagrangian decomposition based branch-and-bound method is suitable to generic quadratic binary programs.*

The remainder of the chapter is organized as follows. In Section 3.2, we introduce several representations of the non-anticipativity condition. We derive the corresponding Lagrangian duals and compare their tightness analytically. In Section 3.3, we report preliminary computational experiments on bound tightness and solution efficiency of the SQBPs. In Section 3.4, we present the innovative Lagrangian decomposition based approach for general quadratic binary programs. Section 3.5, we draw conclusions and outline future research.

3.2 Alternative Dual Decomposition Schemes

In this section, we explore alternative dual decomposition schemes with respect to several representations of the non-anticipativity condition. The two key factors are 1) inclusion of quadratic constraints to represent the non-anticipativity condition and 2) sequence of linearization of quadratic cross terms and dual decomposition over scenarios.

We present the two generic forms of the Lagrangian relaxation with respect to the non-anticipativity condition. First, we represent the condition only with linear constraints. The Lagrangian relaxation is the problem of finding x^k, y^k, $k = 1, \ldots, K$, such that

$$D(\mu^\gamma) = \max \left\{ \sum_{k=1}^{K} L_k(x^k, y^k, \mu^\gamma) : (x^k, y^k) \in \mathcal{F}^k \right\}, \tag{7}$$

where μ^γ has proper dimension and $L_k(x^k, y^k, \mu^\gamma) = p^k \left(c^T x^k + (x^k)^T C x^k + (d^k)^T y^k + (y^k)^T D y^k \right) + \mu^\gamma(\Gamma^k x^k)$ for $k = 1, \ldots, K$. The Lagrangian dual of Problem (7) then becomes the problem

$$z^{LD} := \min_{\mu^\gamma} D(\mu^\gamma). \tag{8}$$

Second, we represent the condition with both linear and quadratic constraints. Thus, the Lagrangian relaxation is

$$D'(\mu^\gamma, \mu^\lambda) = \max \left\{ \sum_{k=1}^{K} L'_k(x^k, y^k, \mu^\gamma, \mu^\lambda) : (x^k, y^k) \in \mathcal{F}^k \right\}, \tag{9}$$

where μ^γ and μ^λ have proper dimensions, and $L'_k(x^k, y^k, \mu^\gamma, \mu^\lambda) = p^k \left(c^T x^k + (x^k)^T C x^k + (d^k)^T y^k + (y^k)^T D y^k \right) + \mu^\gamma(\Gamma^k x^k) + \mu^\lambda(\Lambda^k x^k (x^k)^T)$ for $k = 1, \ldots, K$. The Lagrangian dual of Problem (9)

$$z_Q^{LD} := \min_{\mu^\gamma, \mu^\lambda} D'(\mu^\gamma, \mu^\lambda). \tag{10}$$

Since there are an enormous amount of possible representations on the non-anticipativity condition, i.e., enormously many forms of Γ and/or Λ, we consider four commonly used representations. The first set of two presentation are universally applicable. They use at least one constraint to force the identity of first-stage variables with respect to each pair of scenarios. For the first representation, we impose constraints $x^{k_1} = x^{k_2}$ for all $1 \leq k_1 < k_2 \leq K$. For the second representation, in addition to imposing the above constraints, we impose constraints $x^{k_1}(x^{k_1})^T = x^{k_2}(x^{k_2})^T$ for all $1 \leq k_1 < k_2 \leq K$. In other words, for any cross term of first-stage variables, we impose constraints to force the scenario-wise identity. With proper specifications of Γ and Λ, we can further derive (7) and (9). We denote the corresponding Lagrangian duals by z_1^{LD} and z_{1Q}^{LD}, respectively.

The second set of two representations of the non-anticipativity condition that we consider in this chapter, are applicable when all first-stage variables are required to be binary, which is the case in our problem. These two representations use one or two constraints to force the scenario-wise identity. To distinguish with the two representations described in the previous paragraph, we call these two representations the third and fourth representations. For the third representation, we impose constraints (6) for all pairs of scenarios. For the fourth representation, in addition to imposing constraints (6), we impose constraints

$$\left(\sum_{k=2}^{K} \lambda_k \right) x^1 (x^1)^T = \sum_{k=2}^{K} \lambda_k x^k (x^k)^T. \tag{11}$$

With the third representation, we further write the Lagrangian relaxation as follows. For $k = 1$, the Lagrangian relaxation associated with scenario $k = 1$ is

$$D_1(\mu^\gamma) = \max_{(x^1,y^1)\in\mathcal{F}^1} \left\{ p^1 \left((c^T - \frac{1}{p^1}(\sum_{k=2}^{K} \gamma_k)\mu^\gamma)x^1 + (x^1)^T C x^1 + (d^1)^T y^1 + (y^1)^T D^1(y^1) \right) \right\}. \quad (12)$$

For $k = 2, \ldots, K$, the Lagrangian relaxation associated with scenario k is

$$D_k(\mu^\gamma) = \max_{(x^k,y^k)\in\mathcal{F}^k} \left\{ p^k \left((c^T + \frac{1}{p^k}\gamma_k)\mu^\gamma)x^k + (x^k)^T C x^k + (d^k)^T y^k + (y^k)^T D^k(y^k) \right) \right\}. \quad (13)$$

Then the Lagrangian dual, denoted by z_2^{LD} is as:

$$z_2^{LD} := \min_{\mu^\gamma} \left\{ \sum_{k=1}^{K} D_k(\mu^\gamma) \right\}. \quad (14)$$

With the fourth representation, we further write the Lagrangian relaxation as follows. For $k = 1$, the Lagrangian relaxation associated with scenario $k = 1$ is

$$D_1'(\mu^\gamma, \mu^\lambda) = \max_{(x^1,y^1)\in\mathcal{F}^1} \left\{ p^1 \left((c^T - \frac{1}{p^1}(\sum_{k=2}^{K} \gamma_k)\mu^\gamma)x^1 + (x^1)^T (C - \lambda_1 I) x^1 + (d^1)^T y^1 + (y^1)^T D^1(y^1) \right) \right\}. \quad (15)$$

For $k = 2, \ldots, K$, the Lagrangian relaxation associated with scenario k is

$$D_k'(\mu^\gamma, \mu^\lambda) = \max_{(x^k,y^k)\in\mathcal{F}^k} \left\{ p^k \left((c^T + \frac{1}{p^k}\gamma_k)\mu^\gamma)x^k + (x^k)^T (C + \lambda_k I) x^k + (d^k)^T y^k + (y^k)^T D^k(y^k) \right) \right\}. \quad (16)$$

Then the Lagrangian dual, denoted by z_{2Q}^{LD} is as

$$z_{2Q}^{LD} := \min_{\mu^\gamma, \mu^\lambda} \left\{ \sum_{k=1}^{K} D_k'(\mu^\gamma, \mu^\lambda) \right\}. \quad (17)$$

Remark 1 *Several results follow trivially. They are: 1) $z_Q^{LD} \le z^{LD}$; 2) $z_{1Q}^{LD} \le z_1^{LD}$; and 3) $z_{2Q}^{LD} \le z_2^{LD}$.*

Remark 2 *The approach of imposing non-anticipativity constraints on quadratic terms (z_Q^{LD}, z_{1Q}^{LD}, z_{2Q}^{LD}) is identical to first applying standard linearizaton on first-stage variables and then applying dual decomposition, i.e., $(\sum_{k=2}^{K} \lambda^k) z_{ij}^k = \sum_{k=2}^{K} \lambda^k z_{ij}^k$ with $z_{ij} = x_i x_j$ for $1 \le i < j \le n_1$.*

There is a common feature in the above described dual decomposition schemes. That is, all Lagrangian duals are derived without linearizing the cross terms of first-stage decision variables. Next we describe several alternative dual decomposition schemes, which are related to the following reformulation of Problem (4).

$$z' = \max \left\{ \sum_{i=1}^{n_1} c_i x_i + \sum_{i=1}^{n_1-1} \sum_{j=2}^{n_1} c_{ij} z_{ij} \right.$$

$$+\sum_{k=1}^{K} p^k \left(\sum_{i=1}^{n_2} d_i^k y_i^k + \sum_{i=1}^{n_2-1} \sum_{j=2}^{n_2} d_{ij}^k y_i^k y_j^k \right) \Bigg| \; z_{ij} = x_i x_j, (x, y^k) \in \mathcal{F}^k \Bigg\}. \quad (18)$$

In the above formulation (18), without loss of generality, we assume that $c_{ij} = c_{ji}$ for $i, j = 1, \ldots, n_1$, $i < j$, and $d_{ij} = d_{ji}$ for $i, j = 1, \ldots, n_2$, $i < j$. It is easy to see (18) is equivalent to Problem (4). We next impose the non-anticipativity condition only on linear terms of first-stage variables. We rewrite Problem (18) as follows.

$$\max_{x^1,\ldots,x^K,y^1,\ldots,y^K,z} \quad \sum_{k=1}^{K} p^k \left(\sum_{i=1}^{n_1} c_i x_i^k + \sum_{i=1}^{n_2} d_i^k y_i^k + \sum_{i=1}^{n_2-1} \sum_{j=2}^{n_2} d_{ij}^k y_i^k y_j^k \right) + \sum_{i=1}^{n_1-1} \sum_{j=2}^{n_1} c_{ij} z_{ij} \quad (19)$$

$$\text{subject to} \quad (x^k, y^k) \in \mathcal{F}^k, \quad k = 1, \ldots, K, \quad (20)$$

$$\sum_{k=1}^{K} \Gamma^k x^k = 0, \quad (21)$$

$$z_{ij} \le x_i^k, \quad i, j = 1, \ldots, n_1, \; j > i, \; k = 1, \ldots, K, \quad (22)$$

$$z_{ij} \le x_j^k, \quad i, j = 1, \ldots, n_1, \; j > i, \; k = 1, \ldots, K, \quad (23)$$

$$z_{ij} \ge x_i^k + x_j^k - 1, \quad i, j = 1, \ldots, n_1, \; j > i, \; k = 1, \ldots, K. \quad (24)$$

$$z_{ij} \in [0, 1], \quad i, j = 1, \ldots, n_1. \quad (25)$$

Note that to ensure optimality, variables z are only required to be continuous between 0 and 1.

In the following presentation, we fix the presentation of the non-anticipativity condition to be (6). Therefore, we replace (21) with (6). We apply Lagrangian relaxation with respect to both (21) and (22) – (24) as follows. For each i, we denote μ_i to be the Lagrangian multiplier associated with (21). For each $i, j = 1, \ldots, n_1$ with $j > i$ and each $k = 1, \ldots, K$, we denote θ_{ij}^k and θ_{ji}^k to be the Lagrangian multiplier associated with each of constraints (22) and each of constraints (23), respectively. For each $i, j = 1, \ldots, n_1$ with $j > i$ and each $k = 1, \ldots, K$, we denote λ_{ij}^k to the Lagrangian multiplier associated with each (24).

For $k = 1$, the Lagrangian relaxation associated with scenario 1, is

$$D_1''(\mu, \theta^1, \lambda^1) =$$

$$\max \Bigg\{ \sum_{i=1}^{n_1} \left(p^1 c_i - (\sum_{k=2}^{K} \gamma_k)\mu_i + \sum_{j \ne i} \theta_{ij}^1 - \sum_{j=i+1}^{n_1} \lambda_{ij}^1 - \sum_{j=1}^{i-1} \lambda_{ji}^1 \right) x_i^1 + \sum_{i=1}^{n_2} p^1 d_i^1 y_i^1 + \sum_{i=1}^{n_2-1} \sum_{j=2}^{n_2} p^1 d_{ij}^1 y_i^1 y_j^1$$

$$+ \sum_{i=1}^{n_1-1} \sum_{j=2}^{n_1} (c_{ij} - \theta_{ij}^1 - \theta_{ji}^1 + \lambda_{ij}^1) z_{ij} + \sum_{i=1}^{n_1-1} \sum_{j=2}^{n_1} \lambda_{ij}^1 \Bigg| (x^1, y^1) \in \mathcal{F}^1, z_{ij} \in [0, 1] \Bigg\}. \quad (26)$$

For $k = 2, \ldots, K$, the Lagrangian relaxation associated with scenario k, is

$$D_k''(\mu, \theta^k, \lambda^k) =$$

$$\max \Bigg\{ \sum_{i=1}^{n_1} \left(p^k c_i + \mu_i + \sum_{j \ne i} \theta_{ij}^k - \sum_{j=i+1}^{n_1} \lambda_{ij}^k - \sum_{j=1}^{i-1} \lambda_{ji}^k \right) x_i^k + \sum_{i=1}^{n_2} p^k d_i^k y_i^k + \sum_{i=1}^{n_2-1} \sum_{j=2}^{n_2} p^k d_{ij}^k y_i^k y_j^k$$

$$+\sum_{i=1}^{n_1-1}\sum_{j=2}^{n_1}(c_{ij}-\theta_{ij}^k-\theta_{ji}^k+\lambda_{ij}^k)z_{ij}++\sum_{i=1}^{n_1-1}\sum_{j=2}^{n_1}\lambda_{ij}^k \;\Bigg|\; (x^k,y^k)\in\mathcal{F}^k, z_{ij}\in[0,1]\Bigg\}. \tag{27}$$

Therefore, the corresponding Lagrangian dual, denoted by z_3^{LD} is as

$$z_3^{LD}=\min_{\mu,\theta,\lambda}\sum_{k=1}^{K}D_k''(\mu,\theta^k,\lambda^k) \tag{28}$$

Note that in each of the above Lagrangian relaxations, variables z_{ij} are unconstrained except for the bounds. Hence, in (26) and (27), for scenario k, $k=1,\ldots,K$, the optimal solution $z_{ij}^*=1$ if $c_{ij}-\theta_{ij}^k-\theta_{ji}^k+\lambda_{ij}^k\geq 0$, and $z_{ij}^*=0$ otherwise.

Remark 3 *Since to compute z_3^{LD}, one does not relax the non-anticipativity constraints on variables z_{ij}, we have $z_3^{LD}\leq z_{2Q}^{LD}$.*

There are a large number of constraints (22) – (24) when K is large, which presents significant computational challenge in computing z_3^{LD}. Hence, we consider alternative Lagrangian relaxations that are easier to compute but provide inferior bounds. By the non-anticipativity constraint (21), it is sufficient to keep the set of constraints (22) – (24) for only one scenario. Let us define $z_{3\kappa}^{LD}$ to be the Lagrangian dual if we impose constraints (22) – (24) only for a selected scenario κ. Furthermore, one can consider replacing (22) – (24) with the following aggregate constraint sets:

$$\left(\sum_{k=1}^{K}(\alpha_i^k+\alpha_j^k)\right)\cdot z_{ij}\leq\sum_{k=1}^{K}(\alpha_i^k x_i^k+\alpha_j^k x_j^k),\;\; i,j=1,\ldots,n_1,\;\; j>i; \tag{29}$$

$$\left(\sum_{k=1}^{K}\beta_{ij}^k\right)\cdot z_{ij}\geq\sum_{k=1}^{K}(\beta_{ij}^k x_i^k+\beta_{ij}^k x_j^k-1),\qquad i,j=1,\ldots,n_1,\;\; j>i; \tag{30}$$

In (29), we introduce coefficients α_i^k for $i=1,\ldots,n_1$ and $k=1,\ldots,K$, which are associated with constraints (22) and (23). In (30), we introduce coefficients β_{ij}^k for $i,j=1,\ldots,n_1$, $i<j$, and $k=1,\ldots,K$, which are associated with constraint (24). However, constraints (25) need to be replaced by

$$z_{ij}\in\{0,1\},\;\; i,j=1,\ldots,n_1,\;\; j>i. \tag{31}$$

We apply Lagrangian relaxation with respect to alternative non-anticipativity constraints (29) and (30), and constraints (31), to compute an alternative Lagrangian dual as follows. For each index pair (i,j) with $i<j$, we denote θ_{ij} and λ_{ij} to be the Lagrangian multipliers associated with constraints (29) and (30).

For $k=1$, the Lagrangian relaxation associated with scenario 1, is

$$D_1'''(\mu,\theta,\lambda)=\max\Bigg\{\sum_{i=1}^{n_1}\left(p^1c_i-(\sum_{k=2}^{K}\gamma_k)\mu_i+\alpha_i^1(\sum_{j=i+1}^{n_1}\theta_{ij}+\sum_{j=1}^{i-1}\theta_{ji})-\sum_{j=i+1}^{n_1}\beta_{ij}^1\lambda_{ij}-\sum_{j=1}^{i-1}\beta_{ji}^1\lambda_{ji}\right)x_i^1$$

$$+\sum_{i=1}^{n_2}p^1d_i^1y_i^1+\sum_{i=1}^{n_2-1}\sum_{j=2}^{n_2}p^1d_{ij}^1y_i^1y_j^1$$

$$+\sum_{i=1}^{n_1-1}\sum_{j=2}^{n_1}(c_{ij}-(\alpha_i^1+\alpha_j^1)\theta_{ij}+\beta_{ij}^1\lambda_{ij})z_{ij}+\sum_{i=1}^{n_1-1}\sum_{j=2}^{n_1}\lambda_{ij} \;\Bigg|\; (x^1,y^1)\in\mathcal{F}^1, z_{ij}\in\{0,1\}\Bigg\}. \tag{32}$$

Inst No.	z_{3A}^{LD}	z_{2Q}^{LD}
1	1863.19 (2.04)	1883.39 (3.00)
2	1989.14 (1.45)	2008.64 (3.67)
3	2010.63 (2.84)	2134.25 (3.74)
4	1899.41 (2.46)	1843.52 (3.30)
5	1855.97 (4.73)	1897.65 (5.43)

Table 1: Computational results on 5 randomly generated SQBP instances. Note that we relaxed binary restrictions on the second stage

For $k = 2, \ldots, K$, the Lagrangian relaxation associated with scenario k, is as

$$D_k'''(\mu,\theta,\lambda) = \max \left\{ \sum_{i=1}^{n_1} \left(p^k c_i + \mu_i + \alpha_i^k (\sum_{j=i+1}^{n_1} \theta_{ij} + \sum_{j=1}^{i-1} \theta_{ij}) - \sum_{j=i+1}^{n_1} \beta_{ij}^k \lambda_{ij} - \sum_{j=1}^{i-1} \beta_{ji}^k \lambda_{ji} \right) x_i^k \right.$$

$$+ \sum_{i=1}^{n_2} p^k d_i^k y_i^k + \sum_{i=1}^{n_2-1} \sum_{j=2}^{n_2} p^k d_{ij}^k y_i^k y_j^k$$

$$\left. + \sum_{i=1}^{n_1-1} \sum_{j=2}^{n_1} (c_{ij} - (\alpha_i^k + \alpha_j^k)\theta_{ij} + \beta_{ij}^k \lambda_{ij}) z_{ij} + \sum_{i=1}^{n_1-1} \sum_{j=2}^{n_1} \lambda_{ij} \;\middle|\; (x^k, y^k) \in \mathcal{F}^k, z_{ij} \in \{0,1\} \right\}. \tag{33}$$

Therefore, the corresponding Lagrangian dual, denoted by z_{3A}^{LD}, is as

$$z_{3A}^{LD} = \min_{\mu,\theta,\lambda} \sum_{k=1}^{K} D_k'''(\mu,\theta,\lambda). \tag{34}$$

Note that in each of the above Lagrangian relaxations, variables z_{ij} are again unconstrained except for the bounds. Hence, they are easily obtained.

Although it is clear that both $z_{3A}^{LD} \geq z_3^{LD}$ and $z_{2Q}^{LD} \geq z_3^{LD}$, it is unclear the comparison between z_{3A}^{LD} and z_{2Q}^{LD}. We explore the computational tradeoff between the two bounds with preliminary numerical studies presented next.

3.3 Preliminary Numerical Study

We conducted preliminary computational experiments to investigate two Lagrangian relaxation bounds z_{3A}^{LD} and z_{2Q}^{LD}. At the initial stage of computational experimentation, we relaxed the binary restrictions on the second-stage decision variables to be able to test more and larger instances. We implemented the test instance generator in Python. To compute z_{3A}^{LD} and z_{2Q}^{LD}, we applied a standard subgradient method and used Cplex 12.0 when solving the required linear programs.

In Table 1, we present the comparative results for five small test instances, each of which has ten first-stage decision variables, twenty second-stage decision variables, two first-stage constraints, two second-stage constraints, twenty scenarios, i.e., $n_1 = 10, n_2 = 20, m_1 = 2, m_2 = 2, K = 20$. In the table, the numbers in the parentheses are the associated CPU times in seconds. The computational results suggest that in general, computing z_{3A}^{LD} is less time consuming than computing z_{2Q}^{LD}. But it is not clear on the superiority between the two bounds. Our conclusions hold for other classes of test instances.

3.4 A Lagrangian Decomposition Approach for QBPs

3.4.1 Introduction

One of the most well-known and studied classes of nonlinear integer optimization problems is the maximization of a quadratic 0–1 function subject to a set of linear 0–1 constraints:

$$\text{(P0)}: \quad \max_x \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \frac{1}{2} c_{ij} x_i x_j \tag{35}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} a_{ki} x_i \leq b_k, \text{ for } k = 1, \ldots, m; \tag{36}$$

$$x_i \in \{0, 1\}, \text{ for } i = 1, \ldots, n,$$

where $c_i, c_{ij} \in \mathbb{R}$ for $1 \leq i \neq j \leq n$ and $b_k, a_{ki} \in \mathbb{R}$ for $k = 1, \ldots, m$ and $i = 1, \ldots, n$. Problem (P0) is typically referred to as a constrained *quadratic binary problem* (QBP) [25]. Since for each binary variable $x_i^2 = x_i$, then $c_{ii} x_i^2$ can be rewritten as $c_i x_i$ with $c_i = c_{ii}$. Without loss of generality, we also assume that $c_{ij} = c_{ji}$ for any pair (i, j) with $i \neq j$.

A quadratic term that represents a pair of binary variables arises naturally in modeling interactions among entities. Furthermore, given the fact that optimization of general pseudo-boolean functions can be reduced in polynomial time to optimization of a quadratic binary function [25], QBP is, arguably, the most important class of the general pseudo-boolean optimization problem. Many important problems in engineering, physics, chemistry, biology, medicine and a variety of other application domains, can be formulated as QBPs. To name a few, such problems have been studied in scheduling [10], computer-aided design [17, 84], solid-state physics [15, 17], protein design [62, 87], computational biology [60], and epileptic seizure prediction [81]. In addition, many graph-theoretic problems can be naturally formulated with QBPs, including well-studied maximum clique and maximum independent set problems [2, 115, 120, 121].

There are only a limited number of classes of QBPs known to be polynomially solvable [11, 16, 25, 117, 123]. In general, QBPs are NP-hard combinatorial optimization problems. Even if we know that the global optimum is unique, QBPs remain NP-hard [118]. In terms of solving general QBPs, we have witnessed the development of various heuristics [65, 66, 95, 111, 112, 119] and exact solution methods.

Most of the exact solution methods are focused on efficient linearization techniques [3, 4, 25, 64, 109, 110]. The main concept of the linearization is to reformulate the original QBP as an equivalent linear mixed 0–1 problem, which can be solved efficiently with off-the-shelf mixed-integer linear programming solvers such as the CPLEX MIP solver (*www.ilog.com/products/cplex*). Although the proposed reformulations in the above references may differ significantly, almost all of them share the same key idea. That is, replacing the nonlinear terms with auxiliary variables and adding an additional set of linear constraints. In the most commonly used linearization reformulation (i.e., the standard linearization as it is termed in this chapter), a new variable z_{ij} is introduced to replace each cross term $x_i x_j$, which results in the following formulation:

$$\text{(PSL)}: \quad \max \left\{ \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j>i} c_{ij} z_{ij} \right\} \tag{37}$$

$$\sum_{i=1}^{n} a_{ki}x_i \leq b_k,\ k = 1, \ldots, m; \tag{38}$$

$$z_{ij} \geq x_i + x_j - 1,\quad i, j = 1, \ldots, n; i < j; \tag{39}$$

$$x_i \geq z_{ij},\quad i, j = 1, \ldots, n; i < j; \tag{40}$$

$$x_j \geq z_{ij},\quad i, j = 1, \ldots, n; i < j; \tag{41}$$

$$x_i \in \{0, 1\}, z_{ij} \geq 0,\ i, j = 1, \ldots, n, i < j. \tag{42}$$

Note that in cases of constrained QBPs, additional valid inequalities can be introduced via reformulation-linearization techniques (RLT) [138], which often improve the performance of the linearization. Other exact solution methods use approaches based on branch-and-bound and cutting plane [79, 80, 115] techniques.

A relatively less explored class of methods is based on the idea of making copies of decision variables and introducing additional constraints to ensure the identity between each original decision variable and its copies. The most notable decomposition method is the one developed by Chardaire and Sutter [38] for unconstrained QBPs. Subsequently, Billionnet et al. [21, 22] extended their work to QBPs with knapsack constraints. As a special case of their method, for each decision variable x_i, an auxiliary decision variable y_i^j is introduced for each $j = 1, \ldots, n, j \neq i$. To ensure the equivalent reformulation, additional linear constraints $y_i^j = x_i$ are enforced for all $i, j = 1, \ldots, n, j \neq i$. Therefore, in the simplest case, their reformulation is given as:

$$\text{(P1)}:\ \max_{x,y} \left\{ \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \frac{1}{2} c_{ij} x_i y_j^i \right\} \tag{43}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} a_{ki}x_i \leq b_k,\ k = 1, \ldots, m; \tag{44}$$

$$y_i^j = x_i,\ i, j = 1, \ldots, n, i \neq j; \tag{45}$$

$$x_i, y_i^j \in \{0, 1\},\ i, j = 1, \ldots, n, i \neq j.$$

With decision variables y, each term $c_{ij}x_ix_j$ in objective (35) is replaced by $c_{ij}x_iy_j^i$ in (43). To solve reformulation (43)–(46), Chardaire and Sutter [38] applied Lagrangian relaxation on constraints (45). To improve the Lagrangian upper bounding performance, the authors also imposed to (P1) the following set of nonlinear constraints

$$x_i y_j^i = x_j y_i^j,\ i, j = 1, \ldots, n, i < j. \tag{46}$$

The authors showed that the derived Lagrangian relaxation bound when relaxing both (45) and (46) is the same as the bound obtained by the LP relaxation of (PSL), denoted by B_{SL}. To solve the Lagrangian dual problem, the authors proposed a partial enumeration method that fixes the value of each decision variable x and then solves the resultant linear programs with respect to copy variables y. Unfortunately, the partial enumeration method is not computationally appealing [124], especially for constrained problems. The main reason for this is that it does not fully decompose variables x and y by relaxing the constraint $x_i = y_i^j$ for all $i, j = 1, \ldots, n, i \neq j$.

In this chapter, we present a new Lagrangian decomposition based approach, which results in an alternative Lagrangian dual problem that can be solved more efficiently. In our approach, we still introduce decision variable y_j^i for each pair (i, j), $i \neq j$, and replace each $c_{ij}x_ix_j$ with $c_{ij}x_iy_j^i$ in the objective function. However, instead of using linear constraints, i.e., constraints (45), and

quadratic constraints of simple form, i.e., constraints (46), we introduce parameterized quadratic constraints of a more complex form. We show that we can still establish the formulation equivalence with certain specification on the parameters of the introduced quadratic constraints even without imposing linear constraints $x_i = y_i^j$ for all $i, j = 1, \ldots, n, i \neq j$. More importantly, when we apply Lagrangian relaxation on those quadratic constraints, the resultant Lagrangian dual problem can be decomposed to n linear binary programs that only involves decision variables y and one linear binary program that only involves decision variables x. In addition, we provide properties to characterize our Lagrangian relaxation bound. By carefully choosing the parameters of the introduced quadratic constraints, we can guarantee that our Lagrangian bounds are better than B_{SL}. Our preliminary computational experiments demonstrate the superiority of a branch-and-bound algorithm with the proposed Lagrangian relaxation bound.

The remainder of this section is organized as follows. In Section 3.4.2, we describe the new Lagrangian decomposition method that provides upper bounds for (P0). This subsection includes introduction and parameter specification of our proposed quadratic constraints, derivation of decomposable Lagrangian duals, and characterization of the derived upper bounds. In Section 3.4.3, we illustrate our method with several classic QBPs and discuss the proposed decomposition in more detail. In Section 3.4.4, we offer some computational considerations on how to integrate our bounding method into a branch-and-bound (B&B) framework. In Section 3.4.5, we report encouraging results of our preliminary computational experiments on randomly generated test instances from two classes of QBPs and compare our results with solving two well-known linearization formulations directly via CPLEX.

3.4.2 A Lagrangian Decomposition Method

In this section, we first present our reformulation of (P0). We then show the equivalence between (P0) and the reformulation with certain specification on the introduced quadratic constraints. With the proposed reformulation, we finally introduce a new Lagrangian decomposition method that allows us to obtain a Lagrangian relaxation bound by solving only linear binary programs.

An Alternative Reformulation of (P0)

Consider the following parameterized quadratic binary problem, which is constructed by including into (P0) a set of auxiliary decision variables along with additional quadratic constraints.

$$\text{(P2)}: \max_{(x,y)} \sum_{i=1}^{n} \left(c_i x_i + \sum_{j=1, j\neq i}^{n} \frac{1}{2} c_{ij} x_i y_j^i \right) \tag{47}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} a_{ki} x_i \leq b_k, \ k = 1, \ldots, m; \tag{48}$$

$$\alpha_{ij}^l x_i y_j^i + \beta_{ij}^l x_j y_i^j + \theta_{ij}^l x_i + \gamma_{ij}^l x_j \geq \epsilon_{ij}^l, \ i, j = 1, \ldots, n, i < j, l = 1, \ldots, r_{ij}; \tag{49}$$

$$x_i, y_i^j \in \{0, 1\}, \ i, j = 1, \ldots, n, i \neq j.$$

In (P2), a binary decision variable y_j^i variable is introduced for each $j \neq i$ to pair with x_i. Hence, $n-1$ auxiliary variables y_i^j are introduced for each x_i. Note that we do not introduce to (P2) any additional linear constraint that links x and y. Instead, we introduce quadratic constraints to link them. We let r_{ij} be the number of quadratic constraints for each index pair (i, j) with $i < j$ and

allow this number to differ among index pairs. Also note that objective function (47) is identical to (43).

Next we show the equivalence between (P1) and (P2) under certain parameter specifications. Meanwhile, it is easy to see that solving (P1) yields an optimal solution to (P0). Hence, we can establish the validity of solving (P2). For convenience of the exposition, we use shorthand notation (x, y) to denote $(x_1, \ldots, x_n, y_2^1, \ldots, y_n^1, \ \ldots, y_1^n, \ldots, y_{n-1}^n)$ in the following results and proofs. We denote $\mathcal{A} = \{(x, y) \in \{0,1\}^n \times \{0,1\}^{n(n-1)} \ \big| \ (44) - (46)\}$ to be the feasible solution region of (P1) and $\mathcal{B}(\alpha, \beta, \theta, \gamma, \epsilon) = \{(x, y) \in \{0,1\}^n \times \{0,1\}^{n(n-1)} \ \big| \ (48) - (50)\}$ to be the feasible solution region of (P2) parameterized on $\alpha := (\alpha_{ij}), \beta := (\beta_{ij}), \theta := (\theta_{ij}), \gamma := (\gamma_{ij})$, and $\epsilon = (\epsilon_{ij})$. For notational simplicity, we use $\mathcal{B}$ instead of $\mathcal{B}(\alpha, \beta, \theta, \gamma, \epsilon)$ when referring to a parameterized feasible solution region of (P2). The parameters are always specified when making such a reference. We also let $C_1 = \{(1,1,1,1), (1,0,1,0), (0,1,0,1),$
$(0,0,0,0)\}$, $C_2 = \{(1,0,0,0), \ (0,1,0,0), (0,0,0,1), \ (0,0,1,0), (0,0,1,1)\}$, and $C_3 = \{0,1\}^4 \backslash (C_1 \cup C_2)$. To ensure the equivalence between (P1) and (P2), we essentially need to show that $\mathcal{A} \subseteq \mathcal{B}$ and no feasible solution to (P2) is from C_3. Therefore, the parameters have to be specified accordingly.

Proposition 1 *If for all (i, j), $i, j = 1, \ldots, n$ and $i < j$, the following conditions hold:*

$$\alpha_{ij}^l + \beta_{ij}^l + \theta_{ij}^l + \gamma_{ij}^l \geq \epsilon_{ij}^l; \tag{50}$$

$$\theta_{ij}^l \geq \epsilon_{ij}^l; \tag{51}$$

$$\gamma_{ij}^l \geq \epsilon_{ij}^l; \tag{52}$$

$$\epsilon_{ij}^l \leq 0, \tag{53}$$

for all $l = 1, \ldots, r_{ij}$, then $\mathcal{A} \subseteq \mathcal{B}$.

Proof: Given a $(x, y) \in \mathcal{A}$, it is easy to see that $(x_i, x_j, y_i^j, y_j^i) \in C_1$ for every index pair (i, j). Hence, a sufficient condition for $\mathcal{A} \subseteq \mathcal{B}$ is to ensure that all four vectors in C_1 satisfy constraint (49) when conditions (50)–(53) hold for all $l = 1, \ldots, r_{ij}$. We consider four cases to verify the sufficient condition and summarize the verification in the following table:

(x_i, x_j, y_i^j, y_j^i)	Constraint (49)	Ensured by condition
$(1,1,1,1)$	$\alpha_{ij}^l + \beta_{ij}^l + \theta_{ij}^l + \gamma_{ij}^l \geq \epsilon_{ij}^l$	(50)
$(1,0,1,0)$	$\theta_{ij}^l \geq \epsilon_{ij}^l$	(51)
$(0,1,0,1)$	$\gamma_{ij}^l \geq \epsilon_{ij}^l$	(52)
$(0,0,0,0)$	$\epsilon_{ij}^l \leq 0$	(53)

For example, if $(x_i, x_j, y_i^j, y_j^i) = (1,1,1,1)$, each constraint (49) is reduced to $\alpha_{ij}^l + \beta_{ij}^l + \theta_{ij}^l + \gamma_{ij}^l \geq \epsilon_{ij}^l$, which coincides with condition (50). The proposition then follows as any solution (x, y) in $\mathcal{A}$ is also in $\mathcal{B}$. □

Proposition 1 implies that with parameter specification as in (50) – (53), we ensure that any combination in C_1 must satisfy all corresponding constraints (49) for every index pair.

Proposition 2 *Consider any $(x, y) \in \mathcal{B}$. If $(x_i, x_j, y_i^j, y_j^i) \in C_1 \cup C_2$ for each (i, j), $i, j = 1, \ldots, n$ and $i < j$, then there exists $(x, \hat{y}) \in \mathcal{A}$ such that (x, y) in (P2) and $(x, \hat{y})$ in (P1) yield the same objective function value.*

Proof: Given a $(x, y) \in \mathcal{B}$, we consider two cases to construct $(x, \hat{y})$. For each index pair (i, j), 1) if $(x_i, x_j, y_i^j, y_j^i) \in C_1$, we simply let $(x_i, x_j, \hat{y}_i^j, \hat{y}_j^i) = (x_i, x_j, y_i^j, y_j^i)$; 2) if $(x_i, x_j, y_i^j, y_j^i) \in C_2$, we construct $(x_i, x_j, \hat{y}_i^j, \hat{y}_j^i)$ in the following manner:

(x_i, x_j, y_i^j, y_j^i)	$(x_i, x_j, \hat{y}_i^j, \hat{y}_j^i)$
$(1,0,0,0)$	$(1,0,1,0)$
$(0,1,0,0)$	$(0,1,0,1)$
$(0,0,0,1)$	$(0,0,0,0)$
$(0,0,1,0)$	$(0,0,0,0)$
$(0,0,1,1)$	$(0,0,0,0)$

It is easy to see that $(x, \hat{y}) \in \mathcal{A}$ with the above construction. Regarding the objective functions, it is clear that for any pair (i,j), $c_{ij}x_i y_j^i + c_{ji}x_j y_i^j = c_{ij}x_i \hat{y}_j^i + c_{ji}x_j \hat{y}_i^j$ given $(x,y) \in \mathcal{B}$ and $(x,\hat{y}) \in \mathcal{A}$. The proposition then follows as the two objective function values are equal. □

Proposition 2 implies that although $\mathcal{A} \subseteq \mathcal{B}$ allows more feasible solutions to (P2) than (P1), such feasible solution region expansion does not affect the optimality equivalence between (P1) and (P2) as long as for all (i,j), $1 \le i < j \le n$, $(x_i, x_j, y_i^j, y_j^i) \in C_1 \cup C_2$. Next, we show certain expansion should not be allowed as it may affect the optimality equivalence between (P1) and (P2). Thus, we should prohibit such expansion by specifying parameters in constraint (49). In the following, we provide a sufficient condition to ensure this.

Proposition 3 *Suppose for some (s,t), $s,t = 1,\ldots,n$ and $s < t$, there exists an index set $L_{st} := (l_1,l_2,l_3,l_4,l_5) \in \{1,\ldots,r_{st}\}^5$ such that the following conditions hold:*

$$\alpha_{st}^{l_1} + \theta_{st}^{l_1} + \gamma_{st}^{l_1} < \epsilon_{st}^{l_1}; \tag{54}$$
$$\beta_{st}^{l_2} + \theta_{st}^{l_2} + \gamma_{st}^{l_2} < \epsilon_{st}^{l_2}; \tag{55}$$
$$\theta_{st}^{l_3} + \gamma_{st}^{l_3} < \epsilon_{st}^{l_3}; \tag{56}$$
$$\alpha_{st}^{l_4} + \theta_{st}^{l_4} < \epsilon_{st}^{l_4}; \tag{57}$$
$$\beta_{st}^{l_5} + \gamma_{st}^{l_5} < \epsilon_{st}^{l_5}. \tag{58}$$

For any (x,y), if $(x_s, x_t, y_s^t, y_t^s) \in C_3$, then $(x,y) \notin \mathcal{B}$.

Proof: Given (s,t) with $(x_s, x_t, y_s^t, y_t^s) \in C_3$, we check in the following table whether such (x_s, x_t, y_s^t, y_t^s) violates constraint (49) with a set of parameters specified in (54)–(58) and (l_1,l_2,l_3,l_4,l_5).

(x_i, x_j, y_i^j, y_j^i)	Constraint (49)	Violated by condition
$(1,1,0,1)$	$\alpha_{st}^{l_1} + \theta_{st}^{l_1} + \gamma_{st}^{l_1} \ge \epsilon_{st}^{l_1}$	(54)
$(1,1,1,0)$	$\beta_{st}^{l_2} + \theta_{st}^{l_2} + \gamma_{st}^{l_2} \ge \epsilon_{st}^{l_2}$	(55)
$(1,1,0,0)$	$\theta_{st}^{l_3} + \gamma_{st}^{l_3} \ge \epsilon_{st}^{l_3}$	(56)
$(1,0,0,1)$	$\alpha_{st}^{l_4} + \gamma_{st}^{l_4} \ge \epsilon_{st}^{l_4}$	(57)
$(1,0,1,1)$	$\alpha_{st}^{l_4} + \gamma_{st}^{l_4} \ge \epsilon_{st}^{l_4}$	(57)
$(0,1,1,0)$	$\beta_{st}^{l_5} + \gamma_{st}^{l_5} \ge \epsilon_{st}^{l_5}$	(58)
$(0,1,1,1)$	$\beta_{st}^{l_5} + \gamma_{st}^{l_5} \ge \epsilon_{st}^{l_5}$	(58)

Note that the seven cases of (x_s, x_t, y_s^t, y_t^s) in the above table show all possible combinations in C_3. The violation checking in the table indicates that (x_s, x_t, y_s^t, y_t^s) cannot satisfy (49). Hence, $(x,y) \notin \mathcal{B}$. □

Note that it is allowed in Proposition 3 that $l_s = l_t$ for $1 \le s \ne t \le 5$ and the choice of L_{st} may not be unique for a given index pair (s,t). Proposition 3 implies that $\mathcal{B}$ should not contain any (x,y) with component as any of the seven combinations in C_3, since such a solution may destroy the optimality equivalence between (P1) and (P2). With the three propositions presented earlier, we readily state one sufficient condition to ensure (P2) is a valid reformulation of (P0).

Theorem 1 *The two formulations (P1) and (P2) are equivalent in the sense that they yield the same optimal objective function value and the same optimal solution with respect to x if there exists $(\alpha, \beta, \theta, \gamma, \epsilon)$, such that for any index pairs (i, j), $i, j = 1, \ldots, n$ and $i < j$, the following conditions hold:*

1. *for all $l = 1, \ldots, r_{ij}$,*

$$\alpha_{ij}^l + \beta_{ij}^l + \theta_{ij}^l + \gamma_{ij}^l \geq \epsilon_{ij}^l, \tag{59}$$

$$\theta_{ij}^l \geq \epsilon_{ij}^l; \tag{60}$$

$$\gamma_{ij}^l \geq \epsilon_{ij}^l; \tag{61}$$

$$\epsilon_{ij}^l \leq 0. \tag{62}$$

2. *there exists an index set L_{ij}, i.e. $(l_1^{ij}, l_2^{ij}, l_3^{ij}, l_4^{ij}, l_5^{ij})$ such that*

$$\alpha_{ij}^{l_1^{ij}} + \theta_{ij}^{l_1^{ij}} + \gamma_{ij}^{l_1^{ij}} < \epsilon_{ij}^{l_1^{ij}}; \tag{63}$$

$$\beta_{ij}^{l_2^{ij}} + \theta_{ij}^{l_2^{ij}} + \gamma_{ij}^{l_2^{ij}} < \epsilon_{ij}^{l_2^{ij}}; \tag{64}$$

$$\theta_{ij}^{l_3^{ij}} + \gamma_{ij}^{l_3^{ij}} < \epsilon_{ij}^{l_3^{ij}}; \tag{65}$$

$$\alpha_{ij}^{l_4^{ij}} + \theta_{ij}^{l_4^{ij}} < \epsilon_{ij}^{l_4^{ij}}; \tag{66}$$

$$\beta_{ij}^{l_5^{ij}} + \gamma_{ij}^{l_5^{ij}} < \epsilon_{ij}^{l_5^{ij}}. \tag{67}$$

Proof: Let f_1^* and f_2^* be the optimal objective function values of (P1) and (P2), respectively. From Proposition 1, we have $\mathcal{A} \subseteq \mathcal{B}$, and thus $f_1^* \leq f_2^*$. Propositions 2 and 3 show that for any $(x, y) \in \mathcal{B}$, there exists $(x, \hat{y}) \in \mathcal{A}$ that yields the same objective function value with (x, y), which implies that $f_2^* \leq f_1^*$ and thus the result follows. Furthermore, given any $(x, y) \in \mathcal{B}$, the construction of $(x, \hat{y}) \in \mathcal{A}$ in Proposition 2 ensures that the solutions are always identical with respect to x, which completes the proof. □

Corollary 1 *The formulation (P2) is a valid reformulation of (P0) with parameter specification given in Theorem 1.*

Theorem 1 and Corollary 1 establish the fact that with appropriate parameter specification in constraints (49), (P2) is a valid reformulation of (P0). Thus one may solve (P2) instead of (P0). Two issues remain for the algorithmic development. One issue is how to solve (P2) efficiently. The other is how to specify the parameters in constraints (49) so that we ensure not only the formulation equivalence but also the solution efficiency in practice. In Section 3.2.2, we develop an Lagrangian decomposition method to improve the efficiency of solving (P2). In Section 3.2.3, we propose some practical specifications of the parameters in constraints (49) and discuss their resultant Lagrangian relaxation bounds.

Lagrangian Decomposition in (P2)

In this section, we present a Lagrangian decomposition method to solve the reformulation (P2). We assume that the parameters in constraints (49) have been specified to ensure the validity of (P2). For each pair $1 \leq i < j \leq n$ and each $l = 1, \ldots, r_{ij}$, we associate a Lagrangian multiplier λ_{ij}^l with the corresponding constraint (49). Then the Lagrangian dual problem of (P2), denoted by $L(\lambda)$, is

$$L(\lambda) = \max_{x,y} \{L(\lambda, x, y) \mid (48), (50)\}, \tag{68}$$

where $L(\lambda, x, y) = f(x, y) + \sum_{i=1}^{n} \sum_{j>i} \sum_{l=1}^{r_{ij}} \left(\lambda_{ij}^l (\alpha_{ij}^l x_i y_j^i + \beta_{ij}^l x_j y_i^j + \theta_{ij}^l x_i + \gamma_{ij}^l x_j - \epsilon_{ij}^l) \right) =$

$$\sum_{i=1}^{n} \left(c_i + \sum_{j>i} \sum_{l=1}^{r_{ij}} \lambda_{ij}^l \theta_{ij}^l + \sum_{j<i} \sum_{l=1}^{r_{ji}} \lambda_{ji}^l \gamma_{ji}^l + \sum_{j>i} (\frac{1}{2} c_{ij} + \sum_{l=1}^{r_{ij}} \lambda_{ij}^l \alpha_{ij}^l) y_j^i + \sum_{j<i} (\frac{1}{2} c_{ij} + \sum_{l=1}^{r_{ji}} \lambda_{ji}^l \beta_{ji}^l) y_j^i \right) x_i - \sum_{i=1}^{n} \sum_{j>i} \sum_{l=1}^{r_{ij}} \lambda_{ij}^l \epsilon_{ij}^l, \quad (69)$$

and the optimal Lagrangian dual of (P2), denoted by B_{LD}, is:

$$B_{LD} = \min_{\lambda \geq 0} L(\lambda). \quad (70)$$

Lemma 1 *Consider a function $f(x, y)$ with form $f(x, y) = g(y)h(x)$ with $x \in X$ and $y \in Y$. If $h(x) \geq 0$ for all $x \in X$, then $\max_{x \in X, y \in Y} f(x, y) = \max_{x \in X, y \in Y} g(y)h(x) = \max_{x \in X} \{(\max_{y \in Y} g(y))h(x)\}$.*

With Lemma 1, we can further derive $L(\lambda)$ as:

$$\begin{aligned} L(\lambda) = \max_{x,y} L(\lambda, x, y) &= \max_{x,y} \left\{ \sum_{i=1}^{n} \left(g_i(y^i) x_i \right) \right\} - \sum_{i=1}^{n} \sum_{j>i} \sum_{l=1}^{r_{ij}} \lambda_{ij}^l \epsilon_{ij}^l \\ &= \max_{x} \left\{ \sum_{i=1}^{n} \left(\max_{y^i} g_i(y^i) \right) x_i \right\} - \sum_{i=1}^{n} \sum_{j>i} \sum_{l=1}^{r_{ij}} \lambda_{ij}^l \epsilon_{ij}^l, \end{aligned} \quad (71)$$

where

$$g_i(y^i) = c_i + \sum_{j>i} \sum_{l=1}^{r_{ij}} \lambda_{ij}^l \theta_{ij}^l + \sum_{j<i} \sum_{l=1}^{r_{ji}} \lambda_{ji}^l \gamma_{ji}^l + \sum_{j>i} (\frac{1}{2} c_{ij} + \sum_{l=1}^{r_{ij}} \lambda_{ij}^l \alpha_{ij}^l) y_j^i + \sum_{j<i} (\frac{1}{2} c_{ij} + \sum_{l=1}^{r_{ji}} \lambda_{ji}^l \beta_{ji}^l) y_j^i, \quad (72)$$

for $i = 1, \ldots, n$. Note that we drop the constraints in (71) for simplicity of the exposition. Also note that the mutual independence among $g_i(y^i)$ along with Lemma 1 ensure the second equality in (71). Hence, we can decompose $L(\lambda)$ to n unconstrained linear binary problems and one linear binary program. For each $i = 1, \ldots, n$, let

$$g_i^* = \max_{y^i} g_i(y^i), \quad (73)$$

which is an unconstrained linear binary program only involving y^i. Then $L(\lambda)$ is further derived as:

$$L(\lambda) = \max_{x} \left\{ \sum_{i=1}^{n} g_i^* x_i - \sum_{i=1}^{n} \sum_{j>i} \sum_{l=1}^{r_{ij}} \lambda_{ij}^l \epsilon_{ij}^l \;\middle|\; (48), (50) \right\}, \quad (74)$$

which is a linear binary program only involving x. We call (74) the **master subproblem after decomposition (MSAD)**.

By applying Lemma 1, we can compute $L(\lambda)$ by solving n unconstrained linear binary problems, each of which corresponds to y^i, $i = 1, \ldots, n$, and one linear binary program, which corresponds

to x. It is easy to solve each unconstrained integer program (IP) $\max_{y^i} g_i(y^i)$ with $n-1$ binary variables. Thus, generally speaking, the complexity of solving $L(\lambda)$ is determined by the complexity of solving MSAD. In addition, we note that MSAD preserves the feasible solution region of the original problem (P0) whereas linearization methods introduce additional constraints to (P0) (e.g., [64, 68, 109]), which may significantly complicate the original problem structure. With the feature of structure preserving, our method is expected to be efficient, especially for problems with special structures. Furthermore, it is expected to be beneficial when embedding our method within a branch-and-bound framework, as indicated in the following remark.

Remark 4 *For any given $\lambda \geq 0$, solving MSAD provides a feasible solution to (P0) while $L(\lambda)$ offers an upper bound to (P0).*

In a Lagrangian decomposition based branch-and-bound framework, it may be computationally useful to terminate the solution of a Lagrangian relaxation problem before it reaches the optimality. In our case, we terminate a subgraident algorithm for solving problem (70) prematurely and use the obtained Lagrangian dual as the upper bound instead of B_{LD}. Note that we also obtain a potentially promising feasible solution at the same time with the upper bound. In section 3.4.4, We state computational considerations on solving the Lagrangian duals via the subgradient method and discuss their effects in a branch-and-bound framework.

Parameter Specification in Quadratic Constraints (49)

To ensure the equivalence of (P0) and (P2), we need to specify parameters in the set of quadratic constraints (49). Theorem 1 provides a sufficient condition on the parameter specification. However, that sufficient condition leads to infinitely many feasible parameter specifications. In this section, we assume that a constant number of quadratic constraints for each index pair, i.e., $r_{ij} = r$ for all $1 \leq i < j \leq n$ and discuss only some particular subsets of the parameter space. We consider $r = 1, 2, 6$ and identify some characterizations of the obtained quadratic constraints.

Case $r = 1$

As indicated in Propositions 1 and 3, a necessary condition for valid specification of $(\alpha, \beta, \theta, \gamma, \epsilon)$ is that for any index pair (i, j) with $1 \leq i < j \leq n$, the corresponding quadratic constraint (49), $\alpha_{ij} x_i y_j^i + \beta_{ij} x_j y_i^j + \theta_{ij} x_i + \gamma_{ij} x_j \geq \epsilon_{ij}$, is satisfied by $(x_i, x_j, y_i^j, y_j^i) = (1, 1, 1, 1)$ and $(0, 0, 0, 0)$ but violated by $(x_i, x_j, y_i^j, y_j^i) = (0, 1, 1, 1)$ and $(1, 0, 0, 1)$. Therefore, the parameters $(\alpha_{ij}, \beta_{ij}, \theta_{ij}, \gamma_{ij}, \epsilon_{ij})$ must satisfy the following conditions:

$$\begin{aligned} \alpha_{ij} + \beta_{ij} + \theta_{ij} + \gamma_{ij} &\geq \epsilon_{ij}; \\ \epsilon_{ij} &\leq 0; \\ \beta_{ij} + \gamma_{ij} &< \epsilon_{ij}; \\ \alpha_{ij} + \theta_{ij} &< \epsilon_{ij}, \end{aligned}$$

for $1 \leq i < j \leq n$. It is easy to see that the above four conditions can not be satisfied simultaneously, which leads to the following remark.

Remark 5 *A necessary condition to ensure the equivalence between (P0) and (P2) is $r_{ij} \geq 2$ for any index pair (i, j) with $1 \leq i < j \leq n$, in the parameter specification of quadratic constraints (49).*

Case $r = 2$

In the case $r = 2$, there are two quadratic constraints (49) for each index pair, which requires us to specify twice as many parameters as in the case $r = 1$. This gives us more freedom to specify the parameters that satisfy the conditions in Theorem 1. Namely, we must find a set of parameters such that (59)–(62) and (63)–(67) hold simultaneously for each index pair. We next present a valid way to specify the parameters to ensure the equivalence between (P1) and (P2).

For each index pair (i, j), $1 \leq i < j \leq n$, let $\alpha_{ij} := \alpha^1_{ij} = \beta^2_{ij}$, $\beta_{ij} := \alpha^2_{ij} = \beta^1_{ij}$, $\theta_{ij} := \theta^1_{ij} = \gamma^2_{ij}$, $\gamma_{ij} := \theta^2_{ij} = \gamma^1_{ij}$ and $\epsilon_{ij} := \epsilon^1_{ij} = \epsilon^2_{ij}$. Hence, conditions (59) – (62) can be further derived as:

$$\begin{aligned} \alpha_{ij} + \beta_{ij} + \theta_{ij} + \gamma_{ij} &\geq \epsilon_{ij}; && (75)\\ \theta_{ij} &\geq \epsilon_{ij}; && (76)\\ \gamma_{ij} &\geq \epsilon_{ij}; && (77)\\ \epsilon_{ij} &\leq 0, && (78) \end{aligned}$$

for each (i, j), $1 \leq i < j \leq n$. Furthermore, we set $l_1 = 2, l_2 = 1, l_3 = 1, l_4 = 2, l_5 = 1$ for each (i, j), $1 \leq i < j \leq n$. Hence, conditions (63) – (67) can be further derived as:

$$\begin{aligned} \beta_{ij} + \theta_{ij} + \gamma_{ij} &< \epsilon_{ij}, && (79)\\ \theta_{ij} + \gamma_{ij} &< \epsilon_{ij}, && (80)\\ \beta_{ij} + \gamma_{ij} &< \epsilon_{ij}, && (81) \end{aligned}$$

for each (i, j), $1 \leq i < j \leq n$. Note that given the above specification on $(\alpha, \beta, \theta, \gamma, \epsilon)$, with $l_1 = 2$ and $l_2 = 1$, conditions (63) and (64) are identical and become (79); and with $l_4 = 2$ and $l_5 = 1$, conditions (66) and (67) are identical and become (81). Finally, in order to find a valid set of parameters, (78) has to be tightened to $\epsilon_{ij} < 0$. This is because (76), (77), and (80) together, imply $\epsilon_{ij} > 2\epsilon_{ij}$. Hence, we replace (78) with

$$\epsilon_{ij} < 0, \quad (82)$$

for each (i, j), $1 \leq i < j \leq n$, in the following discussion on the case $r = 2$.

Remark 6 *Given the selection on $(l_1, l_2, l_3, l_4, l_5)$ as above, for each index pair (i, j), $1 \leq i < j \leq n$, any solution $(\alpha_{ij}, \beta_{ij}, \theta_{ij}, \gamma_{ij}, \epsilon_{ij})$ that satisfies the set of inequalities (75)–(77), (79)–(81), and (82), is valid in terms of parameter specification. Piecing satisfactory solutions together for all index pairs ensures the equivalence between (P1) and (P2).*

Remark 7 *In the case $r = 2$, once conditions (75)–(77) and (82) are satisfied, one can find a valid set of parameters to ensure the equivalence (P1) and (P2) as long as $l_1 \neq l_2$ and $l_4 \neq l_5$ when selecting $(l_1, l_2, l_3, l_4, l_5)$.*

The above two remarks imply that there are still a large number of valid specifications to ensure the equivalence between (P1) and (P2). In the following, we fix the selection on $(l_1, l_2, l_3, l_4, l_5)$ as used earlier for conditions (79)–(81). However, we further simply the selection on $(\alpha, \beta, \theta, \gamma, \epsilon)$ with the goal of improving computational efficiency in solving the Lagrangian relaxation problem (70). To solve (70), we apply the subgradient method in Held and Wolfe [78], with which Lagrangian multipliers λ are updated at every iteration in the following fashion. For each (i, j) with $1 \leq i < j \leq n$ and $l = 1, 2$,

$$(\lambda^l_{ij})^{k+1} = \max\{0, (\lambda^l_{ij})^k + s(\epsilon^l_{ij} - \alpha^l_{ij} x_i y^i_j - \beta^l_{ij} x_j y^j_i - \theta^l_{ij} x_i - \gamma^l_{ij} x_j)\}, \quad (83)$$

where k is the iteration index and s is the step size. In general, it is desirable to update the Lagrangian multipliers only when (x_i, x_j, y_i^j, y_j^i) is infeasible to (P2). In other words, the fact that (x_i, x_j, y_i^j, y_j^i) satisfies (49), suggests that $(\lambda_{ij}^l)^k$ is likely to be sufficiently good and thus there is no need to further modify it. Therefore, conditions (75)–(77) should be satisfied with equality whenever it is possible , which leads to the following. For each (i, j), $1 \le i < j \le n$ and $l = 1, 2$, we set $\theta_{ij}^l = \gamma_{ij}^l = \epsilon_{ij}^l = \epsilon$ and $\beta_{ij}^l = -(\alpha_{ij}^l + \epsilon_{ij}^l)$. Note that with the above partial parameter specifications, we still need to set α_{ij}^l. Without loss of generality, we arbitrarily let $\alpha_{ij}^1 = \beta_{ij}^2 = 1$ and $\alpha_{ij}^2 = \beta_{ij}^1 = -1 - \epsilon$ for all (i, j). For each (i, j), $1 \le i < j \le n$, we thus have

$$x_i(y_j^i + \epsilon) + x_j((-1 - \epsilon)y_i^j + \epsilon) \geq \epsilon; \tag{84}$$

$$x_i((-1 - \epsilon)y_j^i + \epsilon) + x_j(y_i^j + \epsilon) \geq \epsilon. \tag{85}$$

Finally to satisfy (81) and (82), we must have $-1 < \epsilon < 0$. It is easy to check the above parameter specification satisfies the conditions in Theorem 1 and thus ensure the equivalence between (P1) and (P2) with (84) and (85) in place of (49) for each (i, j), $1 \le i < j \le n$. Given its simplicity, we use the above specification in all our computational experiments. Our preliminary computational results show that although it is likely that we cannot obtain a Lagrangian relaxation bound as good as B_{SL}, the bound obtained by solving the LP relaxation of (PSL), the computational benefit due to Lagrangian decomposition offsets the bounding inferiority. However, the inferiority motivates us to study cases where $r > 2$.

Case $r = 6$

As r increases, it is less a concern for identifying a set of parameters $(\alpha, \beta, \theta, \gamma, \epsilon)$ that satisfies the sufficient condition in Theorem 1 and thus ensures the equivalence of (P1) and (P2). In the case $r = 2$, we explore how to improve the computational efficiency in terms of solving the proposed Lagrangian relaxation problem. In this section, we intend to investigate good parameter specifications such that the the proposed Lagrangian relaxation bound can outperform B_{SL}.

We present a special case of $r = 6$, with which we show that one can generate an upper bound on (P0) at least as tight as B_{SL}. Let us consider the following specification of the parameters. For all index pairs (i, j) with $1 \le i < j \le n$, we set

$$\alpha_{ij}^1 = 1,\ \beta_{ij}^1 = 0,\ \theta_{ij}^1 = -1,\ \gamma_{ij}^1 = -1,\ \epsilon_{ij}^1 = -1; \tag{86}$$

$$\alpha_{ij}^2 = 0,\ \beta_{ij}^2 = 1,\ \theta_{ij}^2 = -1,\ \gamma_{ij}^2 = -1,\ \epsilon_{ij}^2 = -1; \tag{87}$$

$$\alpha_{ij}^3 = -1,\ \beta_{ij}^3 = 0,\ \theta_{ij}^3 = 0,\ \gamma_{ij}^3 = 1,\ \epsilon_{ij}^3 = 0; \tag{88}$$

$$\alpha_{ij}^4 = 0,\ \beta_{ij}^4 = -1,\ \theta_{ij}^4 = 1,\ \gamma_{ij}^4 = 0,\ \epsilon_{ij}^4 = 0; \tag{89}$$

$$\alpha_{ij}^5 = 1,\ \beta_{ij}^5 = -1,\ \theta_{ij}^5 = 0,\ \gamma_{ij}^5 = 0,\ \epsilon_{ij}^5 = 0; \tag{90}$$

$$\alpha_{ij}^6 = -1,\ \beta_{ij}^6 = 1,\ \theta_{ij}^6 = 0,\ \gamma_{ij}^6 = 0,\ \epsilon_{ij}^6 = 0. \tag{91}$$

It is clear that (86)–(91) satisfy the sufficient condition in Theorem 1. To be specific, for all (i, j) with $1 \le i < j \le n$, we have a) (86)–(91) all satisfy conditions (59)–(62); and b) (87) and (91) satisfy (63), (86) and (90) satisfy (64), (86) and (87) satisfy (65), (88) and (91) satisfy (66), and (89) and (90) satisfy (67). Therefore, one can select l_i, $i = 1, \ldots, 5$, accordingly. For example, by setting $(l_1, l_2, l_3, l_4, l_5) = (2, 1, 2, 3, 4)$, the condition in Theorem 1 is satisfied and thus the equivalence between (P1) and (P2) is established. Note that our selection of l_i, $i = 1, \ldots, 5$, indicates that it is sufficient to have the first four sets of parameters. It will be clear later in this section that the last two sets of parameters ensure the superiority of the derived Lagrangian relaxation bound.

With the parameters specified in (86)–(91), we rewrite quadratic constraints (49) to the following five constraints:

$$x_i y_j^i \geq x_i + x_j - 1; \tag{92}$$

$$x_j y_i^j \geq x_i + x_j - 1; \tag{93}$$

$$x_i \geq x_j y_i^j; \tag{94}$$

$$x_j \geq x_i y_j^i; \tag{95}$$

$$x_i y_j^i = x_j y_i^j, \tag{96}$$

for each (i, j), $1 \leq i < j \leq n$. Note that (90) yields $x_i y_j^i \geq x_j y_i^j$ and (91) yields $x_i y_j^i \leq x_j y_i^j$, we thus combine them and form the equality in (96).

By replacing $z_{ij} = x_i y_j^i = x_j y_i^j$ for each pair (i, j), $1 \leq i < j \leq n$, it is clear to see the similarity between constraints (92)–(96) and constraints (39)–(41). We next show that the Lagrangian relaxation on constraints (92)–(96), in fact, achieves an upper bound as least as tight as B_{SL}.

Theorem 2 *The upper bound obtained via the Lagrangian relaxation of (P2) on constraints (92)–(96) is at least as tight as the upper bound obtained via the LP relaxation of (PSL), i.e.* $B_{LD} \leq B_{SL}$.

Proof: It is clear that computing the optimal Lagrangian dual of (PSL) by relaxing (39)–(41) yields an upper bound at least as tight as the one obtained by solving the LP relaxation of (PSL) [57, 63]. Therefore, it suffices to prove that computing the optimal Lagrangian dual of (P2) by relaxing (92)–(96) yields an upper bound at least as tight as the one obtained by computing the optimal Lagrangian dual of (PSL).

For each index pair (i, j), $1 \leq i < j \leq n$, let us denote the Lagrangian multipliers associated with (39) to be ζ_{ij}, and denote those associated with (40) and (41) to be τ_{ij} and τ_{ji}, respectively. We use the short-hand notation ζ and τ to represent the vectors containing all Lagrangian multipliers ζ_{ij} and τ_{ij}. We then present the optimal Lagrangian dual of (PSL) as

$$G_{LD} = G(\zeta^*, \tau^*) = \min_{\zeta, \tau} G(\zeta, \tau) \tag{97}$$

where

$$G(\zeta, \tau) = \max_{x,z} \left\{ \sum_{i=1}^{n} (c_i + \sum_{j \neq i} \tau_{ij} - \sum_{j<i} \zeta_{ji} - \sum_{j>i} \zeta_{ij}) x_i + \sum_{i=1}^{n} \sum_{j>i} (c_{ij} + \zeta_{ij} - \tau_{ij} - \tau_{ji}) z_{ij} \right.$$
$$\left. + \sum_{i=1}^{n} \sum_{j>i} \zeta_{ij} \,\middle|\, (38), (42) \right\}. \tag{98}$$

Next let us denote the Lagrangian multipliers associated with (92)–(96) with $\nu_{ij}, \nu_{ji}, \delta_{ij}, \delta_{ji}$, and κ_{ij}, for each (i, j) with $1 \leq i < j \leq n$. Note that $\nu_{ij}, \nu_{ji}, \delta_{ij}, \delta_{ji}$ must be non-negative whereas κ_{ij} can be any real number. We then present the optimal Lagrangian dual of (P2) as

$$B_{LD} = L(\nu^*, \delta^*, \kappa^*) = \min_{\nu, \delta, \kappa} L(\nu, \delta, \kappa), \tag{99}$$

where

$$L(\nu, \delta, \kappa) = \max_{x,y} \left\{ \sum_{i=1}^{n} \left(c_i + \sum_{j \neq i} \delta_{ij} - \sum_{j \neq i} (\nu_{ij} + \nu_{ji}) \right) x_i + \sum_{i=1} \sum_{j>i} \left(\frac{1}{2} c_{ij} + \nu_{ij} - \delta_{ji} + \kappa_{ij} \right) y_j^i x_i \right.$$

$$+\sum_{i=1}\sum_{j>i}\left(\frac{1}{2}c_{ji}+\nu_{ji}-\delta_{ij}-\kappa_{ji}\right)y_i^j x_j+\sum_{i=1}^{n}\sum_{j>i}(\nu_{ij}+\nu_{ji})\;\middle|\;(48),(50)\right\}. \tag{100}$$

With the notation introduced above, we have $G_{LD} \leq B_{SL}$. Hence, we need to prove $B_{LD} \leq G_{LD}$ to complete the proof of the theorem.

Next we show for any feasible (ζ, τ), there exists a feasible (ν, δ, κ) such that $L(\nu, \delta, \kappa) \leq G(\zeta, \tau)$. If this is true, for (ζ^*, τ^*), the multipliers associated with the optimal Lagrangian dual of (PSL), there must exist some feasible $(\hat{\nu}, \hat{\delta}, \hat{\kappa})$ such that $L(\hat{\nu}, \hat{\delta}, \hat{\kappa}) \leq G(\zeta^*, \tau^*)$ which implies that $B_{LD} \leq L(\hat{\nu}, \hat{\delta}, \hat{\kappa}) \leq G(\zeta^*, \tau^*) = G_{LD}$.

Given any feasible (ζ, τ), we let $\delta_{ij} = \tau_{ij}$, for $1 \leq i \neq j \leq n$, and $\nu_{ij} + \nu_{ji} = \zeta_{ij}$, for $1 \leq i < j \leq n$. It is clear with the above assignments that

$$c_i + \sum_{j\neq i}\delta_{ij} - \sum_{j\neq i}(\nu_{ij}+\nu_{ji}) = c_i + \sum_{j\neq i}\tau_{ij} - \sum_{j<i}\zeta_{ji} - \sum_{j>i}\zeta_{ij}, \tag{101}$$

for $i = 1, \ldots, n$, and

$$\sum_{i=1}^{n}\sum_{j>i}(\nu_{ij}+\nu_{ji}) = \sum_{i=1}^{n}\sum_{j>i}\zeta_{ij}. \tag{102}$$

Let z^* be an optimal solution to (98) with a feasible (ζ, τ). Since binary variable z is unrestricted in (98), we know for each (i, j), $1 \leq i < j \leq n$, $z_{ij}^* = 1$ if $c_{ij} + \zeta_{ij} - \tau_{ij} - \tau_{ji} > 0$ and $z_{ij}^* = 0$, otherwise. We also let y^* be an optimal solution to (100).

For any (s, t), $1 \leq s < t \leq n$, we further write $c_{st} + \zeta_{st} - \tau_{st} - \tau_{ts} = (\frac{1}{2}c_{st} + \nu_{st} - \delta_{ts} + \kappa_{st}) + (\frac{1}{2}c_{ts} + \nu_{ts} - \delta_{st} - \kappa_{st})$. Here we use the fact that $c_{ij} = c_{ji}$ for all (i, j) with $1 \leq i < j \leq n$. We consider three cases on $c_{st} + \zeta_{st} - \tau_{st} - \tau_{ts}$ in (98).

Case I: $c_{st} + \zeta_{st} - \tau_{st} - \tau_{ts} > 0$. This condition implies that $z_{st}^* = 1$. In this case, we can find $\nu_{st}, \nu_{ts} \geq 0$ and $\kappa_{st} \in \mathbb{R}$ such that $\frac{1}{2}c_{st} + \nu_{st} - \delta_{ts} + \kappa_{st} > 0$ and $\frac{1}{2}c_{ts} + \nu_{ts} - \delta_{st} - \kappa_{st} > 0$. Hence, we conclude that $(y_t^s)^* = 1$ and $(y_s^t)^* = 1$, which implies that

$$\left(\frac{1}{2}c_{st} + \nu_{st} - \delta_{ts} + \kappa_{st}\right)(y_t^s)^* x_s^* + \left(\frac{1}{2}c_{ts} + \nu_{ts} - \delta_{st} - \kappa_{st}\right)(y_s^t)^* x_t^* \leq (c_{st} + \zeta_{st} - \tau_{st} - \tau_{ts}) z_{st}^*, \tag{103}$$

since x is a binary variable.

Case II: $c_{st} + \zeta_{st} - \tau_{st} - \tau_{ts} = 0$. Similar to Case I, it is clear that we can find $\nu_{st}, \nu_{ts} \geq 0$ and $\kappa_{st} \in \mathbb{R}$ such that $\frac{1}{2}c_{st} + \nu_{st} - \delta_{ts} + \kappa_{st} = 0$ and $\frac{1}{2}c_{ts} + \nu_{ts} - \delta_{st} - \kappa_{st} = 0$ and thus (103) holds trivially with both sides being 0.

Case III: $c_{st} + \zeta_{st} - \tau_{st} - \tau_{ts} \leq 0$, which implies $z_{st}^* = 0$. It is clear that we can find $\nu_{st}, \nu_{ts} \geq 0$ and $\kappa_{st} \in \mathbb{R}$ such that $\frac{1}{2}c_{st} + \nu_{st} - \delta_{ts} + \kappa_{st} \leq 0$ and $\frac{1}{2}c_{ts} + \nu_{ts} - \delta_{st} - \kappa_{st} \leq 0$. We conclude that $(y_t^s)^*(x_s)^* = 0$ and $(y_s^t)^*(x_t)^* = 0$. Hence, (103) follows.

We have proven for any feasible (ζ, τ), there exists some $(\hat{\nu}, \hat{\delta}, \hat{\kappa})$ such that (103) holds for any (i, j) with $1 \leq i < j \leq n$. Together with (101) and (102), we complete the proof. □

Remark 8 *As a matter of fact, only one of (92) and (93) is needed to ensure the results above. To see this, without loss of generality, we keep (92) and discard (93). This is equivalent to letting $\nu_{ij} = \zeta_{ij}$ and $\nu_{ji} = 0$ for all index pairs (i, j) and the above proof is still valid. Hence, we can strengthen Theorem 2 with $r = 5$ (i.e., with only four quadratic constraints for each index pair).*

Remark 9 *If we keep constraints (92)–(96) and add more constraints of form (49) into (P2), we can always improve the tightness of the resultant Lagrangian relaxation upper bound. For example, we may consider a case of $r = 8$ by adding (92)–(96) together with (84)–(85) into (P2), which yields an upper bound at least as tight as only relaxing (92)–(96) or (84)–(85). This indicates that Lagrangian relaxation on constraints (49) gives us flexibility in the upper bound improvement, which is potentially significant in a Lagrangian relaxation based branch-and-bound framework.*

When we add (92)–(96) to general constrained QBPs, it is typical that no fast solution exists to the MSAD (i.e., problem (74)) that is induced by relaxing (92)–(96). We thus consider solving the LP relaxation of MSAD. In the following, we prove that such an LP relaxation can still produce an upper bound at least as tight as B_{SL}.

Corollary 2 *We consider an MSAD described as above. Let B_{LD}^{LP} be the LP relaxation bound of this MSAD. Such an upper bound is at least as tight as the LP relaxation of (PSL), i.e., $B_{LD}^{LP} \leq B_{SL}$.*

Proof: Let G_{LD}^{LP} be the optimal objective value of the LP relaxation of (97). Relaxing the integrality restrictions on x in (97) yields an unconstrained problem with linear objective function. Thus, $G_{LD}^{LP} = B_{SL}$. To prove $B_{LD}^{LP} \leq G_{LD}^{LP}$, we can use the same Lagrangian multipliers as in the proof of Theorem 2. Hence, we conclude $B_{LD}^{LP} \leq G_{LD}^{LP} = B_{SL}$. □

3.4.3 Structure Preserving Decomposition

As we discuss in Section 3.2.2, in order to solve Lagrangian dual $L(\lambda)$ for given multipliers λ, i.e., problem (68), one can decompose the problem into n unconstrained linear binary problems with respect to only a subset of variables y (i.e., problem (72)–(73)) and one MSAD (i.e., problem (74)) with respect to only the original x variables. It is well known that an unconstrained linear binary program of size n can be solved in $O(n)$ time. Therefore, it takes $O(n^2)$ time to compute all g_i^* for all i in an MSAD. This implies that the computational complexity of solving $L(\lambda)$ as well as obtaining both upper and lower bounds on the original problem (P2) relies on how fast one can solve the corresponding MSAD.

It is easy to see that MSAD has the same set of constraints as (P2). Hence, we can directly apply efficient heuristics or exact solution methods that are available to those linear binary programming counterparts of (P2) that have the same set of constraints. To the best of our knowledge, such "structure preserving" feature is not inherent to existing general linearization techniques. In this section, we survey several classes of QBPs and illustrate how we can exploit their special constraint structure to improve the efficiency of solving MSADs. We also discuss a few ideas regarding how to cope with the computational intractability of general constrained MSADs.

MSAD with Exact Solution in Polynomial Time

Unconstrained QBP

Applications of unconstrained QBPs are numerous. For example, Laughunn [93] studied capital budgeting and investment portfolio selection. Chardaire and Sutter discussed several other applications of unconstrained QBPs in [38]. It is also worth noting that a large portion of QBP solution studies are focused on unconstrained QBPs.

Apparently, if (P2) of size n is unconstrained, then the respective MSAD is also unconstrained and can be solved in $O(n)$. Hence, solving (68) takes only $O(n^2)$ for unconstrained QBPs.

Dense k-Subgraph Problem (DkS)

In the dense k-subgraph maximization problem [55], we are given a graph G with n nodes, weight w_i associated with each node and weight w_{ij} associated with each edge between nodes i and j. For the parameter k, the DkS problem is to find a subgraph of size k which has the maximum total weight of the nodes and edges.

The DkS problem can be formulated as:

$$\max_x \left\{ \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \frac{1}{2} w_{ij} x_i x_j \right\} \tag{104}$$

$$\text{s.t.} \qquad \sum_i x_i = k; \tag{105}$$

$$x_i \in \{0,1\}, \text{ for } i = 1, \ldots, n,$$

where x_i is the variable indicating whether node i should be selected. In the case where only the weights of edges are considered, w_i can be simply set to zero for all i. The MSAD of a DkS problem is then derived as:

$$\max_x \left\{ \sum_{i=1}^{n} g_i^* x_i - \sum_{i=1}^{n} \sum_{j>i} \sum_{l=1}^{r_{ij}} \lambda_{ij}^l \epsilon_{ij}^l \;\middle|\; (105) - (106) \right\}. \tag{106}$$

The problem (106) can be solved by sorting all g_i^*, selecting the indices that correspond to k largest g_i^*, set the corresponding x_i to be 1, and set the other variables x_i to be 0. Since sorting can be done in $O(n \log n)$, the MSAD of a DkS problem can be solved in polynomial time.

Quadratic Semi-Assignment Problem (QSAP)

The quadratic semi-assignment problem [20] is to minimize a quadratic pseudo-Boolean function subject to the semi-assignment constraints. The problem is known to be NP-hard [20]. Many task-assignment problems in distributed systems can be easily formulated as QSAPs [19, 40, 146]. The problem also has important applications in a variety of other fields, e.g., bioinformatics [60].

The quadratic 0–1 formulation of QSAP is given as follows:

$$\max_x \left\{ \sum_{i=1}^{n} \sum_{k=1}^{m} c_{ik} x_{ik} + \sum_{i=1}^{n} \sum_{j\neq i} \sum_{k=1}^{m} \sum_{l\neq k} \frac{1}{2} c_{ijkl} x_{ik} x_{jl} \right\} \tag{107}$$

$$\text{s.t.} \qquad \sum_{k=1}^{m} x_{ik} = 1, \text{ for } i = 1, \ldots, n; \tag{108}$$

$$x_{ik} \in \{0,1\}, \text{ for } i = 1, \ldots, n, \; k = 1, \ldots, m. \tag{109}$$

The MSAD of a QSAP is then derived as:

$$\max_x \left\{ \sum_{i=1}^{n} \sum_{k=1}^{m} g_{ik}^* x_{ik} - \sum_{i=1}^{n} \sum_{j>i} \sum_{k=1}^{m} \sum_{l>k} \sum_{r=1}^{r_{ijkl}} \lambda_{ijkl}^r \epsilon_{ijkl}^r \;\middle|\; (108) - (109) \right\}. \tag{110}$$

An optimal solution to (110) can be obtained in $O(mn)$, as

$$x_{ik}^* = \begin{cases} 1, & k = \arg\max\{g_{il}^*, l = 1, \ldots, m\}; \\ 0, & \text{otherwise}, \end{cases} \tag{111}$$

for $i = 1, \ldots, n$. Hence, QSAP is another example whose corresponding MSAD can be solved in polynomial time. Since there are totally mn variables in this problem, the computational complexity for solving the MSAD for a quadratic semi-assignment problem is the same as for an unconstrained QBP.

Quadratic Assignment Problem (QAP)

QAP is one of the most important classes of constrained QBPs as it can be used to model a variety of real-world problems in facility allocation, parallel and distributed computing, combinatorial data analysis, among others. The surveys on the problem can be found in Burkard [28] and Pardalos et al. [114]. Besides being NP-hard, QAP is known to be computationally challenging even for rather small size instances [97]. QAP was first studied by Lawler [94] about half a century ago, and there is extended literature dedicated to obtaining bounds or suboptimal solutions to QAPs [9, 12, 13, 14].

Similar to QSAP, a QAP can be formulated as:

$$\max_x \left\{ \sum_{i=1}^{n}\sum_{k=1}^{n} c_{ik}x_{ik} + \sum_{i=1}^{n}\sum_{j\neq i}\sum_{k=1}^{n}\sum_{l\neq k} \frac{1}{2} c_{ijkl}x_{ik}x_{jl} \right\} \tag{112}$$

$$\text{s.t.} \quad \sum_{k=1}^{n} x_{ik} = 1, \text{ for } i = 1, \ldots, n; \tag{113}$$

$$\sum_{i=1}^{n} x_{ik} = 1, \text{ for } k = 1, \ldots, n; \tag{114}$$

$$x_{ik} \in \{0, 1\}, \text{ for } i = 1, \ldots, n, \; k = 1, \ldots, n. \tag{115}$$

The MSAD of the QAP is then derived as:

$$\max_x \left\{ \sum_{i=1}^{n}\sum_{k=1}^{n} g^*_{ik}x_{ik} - \sum_{i=1}^{n}\sum_{j>i}\sum_{k=1}^{n}\sum_{l>k}\sum_{r=1}^{r_{ijkl}} \lambda^r_{ijkl}\epsilon^r_{ijkl} \;\middle|\; (113) - (115) \right\}, \tag{116}$$

which has the property of total unimodularity and can thus be solved in polynomial time (e.g., with the Hungarian method in $O(n^3)$ time).

General Constrained MSAD

In general, an MSAD is an constrained binary linear program and no polynomial solution is available. In this section, we describe three approaches that deal with the constraints. We note that the performance of applying these three methods may vary depending on a particular application.

Solving the LP Relaxation of MSAD

A straightforward way to deal with a general constrained MSAD is to relax the integrality restrictions on x and solve the respective LP relaxation. It is easy to see that the LP relaxation provides an upper bound to (P2). A potential drawback of using this method is the inferiority of such a bound. In addition, no feasible solution to (P2) is guaranteed. A possible remedy is solving the LP relaxation of the MSAD iteratively to update the Lagrangian multipliers until the last iteration of a subgradient method and only dealing with the construction of a feasible solution at the last iteration. In our computational experiments on quadratic binary knapsack problem instances, we use this approach and apply a rounding heuristic to obtain a feasible solution at the last iteration.

Relaxing Original Constraints

Fisher [57] discussed the use of Lagrangian relaxation to solve integer programming problems in general. Given any λ, we can obtain an upper bound to (74) by further relaxing constraint (48) and solving the Lagrangian dual problem. The Lagrangian relaxation bound of MSAD is at least as tight as the LP relaxation bound [102]. In the actual computational implementation, one can combine the solutions of the two Lagrangian duals (relaxation of both quadratic constraints (49) and original constraints (48)) without destroying the decomposability. This idea can be beneficial in many problems, especially when there are not many original constraints, such as in a quadratic binary knapsack problem. It may be also beneficial to consider relaxing a subset of (48), as suggested by many studies in the IP literature.

Applying Existing Heuristics

Many binary linear programs (e.g., linear binary knapsack problem) have efficient heuristics to obtain good suboptimal solutions. With our decomposition method, these heuristics can be adapted to solve MSAD and provide feasible solutions. In addition, we note that surrogate subgradient methods (e.g., [150]) may be used here to update the Lagrangian multipliers effectively when only a suboptimal solution is obtained at each step.

3.4.4 Computational Considerations in the B&B Framework

It is natural to embed our Lagrangian relaxation bounding method into a branch-and-bound framework. In this section, we discuss two computational issues in our actual implementation of the branch-and-bound algorithm.

Subgradient Method

The subgradient method used to solve each Lagrangian relaxation problem is the most time-consuming part in the branch-and-bound algorithm. We employ two practically useful ideas in our actual implementation to alleviate this computational burden.

First, we set the maximum number of iterations in the subgradient optimization at each node and terminate each Lagrangian relaxation optimization once the number of iterations exceeds this threshold. We may choose different threshold values for different nodes in the tree. Intuitively, with this approach, spending more time at the beginning of a branch-and-bound solution procedure would likely lead to more promising solutions at the beginning of the procedure and prevent serious "tailing-off" effect at the end. In our actual implementation, we set this threshold to be 1000 at the root node and set it to be 5 at each of the other tree nodes. Our preliminary experiments indicate the benefit of using this specific setting.

Furthermore, unlike Caparara et al. [32] that used the same initial Lagrangian multipliers throughout the entire branch-and-bound tree, we find through our preliminary experiments that it is beneficial to "pass" Lagrangian multipliers from father nodes to children nodes along the tree. That is, when branching at a branch-and-bound tree node, we use its final iterative Lagrangian multipliers in the subgradient method to set the initial Lagrangian multipliers at its children nodes. This implies that we do not start the subgradient method with initial Lagrangian multipliers being zero at each tree node in our actual implementation. Intuitively, with this approach, we are more likely to use near-optimal multipliers from the beginning of each Lagrangian relaxation optimization.

Branching Rule

For a binary optimization problem, branching is equivalent to value fixation of the selected variable to 0 or 1. It is important to note that at each node of our branch-and-bound tree, we will select not only variable x_i to fix, but also all the corresponding variables y_i^j, $j = 1, \ldots, n, j \neq i$. Therefore, the maximum number of variables that need to be branched on is n.

Unlike standard linearization methods that lack a general guideline for branching, our Lagrangian relaxation method naturally lends itself to an efficient branching rule. For each x_i, let $w_i = \sum_{j>i} \sum_{l=1}^{r_{ij}} \lambda_{ij}^l + \sum_{j<i} \sum_{l=1}^{r_{ji}} \lambda_{ji}^l$. We then select the variable with the largest corresponding w value to branch at each node of the branch-and-bound tree, i.e., $i^* = \arg\max_{i=1}^n \{w_i\}$. In other words, we select the variable that offers the largest sum of the Lagrangian multipliers associated with it. Intuitively, this branching rule may identify the most "unstable" x variable. The level of referred "stability" indicates how much a variable x_i deviates from its associated variables y_i^j. To better understand this, take $r = 2$ for instance. In (83), for each (i, j) with $1 \leq i < j \leq n$, $(\lambda_{ij}^l)^k$, $l = 1$ or 2, increases at each iteration k only when the corresponding constraint (84) or (85) does not hold. Therefore, the larger the final Lagrangian multipliers are, the more their corresponding constraints are violated. A major contributor to the violation is the difference between each variable x_i and its associated y_i^j variables. Apparently, this argument applies in general cases.

3.4.5 Computational Experiments

We conducted computational experiments on randomly generated test instances of the unconstrained quadratic binary problem (UQBP) and the quadratic binary knapsack problem (QBKP). To show the superior performance of our Lagrangian relaxation based branch-and-bound method, we compared it to the direct CPLEX MIP solutions of the standard linearized reformulation (PSL) and an MIP reformulation that appears in Oral and Kettani [109], which contains only n auxiliary continuous variables and n additional constraints. This technique was originally presented in Glover [64] and Glover and Woolsey [67]. Some other variations are also discussed in [4]. However, to the best of our knowledge, the MIP reformulation presented in Oral and Kettani [109] introduces the fewest auxiliary variables and additional constraints among various reformulations due to this technique. We use the MIP reformulation in Oral and Kettani [109] to present the computational comparison in this chapter and term their reformulation as *OK reformulation*.

To construct reformulation (P2) for our method, we added to the original formulation (P0), two quadratic constraints (84) and (85) for each index pair. We set $\epsilon = -0.7$ in our experiments. To solve (P2), we implemented our branch and bound method in C with Python 2.6. With either comparative linearization technique, we solved the resultant reformulation using Cplex 10.1 with default settings as well. We primarily recorded the solution time. However, we set a CPU time limit of one hours for all three methods. When the time limit was reached, we reported the best suboptimal solution. All our computational experiments were conducted on a Linux 2.6.18 64bit machine with 16GB RAM and an Intel Xeon X5365 CPU of 3.0GHz.

Test Instance Generation

To design a test instance, we first generated the objective function coefficient matrix, which contains both c_{ij} and c_i. Note that we assumed $c_i = c_{ii}$ for each $i = 1, \ldots, n$. We used the method described by Pardalos [116]. The off-diagonal elements in the matrix were drawn uniformly from $[-50, 50]$. For diagonal elements, two parameter settings were considered. In the first setting, diagonal elements were drawn uniformly from $[-50, 50]$. In the second setting, the diagonal elements were drawn uniformly between $[0, 75]$. All the generated objective coefficient matrices were of full density. Note

that to solve instances with partially dense objective coefficient matrices, we only need to impose the quadratic constraints on the pairs of variables whose cross-term coefficients are nonzero. It helps improve the solution efficiency. However, it increases the implementation complexity. Hence, we leave its implementation to our future work.

For the QBKP instances, we generated the constraints in the following way. First, we randomly generated the constraint matrix using a similar method as the one used to generate the objective function coefficient matrix in [116]. Second, for each test instance, we randomly generated a feasible solution and multiplied it with the constraint matrix to form the nominal right-hand side of the constraints. In this way, we ensured that each test instance would have at least one feasible solution. Finally, we perturbed the right-hand side by adding some variance that was randomly generated from a uniform distribution $U(0, 20)$. In this way, we allowed multiple feasible solutions and thus guaranteed the non-triviality of each instance.

We considered three sizes of the problem ($n = 30, 40$, and 50) for each test problem (UQBP or QBKP). Given each problem size, we randomly generated 10 test instances with each parameter setting for generating the diagonal elements of the objective function coefficient matrix. In summary, we randomly generated 120 test instances in total.

Computational Results and Discussions

In the following two tables, we report the comparative running times (in seconds) with the three methods. Tables 2 and 3 present the results for the tested UQBP and QBKP instances, respectively. There are two portions in each table that correspond to each of the two parameter setting on the objective function coefficient matrix generation. Note that for each QBKP instance, we solved the LP relaxation of the resultant MSAD at each iteration in the subgradient method until the final iteration. We applied a standard rounding heuristic to obtain a feasible solution in the final iteration.

We use "LD", "PSL" and "OK" to represent the Lagrangian decomposition method introduced in this chapter, and the direct solutions of the PSL [5] and OK [109] reformulations, respectively. If the time limit is reached, we use "TO" to indicate it and report in the following parentheses the relative gap of the best feasible solution obtained by the studied algorithm to the optimal solution. For all of the test instances, our Lagrangian relaxation based method outperforms both direct solutions of the PSL and OK reformulations in terms of the solution time. Our method is on average two to three times faster than the direct CPLEX solution of PSL. This ratio seems to be insensitive to the size increase. We also conclude that our method can always find better feasible solutions compared to the standard linearization technique for those instances that cannot be solved to optimality within the time limit. Although the OK reformulation can obtain an equally good feasible solution for all of the test instances, it has to take a rather large amount of time for the method to discern the optimality.

3.5 Concluding Remarks and Future Work

In this line of research, we have proposed two dual decomposition schemes for stochastic QBPs and developed an innovative Lagrangian decomposition based method to solve deterministic QBPs. Our focus has been the latter one up to this point. The key of the Lagrangian decomposition based method is that we introduce parameterized quadratic constraints, which results in solving a series of binary linear programs to compute a Lagrangian relaxation bound. We provide a sufficient condition on the parameter specification for the introduced quadratic constraints. We also discuss several special cases of parameter specifications and their impacts on the bound tightness. We illustrate

Diagonal elements: $U(-50, 50)$									
Size	$n = 30$			$n = 40$			$n = 50$		
Inst. No.	LD	PSL	OK	LD	PSL	OK	LD	PSL	OK
1	0.74	2.07	4.09	76.13	244	TO(0%)	1418.39	3286.58	TO(0%)
2	0.71	2.60	6.78	10.74	37.31	164.86	1373.89	TO(1.46%)	TO(0%)
3	2.08	6.37	41.61	34.13	99.11	680.93	519.99	1672.28	TO(0%)
4	0.10	1.09	0.15	56.36	222.50	2041.93	811.17	1550.94	TO(0%)
5	0.39	1.42	1.75	67.09	318.13	3421.51	1204.44	2868.07	TO(0%)
6	0.77	2.14	4.24	22.38	41.18	342.88	1311.41	3548.89	TO(0%)
7	0.62	2.56	3.01	21.60	63.53	303.31	573.99	1449.92	TO(0%)
8	0.59	2.66	4.31	50.78	147.06	1811.44	984.99	2720.1	TO(0%)
9	0.72	2.24	3.19	66.61	244.12	3335.31	2236.05	TO(0%)	TO(0%)
10	0.56	1.73	1.55	23.44	66.43	485.69	1173.20	TO(0%)	TO(0%)
Diagonal elements: $U(0, 75)$									
Size	$n = 30$			$n = 40$			$n = 50$		
Inst. No.	LD	PSL	OK	LD	PSL	OK	LD	PSL	OK
1	0.35	1.21	1.02	10.15	27.37	145.59	1495.81	2978.02	TO(0%)
2	0.27	1.08	0.27	16.75	31.18	136.12	468.73	1616.57	TO(0%)
3	0.18	0.83	0.35	16.80	47.93	251.56	639.23	1275.13	TO(0%)
4	0.18	0.91	0.50	7.66	20.69	37.88	324.76	1148.55	TO(0%)
5	0.43	1.61	1.40	29.67	66.70	434.75	548.56	1704.43	TO(0%)
6	0.25	0.99	0.98	6.48	16.16	46.87	156.65	360.73	1433.89
7	0.25	1.06	0.35	22.48	79.36	538.45	1072.17	TO(2.08%)	TO(0%)
8	0.69	2.26	2.53	17.54	37.93	279.88	2254.32	TO(0.01%)	TO(0%)
9	0.08	0.68	0.06	6.88	14.43	17.08	362.04	433.83	TO(0%)
10	0.60	1.86	2.21	30.58	81.72	1469.03	639.57	1802.08	TO(0%)

Table 2: Comparative results on the UQBP instances (running times, in seconds)

Diagonal elements: $U(-50, 50)$									
Size	$n = 30$			$n = 40$			$n = 50$		
Inst. No.	LD	PSL	OK	LD	PSL	OK	LD	PSL	OK
1	0.33	1.17	0.68	33.97	69.48	791.43	1517.75	2982.94	TO(0%)
2	1.40	3.81	7.55	20.35	24.89	107.72	2686.68	TO(0.74%)	TO(0%)
3	1.35	3.15	15.74	38.59	125.41	997.92	459.77	1199.46	TO(0%)
4	1.39	3.53	16.23	39.18	73.95	978.38	1064.81	2813.01	TO(0%)
5	1.57	2.61	5.60	29.99	42.17	386.29	586.36	1245.19	TO(0%)
6	0.51	1.53	2.35	52.13	147.57	1645.26	1282.25	2838.76	TO(0%)
7	0.63	2.69	4.55	49.67	57.78	674.76	TO(0%)	TO(6.53%)	TO(0%)
8	0.73	2.10	3.39	18.97	43.66	239.95	TO(0%)	TO(3.00%)	TO(0%)
9	2.11	4.65	11.55	69.06	128.07	2405.16	2649.65	TO(6.01%)	TO(0%)
10	0.36	1.62	1.58	12.38	23.09	81.17	400.87	876.90	TO(0%)
Diagonal elements: $U(0, 75)$									
Size	$n = 30$			$n = 40$			$n = 50$		
Inst. No.	LD	PSL	OK	LD	PSL	OK	LD	PSL	OK
1	1.15	2.57	3.17	37.15	137.11	1781.01	2498.06	TO(1.28%)	TO(0%)
2	1.18	3.84	11.81	128.24	228.96	3147.89	1078.95	2585.18	TO(0%)
3	0.57	2.09	3.15	21.46	27.70	178.57	TO(0%)	TO(8.01%)	TO(0%)
4	0.31	1.10	0.65	90.10	226.75	1821.99	2208.05	3158.26	TO(0%)
5	5.57	10.22	72.21	96.19	117.34	1159.45	1299.29	3337.02	TO(0%)
6	0.36	1.32	0.83	42.71	75.95	726.00	2422.73	TO(0%)	TO(0%)
7	0.96	1.64	3.87	104.81	217.69	TO(0%)	3507.39	TO(2.46%)	TO(0%)
8	1.82	2.56	9.35	57.75	82.70	1232.01	273.90	486.37	TO(0%)
9	1.37	4.69	16.56	68.99	124.38	2573.05	921.49	TO(0%)	TO(0%)
10	1.68	3.51	5.50	49.87	81.94	800.43	TO(0%)	TO(3.42%)	TO(0%)

Table 3: Comparative results on the QBKP instances (running times, in seconds)

that our method does not change the underlying structure of the original QBP. Therefore, we have the potential to use the existing fast solution methods for the problem's IP counterpart. Computationally, we discuss several practically useful ideas, including parameter specifications, subgradient method for computing Lagrangian duals, and variable selection in the branch-and-bound algorithm. Our numerical experiments on the two classes of QBPs show that our method outperforms two well known linearization techniques. As for the development of the dual decomposition schemes for generic SQBPs, our aim is to seamlessly integrate dual decomposition and linearization. We are currently conducting computational experiments on larger instances to investigate the tradeoff between bound tightness and computational time on various proposed Lagrangian relaxation bounds.

For the Lagrangian decomposition based method, we propose several research items for further improvement of our method. First, in this work we use a specific parameter setting for the pair of quadratic constraints (84) and (85) in all of our computational experiments. In the future, it is worthwhile to investigate the impact of other parameter settings (e.g., the value of ϵ) for the case $r = 2$ both analytically and computationally. Furthermore, in this work, we present some preliminary attempts to understand the impact of $r > 2$. It is worthwhile to explore more general cases of using the proposed family of quadratic constraints. Analytically, we plan to evaluate the bounding performance with additional constraints in the reformulation. Computationally, we plan to investigate how each case affects the solution of the respective Lagrangian duals. Note also that in this work the subgradient method for computing the Lagrangian relaxation bound is the most time-consuming part in the branch-and-bound algorithm. Therefore, we need to investigate more advanced subgradient methods as well as their integration within the branch-and-bound algorithm. Finally, we plan to tune our algorithm for solving other classes of QBPs, e.g., quadratic assignment problems. *For the application of dual decomposition to solving SQBPs, our future work will be focused on the computational aspect. Once both parts of this research are more mature, we plan to incorporate the parametric quadratic constraints introduced in our deterministic QBP research into dual decomposition for SQBPs, our stochastic QBP research. Our ultimate goal is to develop a computationally attractive Lagrangian decomposition based branch-and-bound method for generic SQBPs.*

4 Two-Stage Stochastic Minimum $s-t$ Problem

This chapter is mostly based on the results from:

- S. Rebennack, O.A. Prokopyev, "Two-Stage Stochastic Minimum $s-t$ Problem: Formulations and Complexity," Technical Report, 2011.

4.1 Introduction

Let $G = (V, E)$ be a directed graph with node set V, arc set $E \subseteq V \times V$ and nonnegative costs c_{ij} given for each arc $ij \in E$. The *minimum $s-t$ cut problem* for directed graphs can be defined as follows [130]. For a given directed connected graph $G = (V, E)$ with root node s and terminal node t, the task is to find a node set $S \subset V$ with $s \in S$ and a node set $T \subset V$ with $t \in T$, such that $S \cup T = V$, $S \cap T = \emptyset$ and the *cost of the cut* $c[S,T] := \sum_{ij \in E: i \in S \wedge j \in T} c_{ij}$ is *minimized*.

Graph connectivity is one of the classical research topics in the graph theory, with a variety of practical applications, in particular, in network design [8]. The minimum $s-t$ cut problem and the maximum flow problem are dual problems to each other. This relation is called *Max-Flow Min-Cut Theorem* and was first proven by Ford and Fulkerson [58]. This duality enhanced the development of many polynomial time algorithms computing minimum $s-t$ cuts [8].

The following mathematical programming formulation of the minimum $s-t$ cut problem dates back to Ford and Fulkerson in 1962 [59]. Let

$$x_i = \begin{cases} 1, & \text{if } i \in T \\ 0, & \text{if } i \in S \end{cases} \qquad \text{and} \qquad y_{ij} = \begin{cases} 1, & \text{if } i \in S, j \in T \\ 0, & \text{otherwise} \end{cases} . \tag{117}$$

This allows the following linear 0-1 programming problem formulation:

$$\min \sum_{ij \in E} c_{ij} y_{ij} \tag{118}$$

$$s.t.\ y_{ij} \geq x_j - x_i \qquad \forall ij \in E \tag{119}$$

$$x_s = 0, \quad x_t = 1 \tag{120}$$

$$x_i, y_{ij} \in \{0,1\} \qquad \forall i \in V, ij \in E \tag{121}$$

Note that Ford and Fulkerson consider equation $x_t - x_s = 1$ instead of (120) in [59]. The constraint matrix defined by (119) - (120) is *totally unimodular* [59] (see further discussion in Sections 4.2.1 and 4.2.2), allowing the relaxation of the variables x and y to be non-negative, continuous and bounded above by 1. This provides another indication of the fact that the minimum $s-t$ cut problem is polynomially solvable.

An alternative definition of the minimum $s-t$ cut problem can be provided using the notion of the *cutset*. Define *cutset* as a set of arcs whose removal ensures that there is no no directed path from s to t. Then the minimum $s-t$ cut problem can be defined as the problem of finding a cutset of the minimum weight. Note that formulation (118)-(121) has a clear interpretation in this framework since variable y_{ij} is 1 if the corresponding arc $ij \in E$ belongs to the required cutset. Motivated by this fact, we introduce the following two-stage stochastic extension of the original deterministic problem.

Definition 1 (two-stage stochastic minimum $s-t$ cut) *Given is a directed graph $G = (V, E)$ with node set V and arc set E and a root $s \in V$. There are K scenarios. The kth scenario consists*

of a single terminal t^k and has probability p^k of being realized. Arc $ij \in E$ has cost c_{ij} in the first stage and d^k_{ij} in the recourse stage (or second stage) if the kth scenario is realized. The task is to find a set of arcs E_0 to be cut in the first stage and for each scenario k, an arc set E_k to be cut in the recourse stage if scenario k is realized, such that removing $E_0 \cup E_k$ from the graph G disconnects s from the terminal t^k. The objective is to minimize the expected cost over all scenarios:

$$z^{A*} := \min \left(\sum_{ij \in E_0} c_{ij} + \sum_{k=1}^{K} p^k \sum_{ij \in E_k} d^k_{ij} \right).$$

A number of studies demonstrates the potential benefits of stochastic programming solutions over deterministic approaches [133]. Stochastic programming models take advantage of the fact that probability distributions governing the data are known or can be estimated. Therefore, it is natural to consider stochastic extensions of classical graph and network design problems. The authors in [48, 70] discuss a somewhat related *robust $s-t$ min-cut problem*, where the task is to minimize the *maximum* cost over all scenarios while disconnecting s from terminals t^k. Other recent examples include two-stage stochastic extensions of maximum weight matching [90], shortest path [70], minimum spanning tree [49, 61] and Steiner tree [75] problems. For an introduction to stochastic programming, we refer the reader to Birge and Louveaux [23].

The remainder of this chapter is organized as follows. In Section 4.2, we provide a linear mixed 0–1 programming formulation of the two-stage stochastic minimum $s-t$ cut problem that is a generalization of the classical model (118)-(121). Unfortunately, the constraint matrix of the proposed mathematical program loses the total unimodularity property (Section 4.2.2) of the original deterministic formulation; however, this property is preserved if graph G is a tree (Section 4.2.3). This fact turns out to be not surprising as we prove in Section 4.3 that the considered problem is NP-hard, while a linear time solution algorithm is available when the graph is a tree (see Section 4.4). In Section 4.5, we discuss another variation of the two-stage stochastic minimum $s-t$ cut problem (referred to as the node-based version), that is motivated by an alternative formulation for the deterministic problem via a quadratic 0–1 program. Finally, Section 4.6 concludes the discussion.

We should also note that as a side result in Section 4.2.1 we derive a new characterization of totally unimodular matrices that generalizes some of the well-known results by Camion [30, 31]. This characterization is necessary for our discussion in Section 4.2.3.

4.2 Mathematical Programming Formulation

Let us discuss the two-stage stochastic minimum $s-t$ cut problem considering the graph in Figure 1. This graph has four nodes with node 1 as the root and node 4 as the terminal. Two scenarios are given with equal probabilities of 0.5.

In the two-stage stochastic minimum $s-t$ cut problem, one has to decide which *arcs* have to be cut in the first stage and in the second stage, where the cut in the second stage depends on the particular scenario of the second stage. An optimal solution using this arc-based interpretation is shown in Figure 2. In the first stage, both arcs (2,3) and (3,2) are cut. In the second stage, either arcs (1,2) and (3,4) are cut in case of scenario 1, or arcs (1,3) and (2,4) are cut in case of scenario two. This way, the optimal objective function value is 4.

Interestingly, in both scenarios, the resulting graph is disconnected after removing the arcs but the removal of the arcs does not correspond to a "minimum cut" in the classic sense, *i.e.*, as a partition of the nodes. This is the case, as always both arcs (2,3) and (3,2) are cut in the first stage. We interpret this solution as ***hedging*** of arcs. However, note that the resulting arcs define a valid cutset.

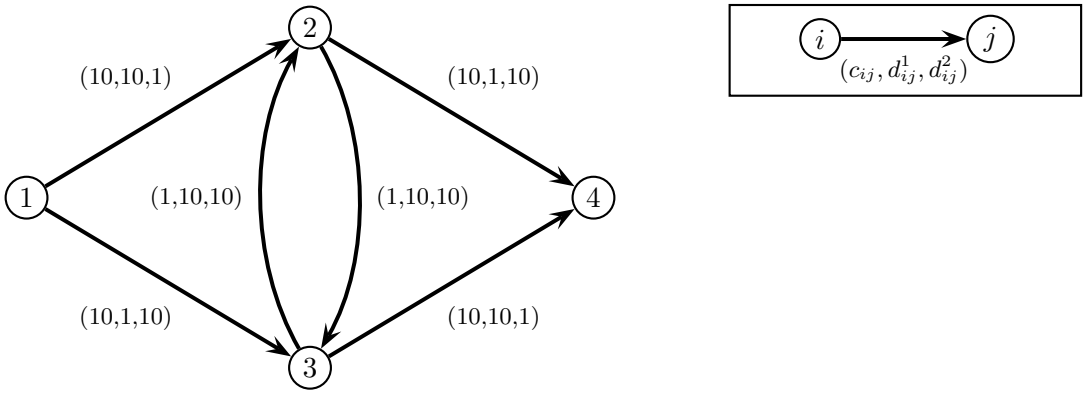


Figure 1: A two-stage minimum $s-t$ cut instance with node 1 as the root node and node 4 as the destination node. Given are two scenarios with equal probability.

On the other hand, let us consider each scenario independently after the arcs that are cut in the first stage are removed from G. Then we can observe that the resulting subproblem for each scenario is a classical minimum $s-t$ cut problem that can be equivalently interpreted either as a partition of the nodes or a cutset of arcs.

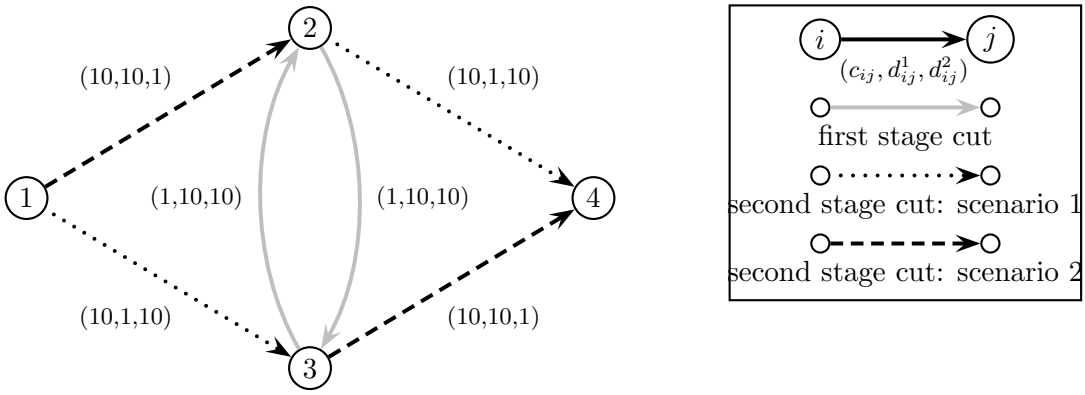


Figure 2: Hedging: in the first stage, both arc (2,3) and (3,2) are cut; all arcs are cut in this graph. The cost of this minimum cut is 4.

Let us now discuss two cases when specific structure on the arc cost are present.

Case 1: $c_{ij} \leq p^k d^k_{ij}, \quad \forall ij \in E,\, k = 1, \ldots, K$ The first stage cutting cost for each arc is less than or equal to the cost of cutting in each of the K scenarios in the second stage, weighted with the scenario probability. Thus, there is no need to cut in the second stage. Now, the problem transforms into a cut problem, where K terminals have to be cut from a single source s. This problem can be transformed into a (deterministic) minimum $s-t$ cut problem by introducing a super terminal node t which is connected to each of the k terminals with arc cost $+\infty$.

Case 2: $c_{ij} \geq d^k_{ij}, \quad \forall ij \in E,\, k = 1, \ldots, K$ The first stage arc cost is greater than or equal to the cost of cutting in the second stage at any of the scenarios. In this case, no arcs need to be cut in the first stage and the second stage problem decomposes into K independent (deterministic) minimum $s-t$ cut problems.

Thus, in both cases, the two-stage stochastic minimum $s-t$ cut problem is solvable in polynomial time. This is summarized in the following Corollary.

Corollary 1 *The two-stage stochastic minimum $s-t$ cut problem is solvable in strongly polynomial time for arbitrary graphs if one of the two cases holds true:*

$$c_{ij} \leq \min_{k=1,\ldots,K} \{p^k d_{ij}^k\} \qquad \forall ij \in E \quad ,$$

or

$$c_{ij} \geq \max_{k=1,\ldots,K} \{d_{ij}^k\} \qquad \forall ij \in E \quad .$$

Allowing "hedging" of arcs, based on (118)-(121), we obtain the following formulation for the two-stage stochastic minimum $s-t$ cut problem

$$\min \sum_{ij \in E} c_{ij} y_{ij} + \sum_{k=1}^{K} p_k \sum_{ij \in E} d_{ij}^k u_{ij}^k \tag{122}$$

$$s.t.\ u_{ij}^k + y_{ij} \geq x_j^k - x_i^k, \qquad \forall ij \in E, \quad k = 1, \ldots, K; \tag{123}$$

$$x_s^k = 0, \qquad k = 1, \ldots, K; \tag{124}$$

$$x_{t^k}^k = 1, \qquad k = 1, \ldots, K; \tag{125}$$

$$x_i^k, y_{ij}, u_{ij}^k \in \{0,1\} \qquad \forall i \in V, \quad \forall ij \in E, \quad k = 1, \ldots, K, \tag{126}$$

where we define variables u_{ij}^k similar to (117) as the arc to be cut for scenario k and variable x_i^k has value one if node $i \in V$ belongs to set T^k, otherwise it has value 0 and belongs to set S^k.

Proposition 1 *Formulation (122) - (126) models the two-stage stochastic minimum $s-t$ cut problem correctly.*

Proof: Let arc sets E_0 and E_k define a two-stage $s-t$ cut for graph G. Arc set E_k defines sets S^k and T^k for each scenario k. With this, assign variables x^k values 0 or 1 accordingly. The cut variables y and u^k obtain their values according to the arc sets E_0 and E_k. With this assignment, equation (123) is satisfied because otherwise, the sets S^k and T^k do not define a cut. The objective function value of this cut is calculated correctly.

It remains to show that any optimal solution of (122) - (126) defines a two-stage $s-t$ cut for graph G. Therefore, assume that we are given an optimal solution of (122) - (126) and that the graph is not disconnected. Hence, for at least one scenario k, there is a (directed) path from root s to terminal t^k. Equations (123) imply that all variables x_i^k for nodes i along this path have the same value, which contradicts equations (124) and (125). □

The y_{ij} variables in equations (123) represent the cuts in the first stage. As such, the y_{ij} variables connect the K stages. Thus, if variables y_{ij} are fixed to 0 or 1, then problem (122) - (126) decomposed into K separate minimum $s-t$ cut problems of type (118) - (121). Thus, each of the K optimization problems is a linear program. This structure suggests that we could develop of a solution algorithm based on the Benders Decomposition approach [101], where the master problem contains the y_{ij} variables and the K sub-problems are minimum $s-t$ cut problems for trial values $\bar{y}_{ij}$ obtained from the master problem.

Recognize that formulation (122) - (126) does not involve any variables x_i for the first stage, but only for the recourse stage. Furthermore, variables y and u^k can be relaxed to be non-negative continuous if variables x^k are binary. In order to see this, consider an optimal, fractional solution

for variables y and u^k, as well as the corresponding binary solution values of variables x_i^k and x_j^k, and assume that all $d_{ij}^k > 0$. This implies that there must be an arc $ij \in E$ with $y_{ij} = l \in (0,1)$; otherwise, each fractional variable u_{ij}^k implies that $d_{ij}^k = 0$. Then, equations (123) imply that either $u_{ij}^k = 1 - l$ if $x_j^k - x_i^k = 1$ or $u_{ij}^k = 0$ in all other cases. Considering all scenarios k, arc $ij \in E$ contributes to the objective function the value $c_{ij} \cdot l + \sum_{k=1}^{K} p_k d_{ij}^k u_{ij}^k = l \cdot c_{ij} + (1-l) \cdot \sum_{k:k\in K \wedge u_{ij}^k \neq 0} p_k d_{ij}^k$, which is a convex combination of the two values c_{ij} and $\sum_{k:k\in K \wedge u_{ij}^k \neq 0} p_k d_{ij}^k$. As the solution defined by variables y, u^k and x^k is optimal, we obtain that $c_{ij} = \sum_{k:k\in K \wedge u_{ij}^k \neq 0} p_k d_{ij}^k$. Hence, the extreme point solution $y_{ij} = 1$ and $u_{ij}^k = 0$ is also an optimal solution.

It is natural to consider whether variables x^k can be relaxed as well, similar to the deterministic case. This leads us to the discussion of whether constraint matrix (123) - (125) is totally unimodular. We show that the constraint matrix of the two-stage stochastic minimum $s-t$ cut problem loses its property of being totally unimodular when extended from the deterministic case.

4.2.1 Total Unimodularity

In this section, we review properties of *totally unimodular* (TU) matrices. Furthermore, we also derive a new characterization of TU matrices that generalizes some of the well-known results by Camion [30, 31]. This characterization is necessary for our further discussion in Section 4.2.3.

A matrix $\mathbb{A}$ is *totally unimodular* (TU), if the determinant of each square submatrix of $\mathbb{A}$ has the value 0, 1, or -1. Recognize that a totally unimodular matrix does not need to be square itself. From the definition it follows that any totally unimodular matrix has only $\{\pm 1, 0\}$ entries. Therefore, in the remainder of this section we assume that $\mathbb{A}$ always denotes a matrix with $\{\pm 1, 0\}$ entries.

The next theorem shows the importance of totally unimodular matrices for integer programming.

Theorem 1 ([149]) *Matrix $\mathbb{A}$ is totally unimodular, if and only if of for each integral vector b, set $\{x \in \mathbb{R}^n : \mathbb{A}x \leq b\}$ is an integer polyhedron.*

We state two sufficient and necessary conditions for a matrix to be totally unimodular, using a specific matrix called an *Eulerian* matrix.

Definition 2 ([18]) *A matrix $\mathbb{A}$ is said to be Eulerian, if*

$$\sum_i a_{ij} \equiv 0 \bmod 2 \qquad \forall j \quad , \tag{127}$$

and

$$\sum_j a_{ij} \equiv 0 \bmod 2 \qquad \forall i \quad . \tag{128}$$

This enables us to state the following two theorems characterizing totally unimodular matrices.

Theorem 2 ([30]) *Matrix $\mathbb{A}$ is totally unimodular, if and only if every square Eulerian submatrix is singular.*

Theorem 3 ([31]) *Matrix $\mathbb{A}$ is totally unimodular, if and only if every square Eulerian submatrix $\mathbb{E}$ satisfies:*

$$\sum_{i,j} e_{ij} \equiv 0 \bmod 4 \quad .$$

In the next result, we combine Theorems 2 and 3. The difference to each theorem is that it suffices to check one of the two criteria for Eulerian submatrices. Thus, one can "mix" the criterion as needed.

Theorem 4 *Let $\mathcal{E}$ be the set of all square Eulerian submatrices of matrix $\mathbb{A}$. Matrix $\mathbb{A}$ is totally unimodular, if and only if*

$$\forall \mathbb{E} \in \mathcal{E} : \mathbb{E} \text{ is singular or satisfies } \sum_{i,j} e_{ij} \equiv 0 \bmod 4 \quad . \tag{129}$$

Proof: "⇒" This follows directly from Theorems 2 and 3.

"⇐" The proof of this direction is along the lines of *cf.* [31, Theorem 2]. We need the following results from [31] and [144]:

Statement 1 (due to R. Gomory) [31]. Let E be a square Eulerian submatrix of a matrix $\mathbb{A}$, such that every proper submatrix of E is totally unimodular, then $\sum_{i,j} e_{ij} \equiv det(E) \bmod 4$.

Statement 2 (due to R. Gomory) [144]. If $\mathbb{A}$ is not TU, then $\mathbb{A}$ has a submatrix of determinant ± 2.

Statement 3 (due to R. Gomory) [31]. If for every square Eulerian submatrix E of $\mathbb{A}$, $det(E) = 0$, then, for every square submatrix B of $\mathbb{A}$, $det(B) \equiv 0 \bmod 2$ implies $det(B) = 0$.

We prove the necessary result by contradiction. Let $\mathbb{A}$ be not TU. By Theorem 2, there exists a square Eulerian submatrix E of $\mathbb{A}$ which is not singular, i.e., $det(E) \neq 0$. Note that E is not TU. However, by (129) we know that this matrix satisfies $\sum_{i,j} e_{ij} \equiv 0 \bmod 4$.

Without loss of generality assume that E is the smallest square Eulerian submatrix of $\mathbb{A}$ such that $det(E) \neq 0$. In other words, every proper square Eulerian submatrix $\tilde{E}$ of E is singular, i.e., $det(\tilde{E}) = 0$. Then by Theorem 2, every proper square Eulerian submatrix $\tilde{E}$ of E is TU, which, due to Statement 1, implies that $\sum_{i,j} e_{ij} \equiv det(E) \bmod 4$. Recall that $\sum_{i,j} e_{ij} \equiv 0 \bmod 4$. Thus, $det(E) \equiv 0 \bmod 4$. Then $det(E) \geq 4$ since $det(E) \neq 0$.

Note that E is not TU. Then by Statement 2 there exists submatrix B of E such that $|det(B)| = 2$. Observe that B is a proper submatrix of E; therefore, every square Eulerian submatrix $\bar{E}$ of B satisfies $det(\bar{E}) = 0$. Therefore, Statement 3 can be applied implying that $det(B) = 0$, which results in a contradiction. □

Denote by $\mathbb{I}$ an identity matrix with an appropriate dimension. $\mathbb{I}|\mathbb{A}$ denotes "gluing" matrix $\mathbb{A}$ to the right of matrix $\mathbb{I}$. We also use the following two results.

Lemma 1 ([149]) *Matrix $\mathbb{I}|\mathbb{A}$ is totally unimodular, if and only if $\mathbb{A}$ is totally unimodular.*

Lemma 2 *Let $\mathbb{A}$ be a matrix and $\mathbb{B}$ be the matrix where one row with exactly one entry with value +1 or -1 is added to $\mathbb{A}$. Then, matrix $\mathbb{B}$ is totally unimodular, if and only if $\mathbb{A}$ is totally unimodular.*

Proof: Use the criterion in [1] and observe that J^1 and J^2 is a partition of the columns for matrix $\mathbb{A}$ with $\left|\sum_{j \in J^1} a_{ij} - \sum_{j \in J^2} a_{ij}\right| \leq 1$ for each row i, if and only if it is for $\mathbb{B}$. □

4.2.2 General Case: Total Unimodularity is Lost

In order to discuss totally unimodularity of the constraint matrix (123) - (125), it suffices to consider the matrix defined by (123), due to Lemma 2. Let us re-write the constraints in the form $A\tilde{x} \leq b$. Therefore, let y be the vector of variables y_{ij}, u^k be the vector of variables u^k_{ij} and x^k be the vector of variables x^k_{ij}. Furthermore, let $\mathbb{B}$ be the arc-node incidence matrix of graph G with dimension

$(m \times n)$, let $\mathbb{I}$ be the identity matrix (with appropriate dimension) and $\mathbf{0}$ be the matrix consistent of entries all 0 (with appropriate dimension). Finally, let $\tilde{x} = \left(u^1, \ldots, u^K, y, x^1, \ldots, x^K\right)^T$. This enables us to re-write (123) as

$$Ax^\top = \left(\begin{array}{cccc|c|cccc} -\mathbb{I} & \mathbf{0} & \cdots & \mathbf{0} & -\mathbb{I} & \mathbb{B} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & -\mathbb{I} & \vdots & \mathbf{0} & -\mathbb{I} & \mathbf{0} & \mathbb{B} & \vdots & \mathbf{0} \\ \vdots & \cdots & \ddots & \vdots & \vdots & \vdots & \cdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & -\mathbb{I} & -\mathbb{I} & \mathbf{0} & \mathbf{0} & \cdots & \mathbb{B} \end{array}\right) \left(\begin{array}{c} u^1 \\ u^2 \\ \vdots \\ u^K \\ \hline y \\ \hline x^1 \\ x^2 \\ \vdots \\ x^K \end{array}\right) \leq 0 \quad . \tag{130}$$

With Lemma 1, it suffices to consider matrix

$$\mathbb{C} = \left(\begin{array}{c|ccc} -\mathbb{I} & \mathbb{B} & \mathbf{0} & \cdots & \mathbf{0} \\ -\mathbb{I} & \mathbf{0} & \mathbb{B} & \vdots & \mathbf{0} \\ \vdots & \vdots & \cdots & \ddots & \vdots \\ -\mathbb{I} & \mathbf{0} & \mathbf{0} & \cdots & \mathbb{B} \end{array}\right) \quad . \tag{131}$$

Consider now the graph with four nodes and four arcs shown in Figure 3. Recognize that the graph does not contain a directed cycle (but is not a tree).

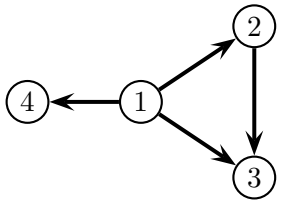


Figure 3: The corresponding matrix defined through constraints (123) - (125) is not totally unimodular for $K \geq 2$.

For $K = 2$ scenarios, matrix $\mathbb{C}$ is given as

$$\mathbb{C} = \left(\begin{array}{cccc|cccc|cccc} (1,4) & (1,2) & (1,3) & (2,3) & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 \\ -1 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \end{array}\right) , \tag{132}$$

where the first four columns correspond to the arc variables y_{ij}^k, columns 5 to 8 to the node variables x_i^1 for scenario 1 and the last four columns to variables x_i^2 for scenario two, respectively. Matrix $\mathbb{C}$

in (132) is not totally unimodular as the square sub-matrix

$$\mathbb{E} = \begin{array}{c} \begin{array}{cccccc} (1,4) & (2,3) & 1 & 2 & 1 & 3 \end{array} \\ \left(\begin{array}{cc|cc|cc} -1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & 0 \\ \hline -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & -1 & 0 & 0 & 0 & 1 \end{array}\right) \end{array} \tag{133}$$

has determinant -2. Recognize that the square sub-matrix $\mathbb{E}$ is also Eulerian but neither satisfies the criteria of Theorem 2 nor of Theorem 3.

4.2.3 Trees: Total Unimodularity is Preserved

In this section, we consider the special case that graph G is a tree, *i.e.*, the graph does not contain any (undirected) circles. We provide two proofs that the constraint matrix of (123) - (125) is totally unimodular in the case of trees.

The first proof described in Section 4.2.3 defines so-called *walks* in matrices to show that the criteria of Theorem 4 is satisfied. We gain deep insights of the structure of matrix $\mathbb{C}$ and we can identify the elements of the tree needed for the TU property of Theorem 4.

In Section 4.2.3, we present an alternative proof that the constraint matrix of (123) - (125) is TU. The proof relies on regular matroids and their transformations. As a by-product, we learn that the two-stage stochastic minimum $s-t$ cut problems for trees are single commodity flow problems on a transformed graph. Furthermore, in Section 4.4, we discuss a linear time algorithm for the two-stage stochastic minimum $s-t$ cut problem. All this implies that the two-stage stochastic minimum $s-t$ cut problem is polynomially solvable for trees – just as the demand robust minimum $s-t$ cut problem is [70].

In the following, we consider only (directed) rooted-out trees, that is, a directed graph where each node except the root node has indegree 1. The more general case considering trees where the direction of the arcs does not matter is not of interest in the context of two-stage stochastic min-cut problems. The reason is that not strongly connected trees lead to a 0-cost cut (in case that a terminal node is not strongly connected to the root node). Recall that a directed graph is strongly connected if for each pair of nodes, there is a directed path connecting them. If not mentioned otherwise, we mean by a tree a (directed) rooted-out tree.

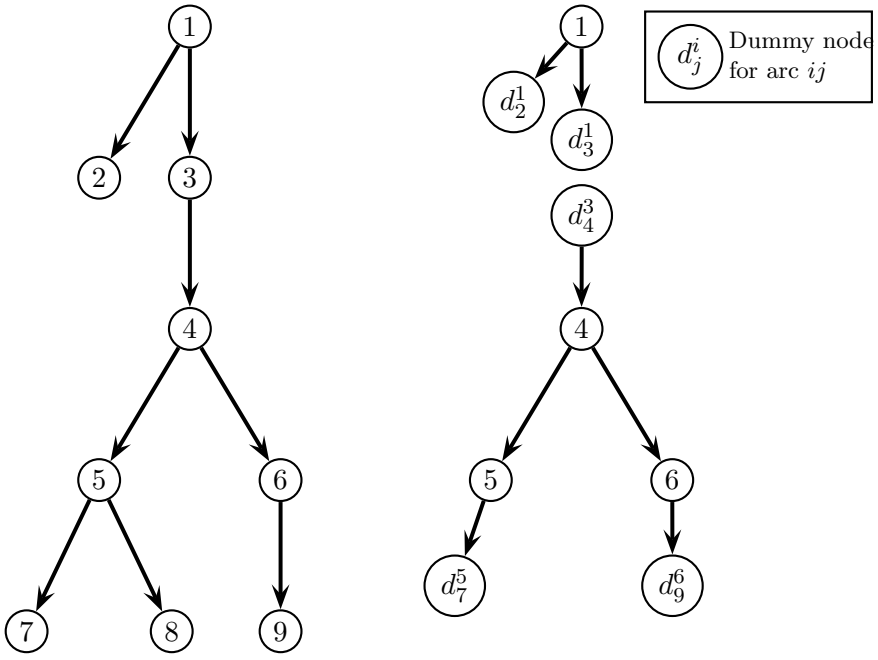
We have already seen with Figure 3 that, in general, the constraint matrix (123) - (125) is not totally unimodular. The underlying reason is that the graph contains an undirected loop. Forbidding undirected loops for graphs leads to trees – and this property of a tree is exactly what we need in order to prove that the corresponding constraint matrix is totally unimodular.

Proof via Walks

We start with the following lemma, which is true because in a tree each node has at most one predecessor.

Lemma 3 *If the graph G is a tree, then any column of the constraint matrix defined by (123) has at most one entry with value $+1$.*

In this section, we denote by $\mathbb{E}$ an Eulerian (square) sub-matrix of the matrix $\mathbb{C}$ defined in (131). Let us introduce the following notation: Let J be the set of columns of matrix $\mathbb{C}$, $\bar{J} \subset J$ be the



(a) Graph G (b) Forest T^{D}

Figure 4: Graph G and the corresponding forest T^{D} for the Eulerian matrix $\mathbb{E}$ defined in (134)

columns of J corresponding to variable y and let $J^k \subset J$ be the columns of J corresponding to variable x^k.

Consider now the tree G given in Figure 4 (a) and assume that the number of scenarios is greater than or equal than 3. One square Eulerian sub-matrix $\mathbb{E}$ of the corresponding constraint matrix is given by

$$\mathbb{E} = \left(\begin{array}{ccccc|ccc|ccc|ccc}
 & & \bar{J} & & & & J^1 & & & J^2 & & & J^3 & \\
(1,2) & (1,3) & (3,4) & (5,7) & (6,9) & 1 & 4 & 6 & 4 & 5 & 6 & 1 & 4 & 5 \\
-1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
\hline
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1
\end{array}\right) \tag{134}$$

We can easily check that the square sub-matrix $\mathbb{E}$ of $\mathbb{C}$ is Eulerian. In order to more easily recognize the similarity of $\mathbb{E}$ to the original constraint matrix, we include the corresponding column names. The first five columns of $\mathbb{E}$ are from set $\bar{J}$, corresponding to the arc variables y. The rest of the columns correspond to one of the scenarios $k = 1, 2, 3$ (with $K \geq 3$) and to the node variables x_i^k.

Let $\mathbb{E}$ be an Eulerian sub-matrix of $\mathbb{C}$ corresponding to directed graph G and consider the case in which $\bar{J} \neq \emptyset$. Let $\bar{j}$ be a column of $\bar{J}$ with a -1 entry for a scenario k. Then, the corresponding row of this -1 entry is unique and is denoted by i_1^k. In the following discussion, we omit the superscript k and write i_1 instead. As $\mathbb{E}$ is Eulerian, each row sum has value 0 mod 2. Applying this logic to row i_1 implies that there is exactly one column $j_1 \in J^k$ which has entry ± 1 in row i^1. Similarly, each column sum has to equal 0 mod 2. This ensures that there is an odd number of entries ± 1 in column j_1 other than the entry ± 1 in row i_1. Selecting one of these values and applying the same argument for the new row, we either obtain a -1 entry in a column of $\bar{J}$ or a ± 1 entry in one of the other columns of J^k. Applying this argument for the new value will lead to a new row selection. This can be repeated until, eventually, we will end up at a -1 entry of a column in $\bar{J}$ (as each row sum is 0 mod 2). Let us call this process of proceeding from one column $\bar{j}_1$ of $\bar{J}$ to another column $\bar{j}_2$ of $\bar{J}$ a "**walk**" in matrix $\mathbb{E}$ from column $\bar{j}_1$ to $\bar{j}_2$. Note that by construction $\bar{j}_1$ and $\bar{j}_2$ belong to the row block of the same scenario.

With this notation, we can state the next lemma which holds true for any directed graph G.

Lemma 4 *Let G be a graph and $\mathbb{E}$ be an Eulerian sub-matrix of $\mathbb{C}$. Then, for any k, $\mathbb{E}$ has an even number of -1 entries in columns of $\bar{J}$ belonging to the row block of scenario k.*

Proof: If $\bar{J} = \emptyset$, there is nothing to show. Hence, assume $\bar{J} \neq \emptyset$ and consider a walk from column $\bar{j}_1$ to $\bar{j}_2$. For this walk, mark all the ± 1 entries contained in this walk. According to construction of the walk, in each row and each column, we mark either exactly two entries or none at all; that is, an *even* number of entries.

For that same scenario k, pick any column $\bar{j} \in \bar{J}$ with an un-marked -1 entry. If there is none, then we are done with this scenario. Now, assume that there is such an entry.

Construct a "walk" from this column j to any other, not fixed, column in $\bar{J}$. Assume that we cannot find any such walk. This means that we arrive at a row or column in which all non-zero entries have been marked. However, this is a contradiction as this would imply that there is an odd number of non-zero entries in this particular row or column (as we marked an even number of entries for each row and column).

Finally, we can apply this analysis to each scenario k separately, until all -1 entries in columns of $\bar{J}$ have been marked. □

Recognize that if there is a walk from a column $\bar{j}_1 \in \bar{J}$ to $\bar{j}_2 \in \bar{J}$ for any particular scenario k, then this walk is unique, as G is a tree. However, starting at a particular column $\bar{j} \in \bar{J}$, we can arrive at different columns of $\bar{J}$. This is due to (possible) multiple choices for selecting a value in a column (for a fixed row).

Furthermore, recognize that the number of columns $\bar{J}$ of matrix $\mathbb{E}$ can be odd; see for instance the Eulerian matrix $\mathbb{E}$ defined in (134). Notice that Lemma 4 does not require the Eulerian matrix to be square.

With the help of Lemma 4, we are able to prove the following results, which seem to be surprising at the first glance.

Lemma 5 *Let G be a graph and $\mathbb{E}$ be an Eulerian sub-matrix of $\mathbb{C}$. Then, if $\bar{J} \neq \emptyset$ and none of the columns and rows of $\mathbb{E}$ is the 0 vector, then matrix $\mathbb{E}$ is square, if and only if the sum of each element in column $j \in \bar{J}$ has value -2; i.e., in each column $j \in \bar{J}$, there are exactly two non-zero entries (both having value -1).*

Proof: Let us first prove direction "$\Leftarrow$". Therefore, consider a walk in $\mathbb{E}$ from a column $\bar{j}_1 \in \bar{J}$ to $\bar{j}_2 \in \bar{J}$ and "mark" the rows and columns used in this walk. Assume that both $\bar{j}_1$ and $\bar{j}_2$ are not "marked" initially. Then it is easy to observe that the number of rows covered by this walk is $2+\ell$, with ℓ being non-negative and integral. Then, the number of columns for this walk is 2 (for the columns in $\bar{J}$) + 1 + ℓ. Hence, the number of "marked" columns is one more than the number of "marked" rows.

Now, consider another walk in $\mathbb{E}$ from "marked" column $\bar{j}_1 \in \bar{J}$ (but not "marked" row) to one arbitrary column $\bar{j} \in \bar{J}$. In the case that $\bar{j} = \bar{j}_2$, we "mark" in a new walk additionally $2+m$ new rows but only $1+m$ columns (as the columns in $\bar{J}$ have been already "marked") for some non-negative and integer m. Hence, the number of "marked" columns is one less(!) than the number of "marked" rows. In the other case in which $\bar{j} \neq \bar{j}_2$ (i.e., the final column has not been "marked" yet), we cover with "marks" additionally $2+m$ rows and $2+m$ columns.

We can construct a set of walks for each scenario which are row distinct (through the argument with the "marks") and cover all rows of matrix $\mathbb{E}$ (recognize that $\mathbb{E}$ contains no row or column having only 0 entries). Simply speaking every iteration of the "marking" procedure (by iteration we imply construction of a new walk) has three possible outcomes: (i) the number of additional "marked" columns and rows is the same, i.e., exactly one column from $\bar{J}$ is encountered (and "marked") for the first time in a new walk; (ii) the number of additional "marked" columns is one less than the number of new "marked" rows, i.e., no new columns from $\bar{J}$ are "marked"; (iii) the number of additional "marked" columns is one more than the number of new "marked" rows, i.e., exactly two new columns from $\bar{J}$ are "marked."

Let x_1, x_2 and x_3 be the numbers of times outcomes (i), (ii) and (iii) occurred, respectively. During the procedure the number of new columns from $\bar{J}$ encountered is $x_1 + 2x_3$; the number of "marked" columns from $\bar{J}$ encountered is $x_1 + 2x_2$. Since each column of $\bar{J}$ has exactly two -1 entries in $\mathbb{E}$, then each column is encountered exactly twice: once as a new one and the other time as a "marked" one. Therefore, $x_1 + 2x_2 = x_1 + 2x_3$ and $x_2 = x_3$. It immediately implies that for every outcome (ii) that increases the difference between the numbers of "marked" rows and columns by one, there exists exactly one outcome (iii) that decreases this difference by one. Therefore, at the end of the procedure the number of "marked" columns must be equal to the number of "marked" rows and matrix $\mathbb{E}$ has to be square.

For the other direction "$\Rightarrow$", assume that there exists a column in $\bar{J}$ that has more that two -1 entries in $\mathbb{E}$. With the calculations above, it is easy to observe that during the procedure the number of "marked" columns from $\bar{J}$ encountered should be greater than the number of new columns from $\bar{J}$ encountered, i.e., $x_1 + 2x_2 > x_1 + 2x_3$. Therefore, $x_2 > x_3$ that implies that at the end of the procedure the number of "marked" columns must be less than the number of "marked" rows and matrix $\mathbb{E}$ is not square. □

Now, let us define a special tree structure resulting from an Eulerian sub-matrix $\mathbb{E}$ of $\mathbb{C}$ (131). For the tree construction, we assume that $\bar{J} \neq \emptyset$. The tree T^{D} is then constructed via procedure CT; see below.

Applying Procedure CT to the graph G of Figure 4 (a) for the constraint matrix (134) leads to the forest T^{D} shown in Figure 4 (b). Recognize that graph T^{D} is disconnected and that all leaf nodes are dummy nodes. Furthermore, any dummy node is either a leaf node or the root node of a particular tree in the forest T^{D}. Both observations hold true in general; however, we do not provide a proof here as we do not require this property in the following discussion, even though the proof follows immediately from Lemma 6 together with Lemma 5. Notice that not all nodes of the original graph are included in the forest T^{D}.

The following lemma summarizes the properties of graph T^{D}.

Procedure Construct Graph T^{D} (CT)

1. For each row i^k of matrix $\mathbb{E}$, one of the following cases (for the columns in J^k) holds:

   1.1 if there is a -1 and a +1 entry in i^k other than in columns of $\bar{J}$:
   include the corresponding arc in T^{D} (along with the two end nodes)

   1.2 if there is only a +1 entry in i^k:
   the corresponding column with entry +1 corresponds to a node j in the original graph G. Let i be the predecessor of node j in graph G. Then, include the dummy node d^i_j and node i in T^{D} along with the arc (d^i_j, j)

   1.3 if there is a -1 entry but not a +1 entry in row i^k:
   the corresponding column with entry -1 corresponds to a node i in the original graph G. Let i be the predecessor of node j in graph G. Then, include the dummy node d^i_j and node i in T^{D} along with the arc (i, d^i_j)

   1.4 if there are no non-zero entries in i^k:
   ignore this row

End procedure

Lemma 6 *Graph T^{D} has the following properties:*

1. *Each node in the graph is either a dummy node or corresponds to a node in the original graph G.*
2. *T^{D} is a (directed) forest.*
3. *For each walk in matrix $\mathbb{E}$, there is an (undirected) path in T^{D} between two dummy nodes corresponding to this walk.*
4. *For each dummy node d_i in T^{D}, there is at least one other dummy node d_j in T^{D} which is connected in T^{D}; however, they are not necessarily strongly connected.*
5. *Any undirected path in T^{D} from one dummy node d_m to another dummy node d_n is of one of the three following types:*
   (a) *path is only **up** the tree; i.e., if the path goes from node i to j, then j is a predecessor of node i in T^{D},*
   (b) *path is **down** the tree; i.e., if the path goes from node i to j, then j is a successor of node i in T^{D},*
   (c) *path is first **up** and then **down** the tree.*

Proof: Property 1 is immediate.

For property 2, we note that each dummy node corresponds to an arc in the original graph; that is, the original arc is replaced by a dummy node and one or two arcs. Hence, the tree structure of G is preserved in the construction of T^{D}. However, the connectivity property might be lost, leading to a forest.

Consider any walk from a column $\bar{j}_1 \in \bar{J}$ to $\bar{j}_2 \in \bar{J}$ for a scenario k. According to construction of T^{D}, the columns $\bar{j}_1$ and $\bar{j}_2$ correspond to dummy nodes in T^{D}. In the walk, proceeding from one column $j^k_1 \in J^k$ to another column $j^k_2 \in J^k$ corresponds to arc (j_1, j_2) in T^{D} (when using

appropriate labeling of the columns in $\mathbb{E}$). Hence, any walk in $\mathbb{E}$ corresponds to a path in T^{D} between two dummy nodes.

Property 4 follows from property 3.

Property 5 is immediate when using the facts that T^{D} is a forest along with Lemma 3. □

Recognize that, in general, not every path in T^{D} between two dummy nodes corresponds to a walk in $\mathbb{E}$; that is, property 3 of Lemma 6 is not a 1-to-1 correspondence between walks in $\mathbb{E}$ and paths in T^{D}.

Now, we are ready to prove the main theorem.

Theorem 5 *If the graph G is a tree, then the constraint matrix defined by (123) - (125) is totally unimodular.*

Proof: With Lemma 1, the constraint matrix (123) - (125) is totally unimodluar, if and only if matrix $\mathbb{C}$ is totally unimodular.

In order to prove that $\mathbb{C}$ is totally unimodular, we must show that any (square) Eulerian submatrix is singular or that the sum of all entries is 0 mod 4, using Theorem 4. Therefore, let $\mathbb{E}$ be any square Eulerian submatrix of $\mathbb{C}$.

We use the notation for J, $\bar{J} \subset J$ and $J^k \subset J$ as introduced above.

Assume that $\mathbb{E}$ does not contain any column of $\bar{J}$. In this case, $\mathbb{E}$ is singular, as matrix $\mathbb{C}$ without the first m columns is totally unimodular. Therefore, assume that $\mathbb{E}$ contains at least one column of $\bar{J}$ and $\mathbb{E}$ does not contain any column or row consisting only of 0 entries (otherwise, $\mathbb{E}$ is singular).

Observe that each row sum of $\mathbb{E}$ has either value -2 or 0 (due to the network structure of $\mathbb{B}$); in particular, the row sum of $\mathbb{E}$ is not $+2$.

We have to show that there is always an even number of rows having row sum equal to -2, implying that the sum of all entries is 0 mod 4.

Therefore, construct the forest T^{D} for $\mathbb{E}$ with procedure CT.

Let us identify the row sums (mod 4) of any "walk" in $\mathbb{E}$ from $\bar{j}_1 \in \bar{J}$ to $\bar{j}_2 \in \bar{J}$ for scenario k and let us name this sum the "**walk sum**." Furthermore, we call the corresponding row with entry -1 for column $\bar{j}_1$ the **start row** of the walk, the corresponding row with entry -1 for column $\bar{j}_2$ the **end row** and all other rows visited by the walk **intermediate** rows. The start row and the end row of any walk have either sum -2 or 0. All intermediate rows have sum 0, as they contain exactly one $+1$ and one -1 entry. Hence, the sum of all entries in a walk in $\mathbb{E}$ has either value 0, -2, or -4.

Let us examine the forest T^{D}. We already have the connection of a walk in $\mathbb{E}$ to a path in T^{D} via property 3 of Lemma 6. Now, let us assign the values of the three possible walks identified in property 5 of Lemma 6:

1. path is only **up** the tree: this means that the start row sum is -2 and the end row sum is 0. The corresponding walk in $\mathbb{E}$ has value -2 mod 4.

2. path is **down** the tree: this means that the start row sum is 0 and end row sum is -2. The corresponding walk in $\mathbb{E}$ has value -2 mod 4,

3. path is first **up** and then **down** the tree: this means that the start row sum is -2 and the end row sum is -2 as well. The corresponding walk in $\mathbb{E}$ has value 0 mod 4.

Through the proof of Lemma 4, we know that we can always construct a set of walks, which are row distinct, for matrix $\mathbb{E}$ which have each of the -1 entries of the columns in $\bar{J}$ as either a start or an end row (this is due to the "marking" argument in the proof). Row distinct means in particular that none of the start and end rows are the same for any two walks in this set.

If we can show that the row sum of all walks in this set of walks is 0 mod 4, then the proof is complete. Or, equivalently, one can show that for the paths corresponding to the set of walks going up the tree at a path (between two dummy nodes), implies going down the tree along a path (between two dummy nodes). We will show the latter.

Using Lemma 5, we know that each dummy node $d \in T^{\mathrm{D}}$ has to be visited exactly two times by paths, corresponding to walks of the set of walks.

Let us denote by a **loop** a set of paths in T^{D}, where each path is between two dummy nodes, the end node of one path is the start node of another path, and starting at one particular dummy node d, following the paths will lead back to dummy node d. These paths neither have to be arc nor node distinct.

Now, we know that the number of visits per dummy node is two and that there exists such a collection of paths which visit all the dummy nodes, resulting from the set of walks. This implies that we can group this set of paths into loops in the forest T^{D}.

Therefore, let us consider one such loop. If the loop contains only paths first **up** and then **down** the tree, then the corresponding walk sum is 0 and there is nothing to show. Otherwise, let this loop contain one path going only up the tree (or only down the tree). However, as T^D consists of trees and the loop is closed, we eventually have to go down the tree whenever we go up the tree. Hence, the number of paths going only up equals the number of path going only down, leading to a loop sum of 0 mod 4.

This concludes the proof. □

Recognize that for the proof of Theorem 5, we used Lemma 3 (and hence the tree property) indirectly.

Second Proof via Matroids

In this section, we provide a short (but much more involved) proof of Theorem 5 via Matroid theory. An excellent overview of Matroids is given in the book [147]; particularly, we are using Chapters 9 and 11 in this section.

We restrict ourselves again to directed, connected graphs G, which are out-rooted trees. Recall the following notation adapted to the tree G:

- $\mathbb{B}$ - arc-node incidence matrix of graph G with dimension $(n-1 \times n)$
- $\mathbb{B}^T$ - node-arc incidence matrix of graph G with dimension $(n \times n-1)$
- $\mathbb{I}$ - identity matrix (with appropriate dimension)
- $\mathbf{0}$ - matrix consistent of entries all 0 (with appropriate dimension)
- $\mathbb{C}$ - matrix, as defined in (131).

Consider the following matrix

$$\mathbb{A} = \left(\begin{array}{cccc} \mathbb{B}^T & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbb{B}^T & \vdots & \mathbf{0} \\ \vdots & \cdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbb{B}^T \\ \hline -\mathbb{I} & -\mathbb{I} & \cdots & -\mathbb{I} \end{array}\right) \quad , \tag{135}$$

where $\mathbb{I}$ has dimension $(n-1 \times n-1)$. By re-numerating the columns, matrix $\mathbb{A}^T$ is the matrix of our interest: $\mathbb{C}$. Thus, $\mathbb{A}$ is TU, if and only if $\mathbb{C}$ is TU.

With Lemma 1, we can consider

$$\mathbb{G} := \mathbb{A}\,|\,\mathbb{I} = \left(\begin{array}{ccccc|c} \mathbb{B}^T\,|\,\mathbb{I} & \mathbf{0} & \cdots & \mathbf{0} & & \mathbf{0} \\ \mathbf{0} & \mathbb{B}^T\,|\,\mathbb{I} & \vdots & \mathbf{0} & & \mathbf{0} \\ \vdots & \cdots & \ddots & \vdots & & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbb{B}^T\,|\,\mathbb{I} & & \mathbf{0} \\ \hline -\mathbb{I}\,|\,\mathbf{0} & -\mathbb{I}\,|\,\mathbf{0} & \cdots & -\mathbb{I}\,|\,\mathbf{0} & & \mathbb{I} \end{array}\right) = \left(\begin{array}{ccccccc|c} \mathbb{B}^T & \mathbb{I} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbb{B}^T & \mathbb{I} & \vdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \cdots & \ddots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \cdots & \cdots & \mathbb{B}^T & \mathbb{I} & \mathbf{0} \\ \hline -\mathbb{I} & \mathbf{0} & -\mathbb{I} & \mathbf{0} & \cdots & -\mathbb{I} & \mathbf{0} & \mathbb{I} \end{array}\right) \quad ,(136)$$

where each of the K identity matrices added to the blocks $\mathbb{B}^T$ has dimension $(n \times n)$ and the additional identity matrix corresponding to the last block of $-\mathbb{I}$ matrices has dimension $(n-1 \times n-1)$.

In order to show that matrix $\mathbb{A}\,|\,\mathbb{I}$ is TU, we use specific transformations which do not alter the TU property and transform $\mathbb{A}\,|\,\mathbb{I}$ into a node-arc incidence matrix of some graph G. More specifically, we use the following

Proposition 2 ([142]) *Let H be a node-arc incidence matrix of a graph and F be a basis. Then matrix $F^{-1}H$ is TU.*

Without loss of generality, the last row of $\mathbb{B}^T$ corresponds to a leaf node of the tree. Let $\mathbb{F}$ be the matrix derived from $\mathbb{B}^T$ by deletion of that last row. Thus, $\mathbb{F}$ is square of dimension $(n-1 \times n-1)$, nonsingular (*i.e.*, invertible) and TU (as a row with exactly one non-zero entry has been removed). Furthermore, $\mathbb{F}$ is a basis of $\mathbb{B}^T$. This is the step where we exploit the assumption that G is a connected tree.

In matrix $\mathbb{G}$, modify the last row with the $-\mathbb{I}\,|\,\mathbf{0}$ matrices by adding multiples of the above rows. Specifically, premultiply the rows corresponding to the rows present in matrix $\mathbb{F}$ of each $\mathbb{B}^T\,|\,\mathbb{I}$ with $\mathbb{F}^{-1}$ and then add the resulting matrix to the last row with the $-\mathbb{I}\,|\,\mathbf{0}$ entries. This results in the modification of the latter row.

Matrix $\mathbb{G}$ now reads

$$\widetilde{\mathbb{G}} = \left(\begin{array}{ccccc|c} \mathbb{B}^T\,|\,\mathbb{I} & \mathbf{0} & \cdots & \mathbf{0} & & \mathbf{0} \\ \mathbf{0} & \mathbb{B}^T\,|\,\mathbb{I} & \vdots & \mathbf{0} & & \mathbf{0} \\ \vdots & \cdots & \ddots & \vdots & & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbb{B}^T\,|\,\mathbb{I} & & \mathbf{0} \\ \hline \mathbf{0}\,|\,\mathbb{F}^{-1} & \mathbf{0}\,|\,\mathbb{F}^{-1} & \cdots & \mathbf{0}\,|\,\mathbb{F}^{-1} & & \mathbb{I} \end{array}\right) \quad . \tag{137}$$

Now multiply the last row with $-\mathbb{F}$. This produces an overall matrix as follows:

$$\overline{\mathbb{G}} = \left(\begin{array}{ccccc|c} \mathbb{B}^T\,|\,\mathbb{I} & \mathbf{0} & \cdots & \mathbf{0} & & \mathbf{0} \\ \mathbf{0} & \mathbb{B}^T\,|\,\mathbb{I} & \vdots & \mathbf{0} & & \mathbf{0} \\ \vdots & \cdots & \ddots & \vdots & & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbb{B}^T\,|\,\mathbb{I} & & \mathbf{0} \\ \hline \mathbf{0}\,|\,-\mathbb{I} & \mathbf{0}\,|\,-\mathbb{I} & \cdots & \mathbf{0}\,|\,-\mathbb{I} & & -\mathbb{F} \end{array}\right) \quad . \tag{138}$$

Matrix $\overline{\mathbb{G}}$ is TU, because in each column with two non-zero entries, the sum of all entries with two nonzero entries amount to $\mathbf{0}$; *cf.* [107, Proposition 2.6]. This completes the proof of Theorem 5.

Note: By removing the one column of matrix $\mathbb{F}$ which has exactly one non-zero entry, the resulting matrix $\widetilde{\mathbb{G}}$ is the node-arc incidence matrix of some graph, as each column contains exactly one $+1$ and one -1 entry.

This matrix transformation from (135) to (138) is a particular case of the single commodity representation produced by the multicommodity network transformation described in [142], where we mainly use the step 3 b) of the transformation algorithm in [142].

The presented proof provides us with a different insight into two-stage minimum $s-t$ cut problems for trees: The two-stage minimum $s-t$ cut problem for a tree G is equivalent to a single commodity flow problem for the graph represented by $\widetilde{\mathbb{G}}$.

4.3 Computational Complexity

The fact that the constraint matrix of the two-stage stochastic minimum $s-t$ cut problem loses its total unimodularity raises the question about the theoretical computational complexity of the problem. In this section, we show that the stochastic programming extension of the polynomially solvable $s-t$ minimum cut problem becomes NP-hard in general. This is consistent with similar observations for the two-stage stochastic extensions of the minimum spanning tree [61] and maximum weight matching [90] problems. In this section for simplicity of further discussion we consider the undirected version of the problem. Define the decision version of two-stage stochastic $s-t$ cut problem as follows:

Definition 3 (Decision Version)
Instance: *A graph $G=(V,E)$ with node set V, arc set E, root $s\in V$, and K scenarios. The k-th scenario consists of a single terminal t^k and has probability p^k of being realized. Arc $ij\in E$ has cost c_{ij} in the first stage and d_{ij}^k in the recourse stage (or second stage) if the k-th scenario is realized.*
Question: *Is there a set of arcs E_0 to be cut in the first stage and for each scenario k, an arc set E_k to be cut in the recourse stage if scenario k is realized, such that removing $E_0\cup E_k$ from the graph G disconnects s from the terminal t^k, while the expected cost of cutting $c:=\sum_{ij\in E_0}c_{ij}+\sum_{k=1}^{K}p^k\sum_{ij\in E_k}d_{ij}^k$ over all scenarios does not exceed C,* i.e., *$c\le C$?*

We call the arc set $E_0\cup E_k$ a *feasible cut* for scenario k, if the removal of $E_0\cup E_k$ from the graph G disconnects s from the terminal t^k. In our reduction, we use the Multiterminal Cut (MC) problem.

Definition 4 *(Multiterminal Cut) [42]*
Instance: *A graph $G(V,E)$, a set of $S=\{s_1,s_2,\ldots,s_\kappa\}\subseteq V$ of κ specified vertices or terminals, and a positive weight $w(e)$ for each arc $e\in E$ and a bound B.*
Question: *Is there a subset of arcs $\bar{E}\subseteq E$ with $w(\bar{E})\le B$ such that the removal of $\bar{E}$ from E disconnects each terminal from all others?*

An arc set $\bar{E}\subseteq E$ is called a *feasible cut* for MC, if the removal of $\bar{E}$ from E disconnects each terminal from all others. We need the following complexity result.

Theorem 6 ([42]) *The Multiterminal Cut problem for $k=3$ and arbitrary graphs is NP-complete even if all edge weights are equal to 1.*

Note that for $\kappa=2$ the MC problem reduces to the standard minimum $s-t$ cut problem which can be solved in polynomial time.

We now establish the strongly NP-completeness of the two-stage stochastic $s-t$ cut problem by reducing the MC problem with $\kappa=3$ to it. We are given an instance of the MC problem with $G=(V,E)$, respective weights w_{ij} for each $ij\in E$, $S=\{s_1,s_2,s_3\}$ and bound B. Without loss of generality, assume that arcs s_1s_2 and s_1s_3 do not exist in G, *i.e.*, $s_1s_2,s_1s_3\notin E$. Next we construct an instance of the two-stage stochastic $s-t$ cut problem such that there is a one-to-one correspondence of their respective solutions. Define $G_{MC}=(\tilde{V},\tilde{E})$ as follows:

- Let $\tilde{V} = V \cup \{t\}$ and $\tilde{E} = E \cup \{s_1s_2, s_1s_3, s_2t, s_3t\}$. In other words, we add an extra node and four additional arcs into the original graph.
- First-stage arc capacities: $c_{ij} = w_{ij}\ \forall\ ij \in E$ and $c_{ij} = +\infty$ for $ij \in \{s_1s_2, s_1s_3, s_2t, s_3t\}$.
- Second-stage arc capacities: $d^1_{ij} = d^2_{ij} = +\infty\ \forall\ ij \in E$; $d^1_{s_1s_2} = d^1_{s_3t} = d^2_{s_1s_3} = d^2_{s_2t} = +\infty$ and $d^1_{s_1s_3} = d^1_{s_2t} = d^2_{s_1s_2} = d^2_{s_3t} = 0$.
- Let the source node s be s_1 and the sink node be t for both scenarios.
- Let the probability of each scenario be given by $p_1 = p_2 = 1/2$.

An example and the corresponding transformation is shown in Figure 5. The MC instance in Figure 5(a) is a "YES" instance for $B \geq 10$, with the feasible cut set $\bar{E} = \{s_2s_3, s_24, s_34, s_35\}$. This set is marked by the gray lines. Figure 5(b) shows the transformed graph G_{MC} with a cut having weight 11.

Lemma 7 *An instance for the Multiterminal Cut problem with bound B is a "YES" instance, if and only if the transformed graph G_{MC} is a "YES" instance for the two-stage stochastic $s-t$ cut problem with cost bound $C = B$.*

Proof: "⇒" Let the MC problem have the feasible cut set $\bar{E}$ with $w := w(\bar{E}) \leq B$. The first stage cuts for G_{MC} are $\bar{E}$ as well, for the first scenario, arcs (s_1s_3) and (s_2t) are cut in the second stage while arcs (s_1s_2) and (s_3t) are cut in the second stage for scenario two. In this way, the cut weight is $c \equiv w$. This leads to a feasible cut for G_{MC} for the two-stage $s-t$ cut problem for both scenarios with weight $c \leq C = B$.

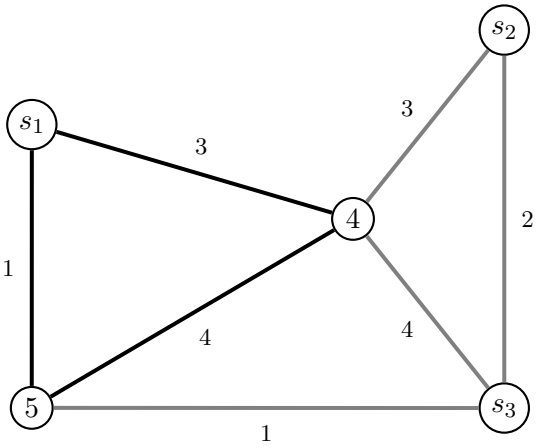
"⇐" Suppose the solution of the constructed two-stage stochastic minimum $s-t$ cut problem is given by the arc set $\tilde{E}_0$ for the first stage and arc sets $\tilde{E}_1$ and $\tilde{E}_2$ for scenarios 1 and 2 in the second stage, respectively, and cut weight C. Because $C < \infty$, any feasible two-stage $s-t$ cut has finite total capacity. We can observe the following:

- We need $\tilde{E}_0 \subseteq E$ since $c_{ij} = +\infty$ for any $ij \in \tilde{E} \setminus E$.
- We may assume that $\tilde{E}_1 = \{s_1s_3, s_2t)\}$ and $\tilde{E}_2 = \{s_1s_2, s_3t\}$ since the respective capacities are zero, while all other capacities are $+\infty$.
- For scenario 1, since $d^1_{s_1s_2} = d^1_{s_3t} = +\infty$, arcs s_1s_2 and s_3t are contained in the graph. Therefore, $\tilde{E}_0$ must contain arcs that completely disconnect s_1 from s_3 and s_2 from s_3.
- For scenario 2, since $d^2_{s_1s_3} = d^2_{s_2t} = +\infty$, arcs s_1s_3 and s_2t are contained in the graph. Therefore, $\tilde{E}_0$ must contain arcs that completely disconnect s_1 from s_2 and s_3 from s_2.

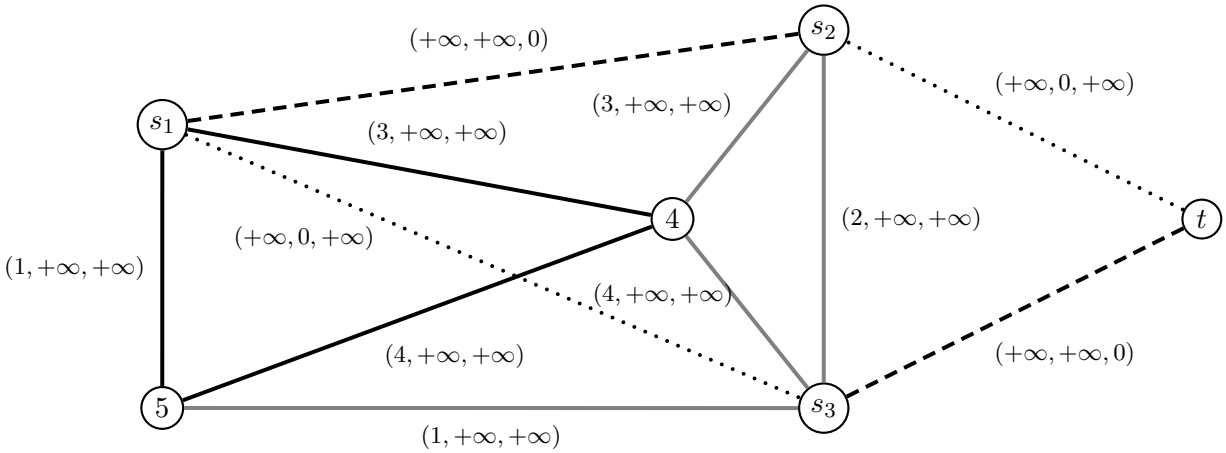
Therefore, any two-stage stochastic $s-t$ cut with the finite total capacity must contain $\tilde{E}_0$ that completely disconnects s_1, s_2 and s_3 from each other, *i.e.*, $\tilde{E}_0$ is a multiterminal cut in the original graph $G(V, E)$. Moreover, since the capacities of arcs in $\tilde{E}_1$ and $\tilde{E}_2$ are zero, then minimizing the total capacity of the two-stage $s-t$ cut corresponds to minimizing the weight of the multiterminal cut. Thus, $\tilde{E}_0$ is a feasible cut for MC with weight $B = C$. □

This allows us to prove the main result.

Theorem 7 *The decision version of the two-stage stochastic $s-t$ cut problem is NP-complete in the strong sense even for two scenarios and the same terminal node for both scenarios.*



(a) Instance for MC problem with cut.



(b) Corresponding graph G_{MC} with cut.

Figure 5: MC instance and corresponding instance G_{MC} for the two-stage stochastic minimum $s-t$ cut problem. The legend for (b) is the same as in Figures 2 and 6, but rather for undirected arcs.

Proof: The two-stage stochastic $s-t$ cut problem is in NP, as a non-deterministic algorithm needs only to guess the arcs to be cut and check if this leads to a feasible cut for each scenario and if the cost of the cut is less than or equal to C.

The given transformation from MC to the two-stage stochastic $s-t$ cut problem via graph G_{MC} is valid according to Lemma 7. By replacing the weights $+\infty$ by $B+1$, the node set, the arc set and the arc weights of the constructed graph G_{MC} are linearly bounded in the input size of MC. Thus, the transformation can be done in (strongly) polynomial time. □

Remark 1 *The directed version of the two-stage stochastic $s-t$ cut problem is also NP-complete in the strong sense. To see this, one can use the same reduction as for the undirected case by preserving the direction of the arcs to be cut in the construction presented in the proof of Lemma 7.*

4.4 Linear Running Time Algorithm for Trees

As we discuss in Section 4.2.3, if graph G is a tree, then the constraint matrix (123) - (125) is totally unimodular. This fact indicates that the two-stage stochastic minimum $s-t$ cut problem is polynomially solvable if G is tree. Moreover, we show next that the problem admits a linear time solution algorithm.

Consider now the following transformation. Given an instance of the two-stage stochastic minimum $s-t$ cut problem on graph $G=(V,E)$ with the notation of Definition 1, construct a graph $\overline{G}=(\overline{V},\overline{E})$ with arc weight function $w: E \to \mathbb{R}^+$ as follows:

- Add one additional node T^k to V for each scenario k; *i.e.*, $\overline{V} := V \bigcup_{k=1}^{K} \{T^k\}$.
- Add one arc between terminal t^k and T^K to E for each scenario k; *i.e.*, $\overline{E} = E \bigcup_{k=1}^{K} \{t^k T^k\}$.
- Weights for $e \in E$ are the first stage cost; *i.e.*, $w(ij) = c_{ij} \quad \forall ij \in E$.
- Weights for arcs $t^k T^K$ are constructed as follows. Let P_k be the (unique) path from the source s to t^k, and $\tilde{e}_k$ be one least cost arc in scenario k (along this path P_k). Then $w(t^k T^k) = p^k d^k(\tilde{e}_k) \quad \forall k = 1, \ldots, K$.

Graph $\overline{G}$ remains a tree by construction. Now, finding a minimum cut in $\overline{G}$ which separates s from all terminals T^k can be done via a linear time dynamic programming algorithm. Any such cut in $\overline{G}$ corresponds then to a cut in G (and vise versa) with the same cost as follows: If arc $e \in E$ for $\overline{G}$ is cut, then in the first stage, e is cut in G; if arc $e = t^k T^K \in \overline{E} \setminus E$ is cut, then in the k-scenario, arc $\tilde{e}_k$ is cut. This proves the following Corollary:

Corollary 2 *If graph G is a tree, then the two-stage stochastic minimum $s-t$ cut problem can be solved in linear time.*

Adjusting the formulation of the deterministic minimum $s-t$ cut problem (118) - (121) to the cut problem for graph $\overline{G}$, one obtains:

$$\min \sum_{ij \in \overline{E}} w(ij) y_{ij} \tag{139}$$

$$s.t. \quad y_{ij} \geq x_j - x_i \qquad \forall ij \in \overline{E} \tag{140}$$

$$x_s = 0, \quad x_{t^k} = 1 \qquad \forall k = 1, \ldots, K \tag{141}$$

$$x_i, y_{ij} \in [0,1] \qquad \forall i \in \overline{V}, ij \in \overline{E}. \tag{142}$$

The constraint matrix defined by (140) - (141) is TU. However, this does not (at least in an obvious manner) imply that $\mathbb{C}$ is TU as well for the tree G.

4.5 Node-Based Version

We define two mathematical programming formulations as *equivalent*, if and only if both formulations have the same set of feasible and optimal solutions. It is well-known that the minimum $s-t$ cut problem can be equivalently reformulated as the following quadratic 0–1 program [25]:

$$\min \sum_{ij\in E} c_{ij}(1-x_i)x_j \tag{143}$$

$$s.t.\ x_s = 0, \quad x_t = 1 \tag{144}$$

$$x_i \in \{0,1\} \qquad \forall i \in V. \tag{145}$$

We also may consider the relaxed concave quadratic programming problem.

$$\min \sum_{ij\in E} c_{ij}(1-x_i)x_j \tag{146}$$

$$s.t.\ x_s = 0, \quad x_t = 1 \tag{147}$$

$$0 \le x_i \le 1 \qquad \forall i \in V. \tag{148}$$

For a concave minimization problem over a bounded polytope, there always is an optimal solution at a corner point of the polytope. Thus, formulations (143) - (145) and (146) - (148) are equivalent.

Due to the nonlinearity of the objective function in (143), the resulting formulation (143)-(145) defines the minimum $s-t$ cut problem without variables y_{ij} that appear in (118)-(121). As we discuss in Section 4.1, variables y_{ij} have a clear interpretation in terms of the *cutset* that can be also used for the equivalent definition of the deterministic minimum $s-t$ cut problem. This observation leads to an alternative definition of the two-stage stochastic minimum $s-t$ cut problem.

Definition 5 (two-stage stochastic minimum $s-t$ cut; node-based version) *Given is a directed graph $G=(V,E)$ with node set V and arc set E and a root $s \in V$. There are K scenarios. The kth scenario consists of a single terminal t^k and has probability p^k of being realized. Arc $ij \in E$ has cost c_{ij} in the first stage and d_{ij}^k in the recourse stage (or second stage) if the kth scenario is realized. The task is to find two node sets S, $T \subset V$ and for each scenario k, additional two node sets S^k, $T^k \subset V$ with $s \in S \cup S^k$, $t \in T \cup T^k$ and $S \cup S^k \cup T \cup T^k = V$ where S, T, S^k, T^k are mutually distinct. The objective is to minimize the expected cost over all scenarios:*

$$z^{N*} := \min \left(\sum_{ij\in E: i\in S, j\in T} c_{ij} + \sum_{k=1}^{K} p^k \sum_{ij\in E: i\in S^k\cup S, j\in T^k\cup T, i\notin S\vee j\notin T} d_{ij}^k \right).$$

We refer to this definition of the two-stage stochastic minimum $s-t$ cut problem as its *node-based version*; the original definition provided in Section 4.1 is further referred to as *the arc-based version.*

Consider an example given by Figure 1 and discussed in Section 4.2. An optimal solution using this node-based interpretation is shown in Figure 6. In the first stage, neither of the two nodes 2 and 3 is assigned. In the second stage, both nodes 2 and 3 are assigned to set T, corresponding to the terminal node 4. Hence, arcs (1,2) and (1,3) are cut in both scenarios, leading to a total cost of 11. Recognize that this solution is not unique.

Comparing the two optimal solutions from the arc-based version and the node-based version, we recognize that the difference in both interpretations is the role of the sets S and T. In the

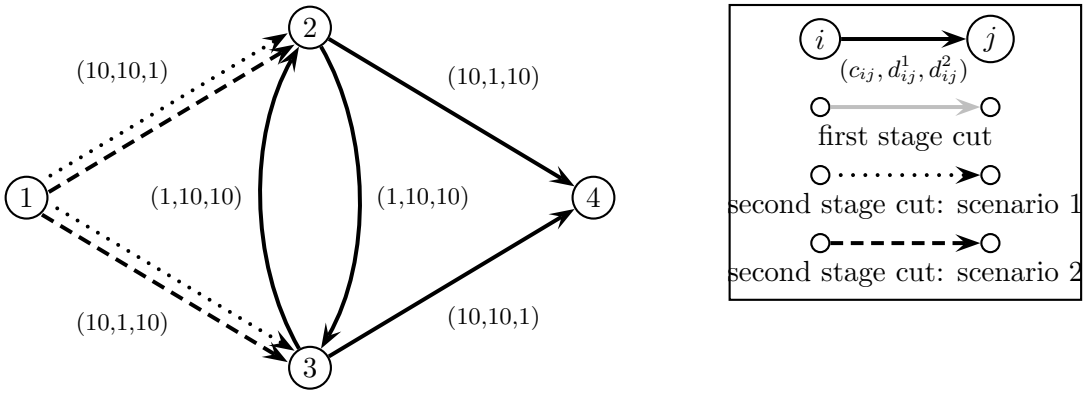


Figure 6: In the first stage, no node is assigned. In the second stage, nodes 2 and 3 are assigned to set T. The second stage decision is the same for both scenarios. The cost of this minimum cut is 11.

arc-based version, nodes 2 and 3 are not assigned to any of those sets, while the arcs (2,3) and (3,2) are both cut in the first stage. The assignments of the nodes are performed in the second stage, dependent on the scenario. The resulting solution cannot be obtained via the node-based version, as assignments of the nodes to one of the sets performed in the first stage are final. However, this is not possible in the node-based version of the problem. Generally speaking, in the node-based case the assignments of the nodes can either be done in the first stage or in the second stage, depending on the scenario that occurred. However, all the assignments performed in the first-stage are final and cannot be changed in the second stage. The cut is then the result of the assignments of the nodes in both stages.

Inspecting the two definitions of the two-stage minimum $s-t$ cut problem, one observes that any solution of the node-based version is also feasible for the arc-based version with the same objective function value. Such a solution can be obtained as follows. Let S, T, S^k, T^k be a partition of V defining a valid cut for the node-based version. Then, define the following cut for the arc-based version:

$$E_0 := \{ij \,|\, ij \in E \text{ and } i \in S \text{ and } j \in T\} \tag{149}$$

$$E_k := \{ij \,|\, ij \in E \text{ and } i \in S \cup S^k \text{ and } j \in T^k\} \qquad \forall k. \tag{150}$$

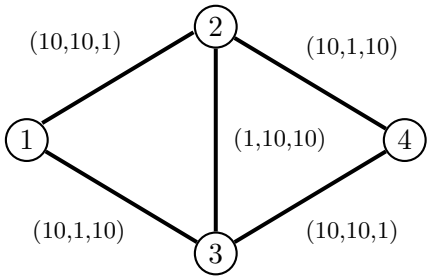
This is summarized in the next proposition.

Proposition 3 $z^{A*} \leq z^{N*}$.

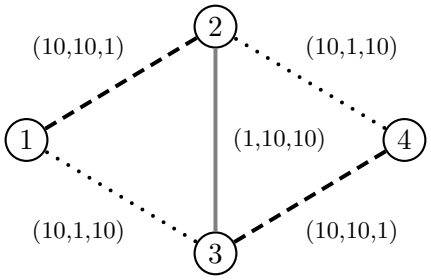
Furthermore, one can observe that the arc-based and node-based versions are equivalent when G is a tree. The reason is that for trees selecting an arc $ij \in E$ in the first stage is equivalent to assigning node i to set S and node j to set T.

Corollary 3 *The arc-based and node-based versions of the two-stage stochastic minimum $s-t$ cut problem are equivalent and solvable in linear time if graph G is a tree.*

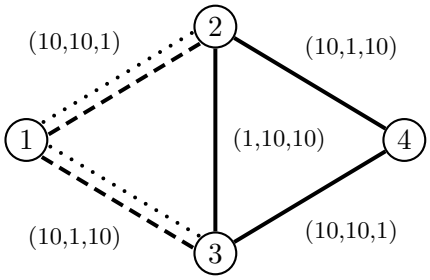
One may wonder whether this difference of the arc-based and the node-based version for directed graphs holds true as well for the corresponding undirected version of the problems. An answer is



(a) Instance for undirected two-stage minimum $s-t$ cut problems.



(b) Optimal solution for arc-based version with cost 2.



(c) Optimal solution for node-based version with cost 11.

Figure 7: Difference of node-based and arc-based versions for undirected graphs. The legend is the same as in Figures 2 and 6, but rather for undirected arcs.

provided in Figures 7, which demonstrates the difference of arc-based and node-based version of the two-stage stochastic minimum cut problem for undirected graphs.

We define the following:

$$x_i^S = \begin{cases} 1, & \text{if } i \in S \\ 0, & \text{otherwise} \end{cases} \qquad \text{and} \qquad x_i^T = \begin{cases} 1, & \text{if } i \in T \\ 0, & \text{otherwise} \end{cases}, \tag{151}$$

as well as

$$z_{ik}^S = \begin{cases} 1, & \text{if } i \in S^k \\ 0, & \text{otherwise} \end{cases} \qquad \text{and} \qquad z_{ik}^T = \begin{cases} 1, & \text{if } i \in T^k \\ 0, & \text{otherwise} \end{cases}. \tag{152}$$

Consider the following bi-linear, linearly constrained 0-1 programming formulation:

$$\min \sum_{ij \in E} \left(c_{ij} x_i^S x_j^T + \sum_{k=1}^{K} p_k d_{ij}^k \left(x_i^S z_{jk}^T + z_{ik}^S x_j^T + z_{ik}^S z_{jk}^T \right) \right) \tag{153}$$

$$\text{s.t. } x_i^S + x_i^T + z_{ik}^S + z_{ik}^T = 1 \qquad \forall i \in V, \quad k = 1, \ldots, K; \tag{154}$$

$$x_s^S + z_{sk}^S = 1 \qquad x_t^T + z_{tk}^T = 1 \qquad \forall k = 1, \ldots, K; \tag{155}$$

$$x_i^S, x_i^T, z_{ik}^S, z_{ik}^T \in \{0, 1\} \qquad \forall i \in V, \quad \forall k = 1, \ldots, K. \tag{156}$$

Constraints (154) ensure that each node $i \in V$ is assigned to exactly one of the sets S, T, S^k or T^k for each scenario k, while constraints (155) make sure that $s \in S \cup S^k$ and $t \in T \cup T^k$ for each

scenario k. The cost of the corresponding cut is evaluated in the objective (153) as follows: the first terms sum all the costs of all cut arcs $ij \in E$ with $i \in S$ and $j \in T$, while the second term sums all costs of the cut arcs $ij \in E$ with $i \in S^k \cup S$ and $j \in T^k$. This proves the following result.

Proposition 4 *Formulation* (153) - (156) *models the node-based version of the two-stage minimum* $s-t$ *cut problem correctly.*

By eliminating variables z_{ik}^S using the relations in equalities (154) and (155), one obtains the following continuous, box-constrained, bi-linear optimization problem:

$$\min \sum_{ij \in E} \left(c_{ij} x_i^S x_j^T + \sum_{k=1}^{K} p_k d_{ij}^k (x_j^T + z_{jk}^T - x_i^S x_j^T - x_i^T x_j^T - z_{ik}^T x_j^T - x_i^T z_{jk}^T - z_{ik}^T z_{jk}^T) \right) \tag{157}$$

$$\text{s.t. } x_s^T = z_{sk}^T = 0 \qquad x_t^S = z_{tk}^S = 0 \qquad \forall k = 1, \ldots, K \tag{158}$$

$$x_i^S, x_i^T, z_{ik}^T \in [0,1] \qquad \forall i \in V, \quad \forall k = 1, \ldots, K. \tag{159}$$

Proposition 5 *Formulations* (153) - (156) *and* (157) - (159) *are equivalent.*

Proof: We need to show that any optimal solution of (157) - (159) is binary. This follows from the fact that there are no quadratic terms but only bi-linear expressions in the objective. □

4.6 Concluding Remarks

Based on two equivalent formulations of the classical minimum $s-t$ cut problem, we introduce two different versions (arc-based and node-based) of the two-stage stochastic minimum $s-t$ cut problem. These versions are equivalent if the considered graph is a tree; however, in the general case they lead to different solutions. We provide a mathematical programming formulation for the arc-based version that is motivated by a standard linear 0–1 programming model for the deterministic minimum $s-t$ cut problem. We prove that the constraint matrix of the new formulation loses its total unimodularity property, in general; however, the matrix preserves the property if the considered graph is a tree. This fact turns out to be not surprising as we show that similar to many other stochastic extensions of classical combinatorial optimization problems, (*e.g.*, minimum spanning tree [61]) the arc-based version of the two-stage stochastic minimum $s-t$ cut problem is NP-hard. In the case of trees, the two-stage stochastic minimum $s-t$ cut problem is polynomially solvable due to the total unimodularity property; we also describe a simple linear time solution algorithm. The computational complexity of the node-based version has yet to be fully explored (*e.g.*, its NP-hardness remains open).

5 Bilevel Knapsack Problems with Stochastic Right-Hand Sides

The details of the work in this chapter can be found in:

- O.Y. Ozaltin, O.A. Prokopyev, A.J. Schaefer, "The Bilevel Knapsack Problem with Stochastic Right-Hand Sides," *Operations Research Letters*, Vol. 38/4 (2010), pp. 328–333.

Bilevel programs [41, 45, 106] model the hierarchical relationship between two autonomous, and possibly conflicting, decision makers: the *leader* and the *follower*. This hierarchical relationship results from the fact that the follower's problem is affected by the decision of the leader. Moreover, the follower's decision in return affects the leader's problem.

The bilevel knapsack problem was first considered by Dempe and Richter [46]. In this problem, the follower solves a 0–1 knapsack problem subject to the capacity set by the leader. The leader earns a profit from the items selected by the follower, and both decision makers seek to maximize their own profits. Dempe and Richter [46] formulated this problem as a mixed-integer bilevel program, and proposed a branch-and-bound algorithm. Recently, Brotcorne et al. [27] considered the same problem, and developed a dynamic programming algorithm that outperformed Dempe and Richter's [46] branch-and-bound algorithm.

In our work we introduce the *bilevel knapsack problem with stochastic right-hand sides* (BKPS). BKPS is a stochastic extension of the bilevel knapsack problem where the leader's decision has an uncertain effect on the follower's knapsack capacity. We model this uncertainty using a finite set of scenarios. Brotcorne et al. [27] identified an application of the bilevel knapsack problem in revenue management, where a company (i.e., the leader) determines the number of units to sell by itself, and handing the remainder over to an intermediary (i.e., the follower). In this context, BKPS arises when there is uncertainty in the number of units transferred to the intermediary. For example, in the distribution of perishable goods [72], some items may be spoiled during the shipment process.

Consider a set of n items where each item $j \in \{1, \dots, n\}$ has an associated weight $a_j \in \mathbb{Z}^1_+$, and two revenues: the follower's revenue $c_j \in \mathbb{R}^1_+$, and the leader's revenue $d_j \in \mathbb{R}^1_+$. The follower must solve a knapsack problem to maximize her own objective subject to a capacity $h(\omega, y)$ that depends on the leader's choice of y as well as a discretely distributed random variable $\omega \in \Omega$. This yields the following stochastic bilevel program:

$$\begin{aligned}
[\text{BKPS}] \quad \text{maximize} \quad & f(y, X) = ty + \mathbb{E}_\omega \left[d^T \mathbf{x}(\omega, y) \right] && \text{(160a)} \\
\text{subject to} \quad & \underline{b} \leq y \leq \bar{b}, \quad y \in \mathbb{R}^1, && \text{(160b)} \\
& \mathbf{x}(\omega, y) \in R\left(h(\omega, y)\right) \qquad \forall\, \omega \in \Omega, && \text{(160c)}
\end{aligned}$$

where $R(h(\omega, y)) = \operatorname{argmax} \left\{ \mathbf{c}^T x : \ \mathbf{a}^T x \leq h(\omega, y), \ \mathbf{x} \in \{0,1\}^n \right\}$, i.e., the follower's *rational reaction set*. For $y \in [\underline{b}, \bar{b}]$, X is an $|\Omega| \times n$ binary matrix whose rows represent the subset of items selected by the follower under the scenario $\omega \in \Omega$, i.e., $\mathbf{x}(\omega, y)$. We assume that $h(\omega, y) : \Omega \times \mathbb{Z}^1 \to \mathbb{Z}^1_+$ is a nondecreasing function of y, and that $h(\omega, \bar{b})$ is finite for all $\omega \in \Omega$.

In our paper cited above we provide necessary and sufficient conditions for the existence of an optimal solution. When the leader's decisions can take only integer values, we present an equivalent two-stage stochastic programming reformulation with binary recourse. We develop a branch-and-cut algorithm for solving this reformulation, and a branch-and-backtrack algorithm for solving the scenario subproblems. Computational experiments indicate that our approach can solve large instances in a reasonable amount of time.

6 Two-Stage Stochastic Assignment Problems

This chapter is mostly based on the results from:

- S. Karademir, O.A. Prokopyev, N. Kong, "On Greedy Approximation Algorithms for a Class of Two-Stage Stochastic Assignment Problems," Technical report, 2011

6.1 Introduction

Given a set V of n agents, a set U of n jobs and a weight (or cost) w_{ij} for each $i \in V$ and $j \in U$, the well-known *linear assignment problem* consists of assigning each agent to exactly one job in such a manner that each job is performed by exactly one of the agents and the total weight (cost) of the obtained assignment is maximized (minimized). In this chapter we consider the maximization version and the mathematical program for the related linear assignment problem can be given as follows [113]:

$$\max_{x} \quad \sum_{i \in V} \sum_{j \in U} w_{ij} x_{ij} \tag{161}$$

$$\text{s.t.} \quad \sum_{j \in U} x_{ij} = 1, \quad \text{for all } i \in V, \tag{162}$$

$$\sum_{i \in V} x_{ij} = 1, \quad \text{for all } j \in U, \tag{163}$$

$$x_{ij} \in \{0,1\}, \quad \text{for all } i \in V,\ j \in U. \tag{164}$$

Linear assignment problem (161)-(164) is also known as the *weighted bipartite matching problem* [113]. Namely, given a weighted bipartite graph $G(V \cup U, E)$ with $|V| = |U|$ and arc weights w_{ij} for all $(i,j) \in E$ we need to find a *perfect matching* of maximum weight. Recall that perfect matching is a matching which matches all vertices of the graph.

It is well known that the constraint matrix for (162)-(163) is totally unimodular [113]. Therefore, we can safely remove integrality constraints (164) and solve the linear programming relaxation (161)-(163) to get the optimal solution. However, the most popular approach to tackle the linear assignment problem is the `Hungarian Method` [91], which can be considered as an implementation of the primal-dual method for the respective minimum cost flow problem [113]. The `Hungarian Method (HM)` works with the dual of the linear program (161)-(163) given by

$$\min_{\alpha,\beta} \quad \sum_{i=1}^{n} \alpha_i + \sum_{j=1}^{n} \beta_j \tag{165}$$

$$\text{s.t.} \quad \alpha_i + \beta_j \geq w_{ij}, \quad i = 1,\ldots,n, \quad j = 1,\ldots,n. \tag{166}$$

In this chapter we are concerned with the following two-stage stochastic programming extension of (161)-(164), which is further referred to as the *two-stage stochastic linear assignment (2SSLA) problem*. Each edge (i,j), $i \in V$ and $j \in U$, is associated with the first-stage weight w_{ij}, and the second-stage weight q_{ij}^k for scenario k, $k = 1,\ldots,K$. The first-stage decision x is to choose some

matching in G that is not necessarily perfect. At the second stage a scenario k is realized with probability p_k. For each scenario k, the second-stage decision y^k is to choose a matching over those agents and jobs that are unmatched in the first stage in order to form a perfect matching. The overall goal is to find a perfect matching with the maximum expected weight. Then the two-stage stochastic programming extension of (161)-(164) can be written as follows:

$$\max_{x,y} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} \cdot x_{ij} + \sum_{k=1}^{K} p_k \cdot \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}^k \cdot y_{ij}^k \tag{167}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} \left(x_{ij} + y_{ij}^k\right) = 1, \qquad i = 1,\ldots,n, \quad k = 1,\ldots,K, \tag{168}$$

$$\sum_{i=1}^{n} \left(x_{ij} + y_{ij}^k\right) = 1, \qquad j = 1,\ldots,n, \quad k = 1,\ldots,K, \tag{169}$$

$$x_{ij} \in \{0,1\}, \quad y_{ij}^k \in \{0,1\}, \qquad i,j = 1,\ldots,n, \quad k = 1,\ldots,K. \tag{170}$$

A number of studies demonstrate the advantage of stochastic programming models over deterministic approaches [23]. Recent examples of these types of studies in the literature include two-stage stochastic extensions of the shortest path [70], minimum spanning tree [49, 61], min-cut [48, 70], and Steiner tree [75] problems. For a detailed introduction to stochastic programming, we refer the reader to [23, 133]. Next we briefly describe two papers that are most closely related to our work in this chapter.

Kong and Schaefer [88] consider the two-stage stochastic maximum weight matching problem on general graphs. They show that the problem is NP-hard and propose a greedy $\frac{1}{2}$-approximation algorithm. Escoffier et al. [52] prove that the two-stage stochastic maximum weight matching problem is APX-complete even for bipartite graphs of maximum degree 4 and general graphs of degree 3, which implies that there is no polynomial-time approximation scheme (PTAS) for this problem as long as $P \neq NP$. Based on the concepts from [88], they also provide a greedy $\max\{\frac{K}{2K-1}, \frac{\Delta}{2\Delta-1}\}$-approximation algorithm, where K is the number of scenarios in the second-stage and Δ is the degree of the bipartite graph.

Our work is essentially built on these two studies [52, 88]. First, we consider the greedy approximation methods from these papers for the two-stage stochastic linear assignment problem. Since the maximum weight matching problem on bipartite graphs can be easily reduced to the linear assignment problem via addition of dummy agents and/or jobs, the 2SSLA problem is also APX-complete. We propose a necessary optimality condition that generalizes and unifies the key ideas behind the two algorithms by Kong and Schaefer [88] and Escoffier et al. [52]. Then based on this optimality condition, we design a new greedy approximation algorithm referred to as `EGA`. While the developed approach preserves the existing approximation guarantees, we are not able to prove whether `EGA` provides a better approximation bound. However, analytical observations and extensive computational results indicate that `EGA` has strictly better results on some rather broad classes of the two-stage stochastic linear assignment problem.

6.2 Greedy Approximation Algorithms

6.2.1 Basic Greedy Approach

Since the 2SSLA problem is NP-hard, we can not expect to solve it exactly for large input sizes. Hence, we seek for an approximation algorithm that will have a reasonable performance guarantee.

We first discuss the `Greedy Algorithm (GA)` for more general two-stage stochastic matching problem given in [88]. Since linear assignment is a specific case of maximum weight matching problem, this algorithm can be simply adopted to 2SSLA with the same performance guarantee. For further discussion we need the following notation:

Definition 1 *A first-stage myopic solution is an optimal solution to:*

$$(\mathbf{GA-I}) : \max \left\{ \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_{ij} \;\Big|\; \forall j \;\sum_{i=1}^{n} x_{ij} = 1,\; \forall i \;\sum_{j=1}^{n} x_{ij} = 1;\; \forall i,j \; x_{ij} \in \{0,1\} \right\}. \tag{171}$$

Definition 2 *A second-stage myopic solution for scenario k is an optimal solution to:*

$$(\mathbf{GA-II}) : \max \left\{ \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}^{k}y_{ij}^{k} \;\Big|\; \forall j \;\sum_{i=1}^{n} y_{ij}^{k} = 1,\; \forall i \;\sum_{j=1}^{n} y_{ij}^{k} = 1;\; \forall i,j \; y_{ij}^{k} \in \{0,1\} \right\}. \tag{172}$$

First- and second-stage myopic solutions are the solutions corresponding to deterministic linear assignment problems with the appropriate choices of weights in the objective functions. Let $\mathbf{x}^{GA}$ and Z_1^{GA} be the first-stage myopic solution and the respective optimal objective function value. Similarly, let $\mathbf{y}_k^{GA}$ and Z_{2k}^{GA} be the second-stage myopic solution and the respective optimal objective function value for scenario k. Finally, denote by Z_2^{GA} the expected value of the second-stage myopic solutions, i.e.,

$$Z_2^{GA} = \sum_{k=1}^{K} p_k Z_{2k}^{GA}.$$

`GA` works as follows. Initially, it finds the first-stage myopic solution (`GA-I`) as well as the second-stage myopic solutions for each scenario (`GA-II`). Then it compares the objective function value of the first-stage myopic solution with the expected objective function value of the second-stage myopic solutions. The final assignment weight Z^{GA} corresponds to the better of them and is given by

$$Z^{GA} = \max\left\{ Z_1^{GA},\; Z_2^{GA} \right\}. \tag{173}$$

The final agent-job assignments are given either by $(\mathbf{x}^{GA}, \mathbf{0}, \ldots, \mathbf{0})$ or by $(\mathbf{0}, \mathbf{y}_1^{GA}, \ldots, \mathbf{y}_K^{GA})$, respectively. That is all assignments are made completely either at the first stage, or at the second stage for each scenario.

Theorem 3 `Greedy Algorithm` *is an approximation algorithm with the performance guarantee $\frac{1}{2}$ for 2SSLA problem.*

Proof: Same as the proof in [88] for the two-stage stochastic matching problem. □

Since `GA` solves each stage and each scenario separately, it actually solves the deterministic linear assignment problem $K+1$ times. Thus, one can utilize the `Hungarian Method`(`HM`) [91] to solve each assignment problem. Consequently, given the complexity of `HM`, the outlined algorithm obtains $\frac{1}{2}$–approximate solution of (167)-(170) in $O(Kn^3)$ arithmetic operations.

6.2.2 Greedy Approach of Escoffier et al.

In this section we describe a slightly more advanced greedy approach proposed by Escoffier et al. in [52]. We will refer to this algorithm as `GAE`. The basic idea is that if $\sum_{k=1}^{K} p_k q_{ij}^k > w_{ij}$ for some i and j, then it can not be optimal to assign agent i to job j in the first stage. This result follows

from the observation that any solution to the 2SSLA problem that assigns i to j in the first stage, could be improved by assigning i to j in the second stage across all scenarios. In fact, as we show in Section 6.3, this result is a special case of a more general necessary optimality condition.

`GAE` works as follows. Initially, it replaces all first-stage weights w_{ij} with

$$\hat{w}_{ij} = \max\left\{w_{ij},\ \sum_{k=1}^{K} p_k q_{ij}^k\right\}$$

and obtains the first-stage myopic solution with the updated weights. Then, all agent-job assignments (i, j), i.e., $j = mate[i]$ and $i = mate[j]$, such that $\hat{w}_{ij} = \sum_{k=1}^{K} p_k q_{ij}^k$ are moved to the second stage. Subsequently, `GAE` solves K assignment problems for all agents and jobs moved to the second stage across all scenarios. Denote the resulting solution by Z_1^{GAE} and the above described algorithmic procedure by `GAE-I`.

Next `GAE` compares Z_1^{GAE} with the expected objective function value of the second-stage myopic solutions Z_2^{GA}. The final assignment weight Z^{GAE} corresponds to the better of them and is given by

$$Z^{GAE} = \max\left\{Z_1^{GAE},\ Z_2^{GA}\right\}.$$

Theorem 4 ([52]) *`GAE` is an approximation algorithm with the performance guarantee $\frac{K}{2K-1}$ for 2SSLA.*

Theorem 5 ([52]) *`GAE` is an approximation algorithm with the performance guarantee $\frac{\Delta}{2\Delta-1}$ for 2SSLA, where Δ is the degree of the bipartite graph.*

Both approximation bounds listed above are slightly better than $\frac{1}{2}$ approximation bound of `GA`. Furthermore, it is easy to observe that running time of `GAE` is given by $O(Kn^3)$.

6.3 Necessary Optimality Condition

In this section we describe a necessary optimality condition for the 2SSLA problem. Let $A \subseteq V$ be a subset of agents and $J \subseteq U$ be a subset of jobs such that $|A| = |J|$. Since cardinality of sets A and J are the same, we can consider a two-stage stochastic linear assignment problem on these subsets of agents and jobs. This assignment will be a perfect one as all agents and jobs can be matched. Let $W_1[A, J]$ be the first-stage myopic solution and $W_2[A, J]$ be the expected value of the second-stage myopic solutions over all scenarios. Next we can state the following result:

Proposition 4 (Necessary Optimality Condition) *Let $A \subseteq V$, $J \subseteq U$ and $|A| = |J|$. If $W_1[A, J] < W_2[A, J]$ ($W_1[A, J] > W_2[A, J]$), then no optimal solution of 2SSLA can contain a perfect assignment between agents in A and jobs in J in the first stage (second stage).*

Proof: Consider an optimal solution of an instance of the 2SSLA problem. If $W_1[A, J] < W_2[A, J]$ ($W_1[A, J] > W_2[A, J]$) and the optimal solution contains a perfect assignment between A and J in the first stage (second stage), then moving assignments between A and J to the second stage (first stage) would increase the weight of the optimal solution, which contradicts our assumption that the solution is optimal. □

Unfortunately, this necessary optimality condition is not sufficient for a solution of 2SSLA to be optimal even if we check it for all $O(2^n)$ possible different subsets of V and U. Consider the following simple instance of the 2SSLA problem given in Figure 8.

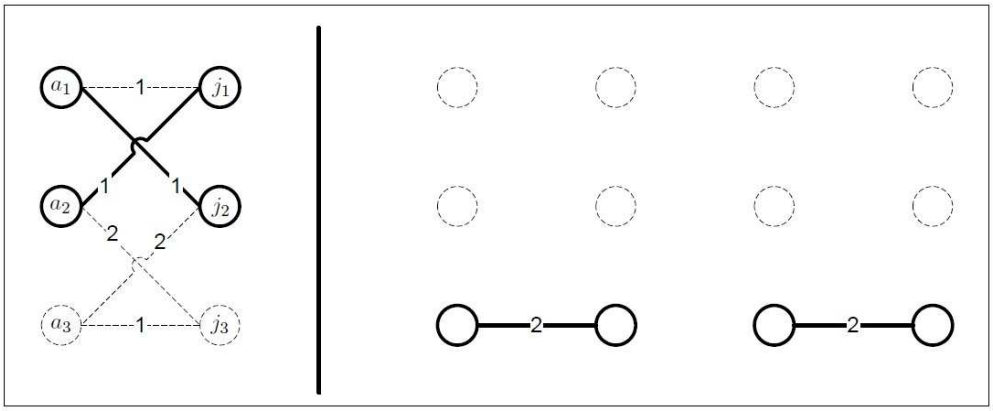


Figure 8: A counterexample to show that the necessary optimality condition given by Proposition 4 is not sufficient for optimality. Only arcs with nonzero weight are shown.

In the first stage, agents a_1 and a_2 are assigned to jobs j_2 and j_1, respectively. In the second stage, agent a_3 is assigned to job j_3 under both scenarios. The total weight of the assignment is 4 units. Notice that this solution does not violate the necessary optimality condition for any subset of agents and jobs. However, optimal assignment has a total weight of 5 units which is achieved by assigning a_1 to j_1, a_2 to j_3, and a_3 to j_2 in the first stage. Therefore, the necessary optimality condition given by Proposition 4 is not sufficient to guarantee optimality.

Nevertheless, Proposition 4 can be actually utilized to construct approximation algorithms. In fact, algorithms `GA` (Section 6.2.1) and `GAE` (Section 6.2.2) are based on the necessary optimality condition for some specific subsets of agents and jobs. Observe that `GA` is the implementation of Proposition 4 when $|A| = |J| = n$. In other words, `GA` verifies the necessary optimality condition only for $A = V$ and $J = U$. If $W_1[V, U]$ (i.e., the weight of the first-stage myopic solution given by `GA-I`) is greater than $W_2[V, U]$ (i.e., the expected weight of the second-stage myopic solutions over all scenarios given by `GA-II`), then it moves assignments between V and U to the second stage. Otherwise all assignments are made in the first stage.

Similarly, `GAE` is the implementation of the necessary optimality condition for all sets A and J such that $|A| = |J| = 1$ and $|A| = |J| = n$. As we have stated in Section 6.2.2, `GAE-I` moves an assignment (i, j) to the second stage if it has a better expected weight in the second stage. Thus, $A = \{i\}$, $J = \{j\}$, $W_1[A, J] = w_{ij}$, and $W_2[A, J] = \sum_{k=1}^{K} p_k q_{ij}^k$. Next, `GAE` compares Z_1^{GAE} with $Z_2^{GA} = W_2[V, U]$ and outputs the better solution, which is equivalent to checking the necessary optimality condition for $|A| = |J| = n$. The only difference here is that instead of $W_1[V, U]$, we use the solution of `GAE-I` and compare it to $W_2[V, U]$.

6.4 Enhanced Greedy Approach

In this section we propose a more generic approximation algorithm, further referred to as `Enhanced Greedy Algorithm` (`EGA`), that attempts to utilize the necessary optimality condition described by Proposition 4 in a more sophisticated manner. `EGA` is based on the `Hungarian Method` as a standard routine to solve all the deterministic linear assignment subproblems. Recall that `HM` works with the dual problem (165)-(166). Furthermore, `EGA` utilizes the dual problem of the LP relaxation of (167)-(169) given by:

$$\min_{\alpha,\beta} \quad \sum_{i=1}^{n}\sum_{k=1}^{K}\alpha_{ik} + \sum_{j=1}^{n}\sum_{k=1}^{K}\beta_{jk} \tag{174}$$

$$\text{s.t.} \quad \sum_{k=1}^{n}(\alpha_{ik}+\beta_{jk}) \geq w_{ij}, \qquad i=1,\ldots,n, \quad j=1,\ldots,n, \tag{175}$$

$$\alpha_{ik}+\beta_{jk} \geq p_k q_{ij}^k, \qquad i=1,\ldots,n, \quad j=1,\ldots,n, \quad k=1,\ldots,K. \tag{176}$$

In the remainder of this chapter we will refer to (175) and (176) as the "first-stage" and "second-stage" dual constraints, respectively, as they correspond to the assignment weights at the first and second stage. Furthermore, for convenience of notation, we let $\tilde{q}_{ij}^k = p_k q_{ij}^k$.

`EGA` has two major steps. The first step (further referred to as `EGA-I`) is to start with first-stage myopic solution and then attempt to improve the objective value by "moving" some of the assignments to the second stage via checking the necessary optimality condition. The second step of the `EGA` (further referred to as `EGA-II`) is to start with the second-stage myopic solutions and then attempt to improve the objective value by "moving" some of the assignments to the first stage. Then similar to `GA` and `GAE`, `EGA` chooses the solution with the better objective value and outputs it as the final solution.

We want to note that there is a strong relationship between the necessary optimality condition given by Proposition 4 and the feasibility of the dual program (175)-(176). The first-stage myopic solution is feasible to the first-stage dual constraints (175) and the second-stage myopic solutions are feasible to the second-stage dual constraints (176). However, the myopic solutions are not necessarily feasible to both (175) and (176) simultaneously. `EGA-I` starts with the first-stage myopic solution feasible to the first-stage dual constraints, and uses the necessary optimality condition to achieve feasibility of the second-stage dual constraints. Specifically, it will be shown that the necessary optimality condition for specific pairs of subsets of agents and jobs corresponds to a second-stage aggregated dual constraint, which is obtained by aggregating the respective subset of the second-stage dual constraints. Similarly, `EGA-II` starts with the second-stage myopic solutions feasible to the second-stage dual constraints, and uses the necessary optimality condition to achieve feasibility of a first-stage aggregated dual constraint, which is obtained by aggregating a subset of the first-stage dual constraints.

6.4.1 Improving the first-stage assignment (EGA-I)

`EGA-I` starts with all agent-job assignments made in the first stage (i.e., the first-stage myopic solution) and then attempts "moving" some of them to the second stage if it is worth doing so. Next we briefly describe the key ideas behind `EGA-I`. The pseudo-code of the approach is given by Algorithm 1.

The initial step of our algorithm is essentially `GAE` described in Section 6.2.2. `EGA-I` applies the necessary optimality condition given by Proposition 4 and starts with the sets of unit cardinality, i.e., $|A| = |J| = 1$. For every agent-job pair (i,j), $A = \{i\}$ and $J = \{j\}$, we know that $W_1[\{i\},\{j\}] = w_{ij}$ and we calculate $W_2[\{i\},\{j\}] = \sum_k \tilde{q}_{ij}^k$. Then, we make the following weight update in the first stage:

$$\hat{w}_{ij} = \max\left\{W_1[\{i\},\{j\}],\, W_2[\{i\},\{j\}]\right\} = \max\left\{w_{ij},\, \sum_k \tilde{q}_{ij}^k\right\}. \tag{177}$$

Algorithm 1: `EGA-I`

Input: n agents, n jobs, K scenarios, w_{ij}, q_{ij}^k, p_k.

Run GAE-I
Reset all the first-stage weights to their original values and remove from consideration all agents and jobs moved to the second stage by GAE-I
Let $\tilde{q}_{ij}^k = p_k q_{ij}^k$; define G_0 and G_k to be the graphs for the first stage and the k^{th} scenario in the second stage, respectively
Run `Hungarian Method` on G_k for all k
Let C include closed subsets of agents and jobs in the obtained second-stage solution
foreach $\{A, J\} \in C$ **do**
    Let $\Delta = W_2[A, J] - (\sum_{i \in A} \alpha_i + \sum_{j \in J} \beta_j)$, where α, β is the dual solution for G_0.
    if $\Delta > 0$ **then**
        Let G be the graph containing only A and J
        Run `Hungarian Method` on G
        Let $\tilde{E}$ be set of edges selected in the resulting assignment
        Let $\tilde{E} = \{e_1, e_2, \dots e_{|A|}\}$ be sorted in non-decreasing order of edge weight
        Set $w_{e_{|A|+1}} = \infty$ and let $w \in \mathbb{R}_+$ and $t \in \mathbb{Z}_+$ satisfy the following condition:
        (1) $t \cdot w - \sum_{r=1}^{t} w_{e_r} = \Delta$
        (2) $w_{e_t} \leq w < w_{e_{t+1}}$
        Set $w_{e_r} = w$ for $r = \overline{1,t}$ in G_0
    else
        $C \leftarrow C \backslash \{A, J\}$
Run `Hungarian Method` on G_0.
begin Reset G_0
    while *there exists* $\{A, J\} \in C$ *not closed in* G_0 **do**
        foreach *Edge weight* w_{ij} *modified in* $\{A, J\}$ **do**
            Reset w
            Perform one iteration of `Hungarian Method`
        $C \leftarrow C \backslash \{A, J\}$
Move all closed subsets in C to the second stage
Redefine G_k to be the subgraph of agents and jobs moved to the second-stage for scenario k.
Run `Hungarian Method` on G_k for all k.
$Z_1^{EGA} = \sum_{i \in G_0} w_{i,mate[i]} + \sum_k \sum_{i \in G_k} \tilde{q}_{i,mate_k[i]}^k$
return Z_1^{EGA}, G_0, *and* $G_k \ \forall \ k$

Then, HM is applied on the resulting graph, all assignments with a modified weight are moved to the second stage, and all edge weights are reset to their original values (lines 1 and 2 of Algorithm 1). We want to emphasize that after this point, EGA-I works only with agents and jobs that are not moved to the second stage in line 1.

Next EGA-I checks the optimality condition for sets with cardinality greater than 1, i.e., $|A| = |J| > 1$. There are $O(2^n)$ possible ways that the subsets A and J can be selected. Furthermore, in order to calculate $W_1[A, J]$ and $W_2[A, J]$, one needs to solve $K + 1$ deterministic linear assignment problems for every subset, where K is the number of scenarios. Thus, it is computationally prohibitive to check all subsets of agents and jobs. Instead, EGA-I considers only a few subsets that are promising and easy to check.

Definition 3 (Closed Subset) *A closed subset in the first stage is a pair of subsets A of agents and J of jobs such that all agent-job assignments remain within these two sets in the first-stage myopic solution, i.e.,*

$$J = \{j \in U \mid j = mate[i] \text{ for some } i \in A\},$$
$$A = \{i \in V \mid i = mate[j] \text{ for some } j \in J\}.$$

A closed subset in the second stage is a pair of subsets A of agents and J of jobs such that all agent-job assignments remain within these two sets across all scenarios in the second-stage myopic solutions, i.e.,

$$J = \{j \in U \mid j = mate_k[i] \text{ for some } i \in A \text{ and some scenario } k\},$$
$$A = \{i \in V \mid i = mate_k[j] \text{ for some } j \in J \text{ and some scenario } k\}.$$

Here we want to provide some details about how EGA-I finds closed subsets. For the first-stage myopic solution, given a subset of agents A, we simply construct J from the mates of agents in A. For the second stage, EGA-I starts constructing a closed subset in the second stage with empty sets A of agents and J of jobs. Given the second-stage myopic solution, we arbitrarily choose an agent and add it to the set of agents A. Then, all jobs that this agent is assigned to across various scenarios are added to the set of jobs J. Notice that an agent may be assigned to the same job in several scenarios. Then, for all jobs that are selected in the previous step, the algorithm updates A to find all agents that jobs from J are assigned to across all scenarios. The algorithm continues in this manner until both sets cease to change, which implies that a closed subset is constructed. The whole process described above is repeated for the remaining agents and jobs until a partition of the set of agents and jobs into a set of closed subsets is obtained.

If A and J correspond to one of the closed subsets in the second stage found by our algorithm, then by the definition of a closed subset, we have that $|A| = |J|$. Moreover, the value of $W_2[A, J]$ is readily available and is calculated using the second-stage myopic solutions. After finding closed subsets in the second stage, EGA-I identifies the ones that satisfy (line 7):

$$\sum_{i \in A} \alpha_i + \sum_{j \in J} \beta_j < W_2[A, J], \tag{178}$$

where α_i and β_j are dual variables associated with the first-stage myopic solution and the respective dual problem (165)-(166). Clearly, closed subsets that satisfy (178) violate the necessary optimality condition since the left-hand side of (178) is an upper bound on $W_1[A, J]$. We use the dual solution due to the fact that the pair (A, J) is not necessarily closed in the first stage and we do not want to solve an assignment problem to find $W_1[A, J]$.

For all subsets that are closed in the second stage and satisfy (178), `EGA-I` updates the first-stage weights (lines 6-6). In contrast to `GAE-I`, updating the first-stage weights properly turns out to be a more difficult task when we have more than one agent-job pair to consider. Our main goal is to update edge weights in the first stage for each closed subset of the second stage in such a way that: (1) the resulting assignment with updated weights should favor the sets to be also closed in the first stage and (2) if the set becomes also closed in the first stage, then the weight of the assignment within this set should be exactly $W_2[A, J]$.

Next we discuss in detail the weight update procedure for a pair (A, J). First, $W_1[A, J]$ is computed by running `HM`. Define $\tilde{E}$ to be the set of all agent-job assignments in the obtained solution. Then we increase the weights w_{ij} in the first stage only for pairs $(i, j) \in \tilde{E}$ according to the following procedure (see also lines 12-16 of Algorithm 1).

- Let $\tilde{E} = \{e_1, \ldots, e_{|A|}\}$ be sorted in non-decreasing order of the edge weights.
- Find $w \in \mathbb{R}_+$ and $t \in \mathbb{Z}_+$, $t \leq |A|$, that satisfy the following conditions:

$$\sum_{r=1}^{t}(w - w_{e_r}) \;=\; t \cdot w - \sum_{r=1}^{t} w_{e_r} \;=\; \Delta, \tag{179}$$

$$w_{e_t} \;\leq w \;<\; w_{e_{t+1}}, \tag{180}$$

  where we assume that $w_{e_{|A|+1}} = +\infty$.
- Set the weight of each $e_r \in \tilde{E}$, $1 \leq r \leq t$, to be w.

Let $\tilde{W}_1[A, J]$ be the weight of the optimal assignment in (A, J) after the weight update procedure described above. It is easy to observe that $\tilde{W}_1[A, J] = W_1[A, J] + \Delta = W_2[A, J]$.

As an example, assume that $\{3, 10, 25\}$ are the weights of the assignment in $\tilde{E}$ (i.e., we have 3 agents and 3 jobs with selected assignment edges that have values 3, 10, and 25). Thus $W_1[A, J] = 38$. Let $W_2[A, J] = 50$. Then, $\Delta = 12$. If we start with $t = 1$, then by (179), we get $w = 15$, which violates (180). Incrementing t, we set $t = 2$ and find $w = 12.5$, which satisfies (180). Thus we set $w_{e_1} = w_{e_2} = 12.5$. Now we have $\tilde{W}_1[A, J] = 12.5 + 12.5 + 25 = 50 = W_2[A, J]$. Simply speaking we increase the weights of the edges with the smallest weights until we have the total increase of Δ.

Subsequently, `EGA-I` finds the first-stage myopic solution with the updated weights (line 19). For every closed subset (A, J), we check whether it remains closed in the first-stage, i.e., $\forall i \in A$ we have that $mate[i] \in J$. If this is not the case, we restore each modified assignment weight of $[A, J]$ and perform one iteration of `HM` to restore optimality. This phase ends when all the remaining closed subsets are also closed in the first stage. Note that the number of weights restored and the number of `HM` iterations performed are bounded by n, the number of agents. Next, the remaining closed sets are moved to the second stage (line 26). Finally, `EGA-I` solves K deterministic assignment problems with all the agents and jobs moved to the second stage (including those moved after line 1) to find the second stage solution, and outputs the resulting assignment.

Lemma 2 *Let Z_1^{EGA} be the weight of the assignment returned by `EGA-I`. Then*

$$Z_1^{EGA} \geq Z_1^{GAE} \geq Z_1^{GA} \;. \tag{181}$$

Proof: It is clear that the solution found after line 1 is exactly the solution found by `GAE-I`. Next consider the first-stage solution and edge weights after line 21. Let $\{A, J\} \in C$ be a closed subset that is not removed from consideration during the procedure between lines 20-21. Any assignment

edge in the first stage that does not belong to closed subsets from C, now has its original value as it was reset in line 23. Any assignment edge weight that belongs to a closed subset $\{A, J\} \in C$ is at least as large as its original value because the weight update mechanism given by (179)-(180) only increases the weights of the edges. By construction, since $\{A, J\} \in C$ is also closed in the first stage, the total weight of the assignments in this subset, $\tilde{W}_1[A, J] (> W_1[A, J])$, is exactly equal to $W_2[A, J]$. It implies that there exists an assignment of A and J at the second stage with the same weight. Hence, the weight of the assignment returned after line 21 is at least as large as weight of the assignment returned after line 1. The necessary result follows. □

6.4.2 Improving the second-stage assignment (EGA-II)

EGA-II starts with all agent-job assignments made in the second stage and then attempts to move some of them to the first stage if it is worth doing so. Due to the lack of control to preserve closed subsets after each weight update in the second stage, this approach is more sensitive to variations in assignments across the scenarios. Using the necessary optimality condition, EGA-II tries to achieve dual feasibility of the model given by (175) and (176). First, it solves the optimization problem given by the objective function (174) and the constraint set (176). Since the constraints are separable for the scenarios, we use HM to solve the assignment problem for each scenario separately (lines 2-3 of Algorithm 2). Therefore, initially all second-stage constraints of the form

$$\alpha_{ik} + \beta_{jk} \geq \tilde{q}^k_{ij}, \qquad \forall i, j, k \tag{182}$$

are satisfied as the obtained assignments are the second-stage myopic solutions. Then the algorithm searches for the pair (i', j') such that

$$(i', j') = \underset{(i,j)}{\operatorname{argmax}} \left\{ w_{ij} - \sum_{k=1}^{K} (\alpha_{ik} + \beta_{jk}) \;\middle|\; w_{ij} - \sum_{k=1}^{K} (\alpha_{ik} + \beta_{jk}) > 0 \right\}.$$

If such pair (i', j') does not exists, then the algorithm stops. Otherwise, it implies that the first-stage dual constraint $\left\{ \sum_{k=1}^{K} (\alpha_{i'k} + \beta_{j'k}) \geq w_{i'j'} \right\}$ is violated. Then consider slack $s_k = \alpha_{i'k} + \beta_{j'k} - \tilde{q}^k_{i'j'}$ for each scenario (line 8). EGA-II updates the weight $\tilde{q}^k_{i'j'}$ according to the following scheme:

- If $\sum_k s_k > 0$:

$$\hat{q}^k_{i'j'} = \tilde{q}^k_{i'j'} + \left(w_{i'j'} - \sum_k \tilde{q}^k_{i'j'} \right) \left(\frac{s_k}{\sum_k s_k} \right) . \qquad \text{(line 11)}$$

- If $\sum_k s_k = 0$:

$$\hat{q}^k_{i'j'} = w_{i'j'} \left(\frac{\tilde{q}^k_{i'j'}}{\sum_k \tilde{q}^k_{i'j'}} \right) . \qquad \text{(line 14)}$$

The intuition behind these update strategies is attempting to keep agent i' assigned to job j' across all scenarios after we update arc weights $q^k_{i'j'}$. Note that after the weight update, we have

$$\sum_{k=1}^{K} \hat{q}^k_{i'j'} = w_{i'j'}.$$

Therefore, any labeling feasible for the second-stage dual constraints $\left\{ \alpha_{i'k} + \beta_{j'k} \geq \hat{q}^k_{i'j'} \; \forall k \right\}$ is also feasible for the respective first-stage dual constraint.

Algorithm 2: `EGA-II`

Input: n agents, n jobs, K scenarios, w_{ij}, q_{ij}^k, p_k.

Let $\tilde{q}_{ij}^k = p_k q_{ij}^k$ and G_k be the graph for the k^{th} scenario in the second-stage
foreach *Scenario* k **do**
    Run `Hungarian Method` on G_k to find an assignment for the k^{th} scenario.
while *There is an* i *and* j *such that* $[w_{ij} > \sum_k (\alpha_{ik} + \beta_{jk})]$ **do**
    Let $(i', j') = \text{argmax}\{w_{ij} - \sum_k (\alpha_{ik} + \beta_{jk})\}$
    Add (i', j') to set $\mathbf{R}$
    begin Update assignments on $G_k \; \forall \; k$
        Let $s_k = \alpha_{i'k} + \beta_{j'k} - \tilde{q}_{i'j'}^k$ `// Slack for the` k^{th} `scenario`
        if $\sum_k s_k > 0$ **then**
            foreach *Scenario* k **do**
                $\tilde{q}_{i'j'}^k := \tilde{q}_{i'j'}^k + \left(w_{i'j'} - \sum_k \tilde{q}_{i'j'}^k\right)\left(\frac{s_k}{\sum_k s_k}\right)$
        else
            foreach *Scenario* k **do**
                $\tilde{q}_{i'j'}^k := w_{i'j'}\left(\frac{\tilde{q}_{i'j'}^k}{\sum_k \tilde{q}_{i'j'}^k}\right)$
        foreach *Scenario* k **do**
            Remove edge $(mate[j'], j)$ from assignment on G_k
            $\beta_{j'k} = \max_i \left\{\tilde{q}_{ij'}^k - \alpha_{ik}\right\}$
            Perform one iteration of *Hungarian Method* on G_k
begin Reset all G_k
    while *There is a pair* $\boldsymbol{(i,j)} \in \boldsymbol{R}$ *such that* $\boldsymbol{mate}[\boldsymbol{i}]$ *is not* $\boldsymbol{j} \; \forall \; k$ **do**
        foreach *Scenario* k **do**
            Reset edge weight $\tilde{q}_{ij}^k$ to its original value
            if $mate[i] = j$ **then**
                Remove assignment (i, j) from G_k
                Perform one iteration of *Hungarian Method* on G_k
        Remove the pair (i, j) from $\mathbf{R}$
Let G_0 be the graph for the first-stage. Move all pairs $(i, j) \in \mathbf{R}$ from G_k, $\forall \; k$, to G_0
Run `Hungarian Method` on G_0
$Z_2^{EGA} = \sum_{i \in G_0} w_{i,mate[i]} + \sum_k \sum_{i \in G_k} \tilde{q}_{i,mate_k[i]}^k$
return Z_2^{EGA}, G_0, *and* $G_k \; \forall \; k$

Then, for each scenario, we remove assignment between job j' and its mate (line 16) and update dual variable $\beta_{j'k}$ (line 17), which is necessary to keep the respective constraints (176) satisfied. Consequently, for each scenario, we lack only one agent-job assignment and the current labeling, i.e., the values of dual variables $(\alpha,\ \beta)$, is feasible for (176). Thus, one iteration of `HM` (line 18) is sufficient to achieve an optimal labeling for updated weights $\hat{q}^k_{i'j'}$ for each scenario. This procedure (lines 4-18) is performed until for every pair (i, j) we have

$$\sum_{k=1}^{K} (\alpha_{ik} + \beta_{jk}) \ \geq \ w_{ij}, \quad \forall i, j\,. \tag{183}$$

Therefore, the following result holds.

Proposition 5 *Let A and J be a pair of subsets of agents and jobs, respectively, such that $|A| = |J|$. Then after line 18 of Algorithm 2, we have that*

$$\sum_{k} \left[\sum_{i \in A} \alpha_{ik} + \sum_{j \in J} \beta_{jk} \right] \geq W_1[A, J]. \tag{184}$$

Proof: Follows directly from (183). □

This result implies that after the above described procedure (lines 4-18), the obtained assignment satisfies the necessary optimality condition for all closed subsets in the second stage. Therefore, contrary to the case for `EGA-I`, `EGA-II` does not check subsets with cardinality strictly greater than one.

Now, consider an assignment $(i, mate_k[i])$ in the k^{th} scenario. If the weight $\hat{q}^k_{i,mate_k[i]}$ is an updated weight, and we have the same assignment in all scenarios, i.e., $(i, mate_k[i])$ is a closed subset in the second stage, we can move this assignment to the first stage without changing the total weight of the current assignments and without affecting other assignments. If we can do this for all such pairs, then we have an optimal solution due to strong duality. However, it may not be the case that each time we have the same assignment $(i, mate[i])$ in all scenarios as the original problem is NP-hard. Therefore, moving this assignment to the first stage will change assignments in scenarios where we do not have the assignment $(i, mate[i])$.

On the other hand, keeping assignment $(i, mate[i])$, for the subsets that are not closed in the second stage indicates that we have an updated arc weight which actually does not exist and we can not find an assignment corresponding to it (i.e., primal infeasible). `EGA-II` decreases the weight of the assignment for such pairs to their original values (line 22) and updates the assignments across all scenarios (lines 23-25) to accommodate this change. Finally, the remaining agent-job pairs with updated weights are moved to the first stage and a separate assignment problem is solved for them (lines 27-28).

Lemma 3 *Let Z_2^{EGA} be the weight of the assignment returned by `EGA-II`. Then*

$$Z_2^{EGA} \ \geq \ Z_2^{GA}\ . \tag{185}$$

Proof: Denote by $(\tilde{\alpha}, \tilde{\beta})$ the dual second-stage myopic solutions. Consider labeling $(\hat{\alpha}, \hat{\beta})$ obtained after line 26 of Algorithm 2. Since the weight updates only increase the arc weights, we have

$$\sum_{k=1}^{K}\sum_{i=1}^{n} \hat{\alpha}_{ik} + \sum_{k=1}^{K}\sum_{j=1}^{n} \hat{\beta}_{jk} \ \geq \ \sum_{k=1}^{K}\sum_{i=1}^{n} \tilde{\alpha}_{ik} + \sum_{k=1}^{K}\sum_{j=1}^{n} \tilde{\beta}_{jk}.$$

Observe that the procedure after line 26 can only potentially improve the weight of the final assignment returned by `EGA-II`, which concludes the proof. □

Theorem 6 *Approximation bounds given for* `GAE` *in Theorems 4 and 5 are valid for* `EGA`*, and its solution satisfies*

$$Z^{EGA} \geq Z^{GAE} \geq Z^{GA}.$$

Proof: The necessary result directly follows from Lemmas 2 and 3. □

Proposition 6 *Time complexity of* `EGA` *is* $O(Kn^4)$.

Proof: First we consider complexity of `GAE-I`. It is easy to observe that lines 1-4 takes $O(Kn^3)$ time. Next step is to find closed subsets. There can be at most n closed subsets and $O(Kn)$ time is required in the worst case to construct each of them. Thus, in total $O(Kn^2)$ time is required to construct all closed subsets. Next, solving the assignment problem for a closed subset (line 10) would take $O(|A||A|^2)$ which is equal to $O(|A|n^2)$. Since $\sum_{\{A,J\}\in C} |A| \leq n$, the total complexity of solving the assignment problems for all closed subsets would take $O(n^3)$ time. The complexity of sorting in line 12 is $O(|A| \lg |A|)$ and similar to our previous argument, total time complexity of sorting would be $O(n \lg n)$. If one starts with $t = 1$ and tries each index sequentially, the total time complexity of the lines 13-16 for all closed subsets would be $O(n)$. Consequently, complexity of updating the weights is of $O(n^3)$. At most n iterations of `HM` is performed between lines 20-21 and thus the resetting procedure is of $O(n^3)$ time complexity. Finally solving $K + 1$ assignment problems (line 28) has a time complexity of $O(Kn^3)$. Consequently, the total time complexity of `EGA-I` is $O(Kn^3)$.

The most time consuming procedure for `EGA-II` is updating weights and assignments (lines 4-4). Since there are n^2 first-stage dual constraints, the outer loop requires $O(n^2)$ operations. The inner loop is to increase cardinality of assignments by 1 across all scenarios, in the worst case. Since each stage of `HM` requires $O(n^2)$ time, the inner loop requires at most $O(Kn^2)$ time. Thus, the time complexity of `EGA-II` is $O(Kn^4)$. This completes the proof. □

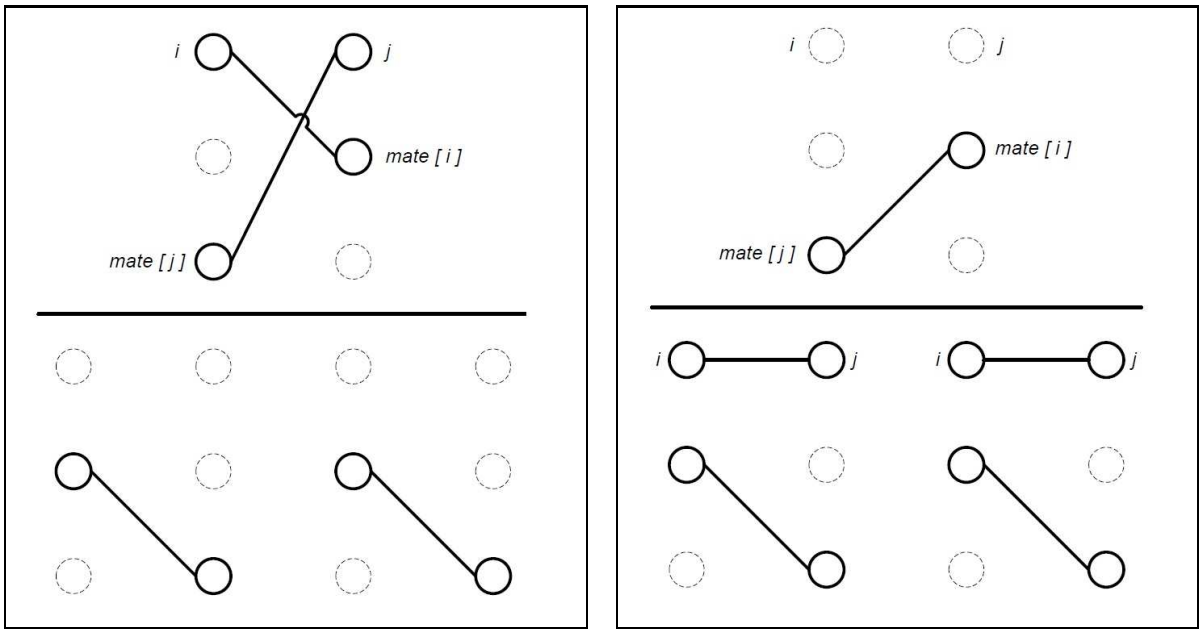
6.4.3 Improving EGA with Local Search

In this section we introduce a greedy local exchange based heuristic that seeks to further improve the results obtained by `EGA-I` and `EGA-II`. Let $(X, Y) = (x, y_1, y_2, \ldots, y_K)$ be a feasible assignment for the 2SSLA problem. Here, we distinguish between an assignment (i, j) and a pair $[i, j]$. The former one indicates that agent i is assigned to job j whereas for the latter one we do not imply any dependence. We say that pair $[i, j]$ belongs to the partial solution X (or Y) if assignments $(i, mate[i])$ and $(mate[j], j)$ are at the first stage (or second stage). Define the following neighborhoods for a given solution (X, Y):

- Neighborhood N_1 for solution (X, Y) is defined to be the set of all solutions obtained by moving any pair $[i, j]$ from X to Y. This implicitly requires $[i, j] \in X$. Thus, to maintain feasibility, if a solution $(\overline{X}, \overline{Y}) \in N_1(X, Y)$, then we have $(mate[j], mate[i]) \in \overline{X}$ and $(i, j) \in \overline{Y}$, assuming that $(\overline{X}, \overline{Y})$ is obtained from (X, Y) with respect to pair $[i, j]$. This exchange process is illustrated in Figure 9.

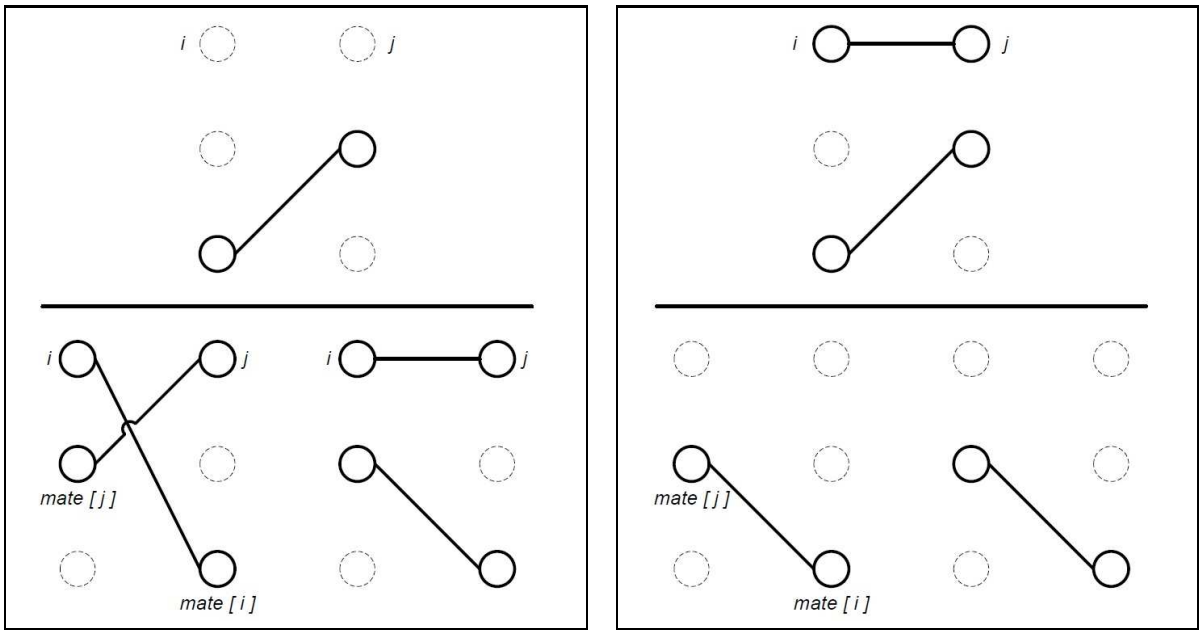
- Neighborhood N_2 for solution (X, Y) is defined to be the set of all solutions obtained by moving any pair $[i, j]$ from Y to X. This implicitly requires $[i, j] \in Y$. Thus, to maintain feasibility, if a solution $(\overline{X}, \overline{Y}) \in N_2(X, Y)$, then we have $(i, j) \in \overline{X}$ and $(mate[j], mate[i]) \in \overline{Y}$. This exchange process is illustrated in Figure 10.

Proposition 7 *The solutions obtained by* `GAE` *and* `EGA-I` *are locally optimal with respect to the neighborhood* N_1.



(a) Initial solution. (b) Final solution obtained.

Figure 9: The neighborhood N_1.



(a) Initial solution. (b) Final solution obtained.

Figure 10: The neighborhood N_2.

Proof: Let (X, Y) be the solution returned after line 1 of Algorithm 1. Consider a pair $[i, j] \in X$. We check whether there is a better solution $(\overline{X}, \overline{Y}) \in N_1(X, Y)$ with respect to $[i, j]$ as follows:

$$\sum_{k=1}^{K} \tilde{q}_{ij}^{k} > w_{i,mate[i]} + w_{mate[j],j} - w_{mate[j],mate[i]}. \tag{186}$$

Assume that $[i,j]$ satisfies (186) and consider the respective dual solution for X. Then we know that the constraints (166) are tight for $(i, mate[i])$ and $(mate[j], j)$:

$$\alpha_i + \beta_{mate[i]} = w_{i,mate[i]}; \quad \alpha_{mate[j]} + \beta_j = w_{mate[j],j}; \quad \text{and} \quad \alpha_{mate[j]} + \beta_{mate[i]} \geq w_{mate[j],mate[i]}.$$

Then, from (186), we have

$$\begin{aligned} \sum_{k=1}^{K} \tilde{q}_{ij}^k &> w_{i,mate[i]} + w_{mate[j],j} - w_{mate[j],mate[i]} \\ &\geq (\alpha_i + \beta_{mate[i]}) + (\alpha_{mate[j]} + \beta_j) - (\alpha_{mate[j]} + \beta_{mate[i]}) \\ &= \alpha_i + \beta_j. \end{aligned}$$

However, this is not possible because the necessary optimality condition for sets with unit cardinality implies that $\alpha_i + \beta_j \geq w_{ij} \geq \sum_{k=1}^{K} \tilde{q}_{ij}^k$ due to the update in (177). Therefore, the necessary result follows. □

Next consider `EGA-II`. Let (X, Y) be the solution obtained by `EGA-II` and let $[i,j] \in Y$. We check whether switching to a solution in the neighborhood N_2 of (X, Y) with respect to $[i,j]$ improves our current solution. Formally, we verify whether

$$w_{ij} > \sum_{k=1}^{K} \left(\tilde{q}_{i,mate[i]}^k + \tilde{q}_{mate[j],j}^k - \tilde{q}_{mate[j],mate[i]}^k \right). \tag{187}$$

If (187) is satisfied, then removing pair $[i,j]$ for all scenarios in the second stage and assigning i to j in the first stage improves our solution. This process is illustrated in Figure 10. However, we may further improve our new solution by running one iteration of Hungarian Method for the first stage and K iterations of Hungarian Method for the second stage. Since each iteration of Hungarian Method requires $O(n^2)$ time, this update requires $O(Kn^2)$ time. Furthermore, at most n pairs may be moved to the first stage which results in a total time complexity of $O(Kn^3)$ for local search after `EGA-II`.

6.4.4 Analytical Observations

Next, we discuss performance of `GA`, `GAE`, and `EGA` on two carefully constructed classes of test instances. Assume that in both classes, we have $2n$ agents, $2n$ jobs, and K scenarios, $K \leq n$. We partition the set of all agents and jobs into two groups: G_1 and G_2, where G_1 contains the first n agents and n jobs and G_2 contains remaining n agents and n jobs. Now we describe two types of instances.

Split Instances: We construct this type of instances as follows:

$$w_{ij} = \begin{cases} 1 & \text{for} \quad (i,j) \in G_1 \ , \\ 2K & \text{for} \quad (i,j) \in G_2 \text{ and } i = j \ , \\ 0 & \text{o/w.} \end{cases}$$

$$q_{ij}^k = \begin{cases} K & \text{for} \quad (i,j) \in G_1 \text{ and } i + k - 1 \equiv j \mod (n) \ , \\ K & \text{for} \quad (i,j) \in G_2 \ , \\ 0 & \text{o/w.} \end{cases}$$

and $p_k = 1/K, \quad k = 1, \ldots, K$. The structure of these instances is illustrated in Figure 11. It is optimal to make assignments for G_2 in the first stage while for G_1 it is optimal to make assignments

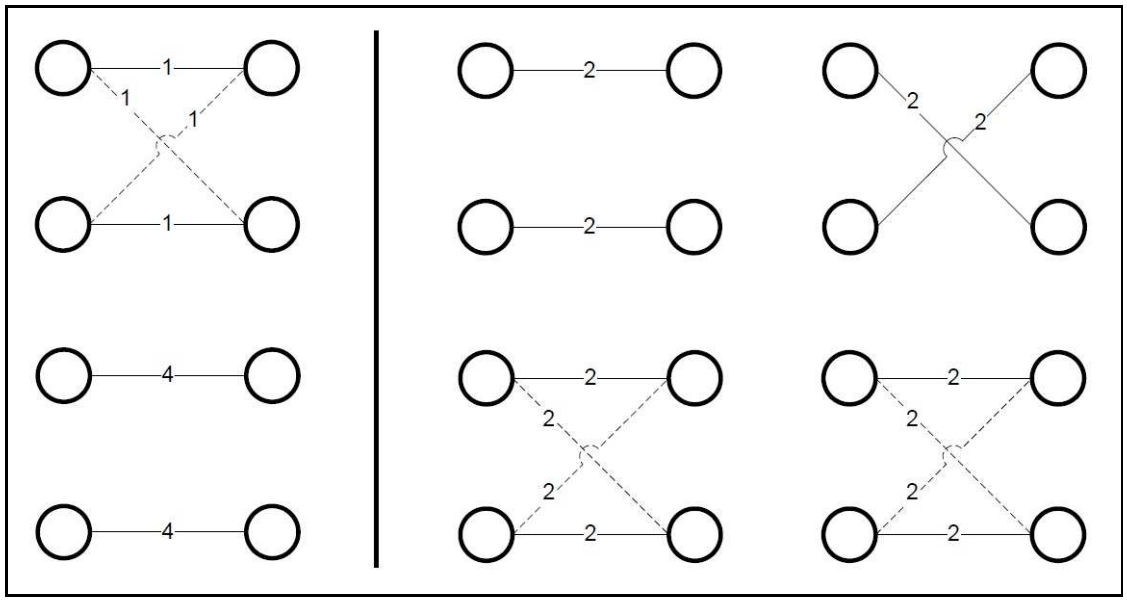


Figure 11: A *split instance* for $K = n = 2$ (only nonzero arcs are shown). Thick lines show first-stage and second-stage myopic solutions.

in the second stage. Therefore, the total weight of the optimal assignment is $3nK$.

Interleaved Instances: We construct this type of instances as follows:

$$w_{ij} = \begin{cases} 1 & \text{for} \quad (i,j) \in G_1 \ , \\ 2K & \text{for} \quad (i,j) \in G_2 \text{ and } i = j \ , \\ 0 & \text{o/w.} \end{cases}$$

$$q_{ij}^k = \begin{cases} K & \text{for} \quad (i,j) \in G_1 \text{ and } i+k-1 \equiv j \mod (n) \ , \\ K & \text{for} \quad i \in G_1,\ j \in G_2, \text{ and } i+k-1 \equiv j-n \mod (n) \ , \\ K & \text{for} \quad i \in G_2,\ j \in G_1, \text{ and } i+k-1 \equiv j \mod (n) \ , \\ 0 & \text{o/w.} \end{cases}$$

and $p_k = 1/K, \quad k = 1, \ldots, K$. The structure of these instances is illustrated in Figure 12. The optimal solution should have all assignments within G_1 in the second stage and all assignments within G_2 in the first stage, with the total weight of $3nK$.

Proposition 8 *The weights of the assignments obtained by* `GA`*,* `GAE`*, and* `EGA` *for split instances are* $n(2K+1)$*,* $n(2K+1)$*, and* $3nK$*, respectively.*

Proof: It is easy to check that `GA` would make all assignments in the first stage and the total weight of this assignment would be $n(2K+1)$. Next we consider `GAE`. From Figure 11, it is clear that the first-stage myopic solution satisfies the necessary optimality condition for the sets of unit cardinality. Thus, solution returned by `GAE` is the same as the solution returned by `GA`.

Finally, we consider `EGA`. Notice that the total weight of the assignments within G_1 in the first stage is n whereas the expected total weight of assignments within G_1 in the second stage is nK. Thus, the first-stage myopic solution violates the necessary optimality condition. Since in the second-stage myopic solution, G_1 will be a closed subset, `EGA-I` would move all assignments to the

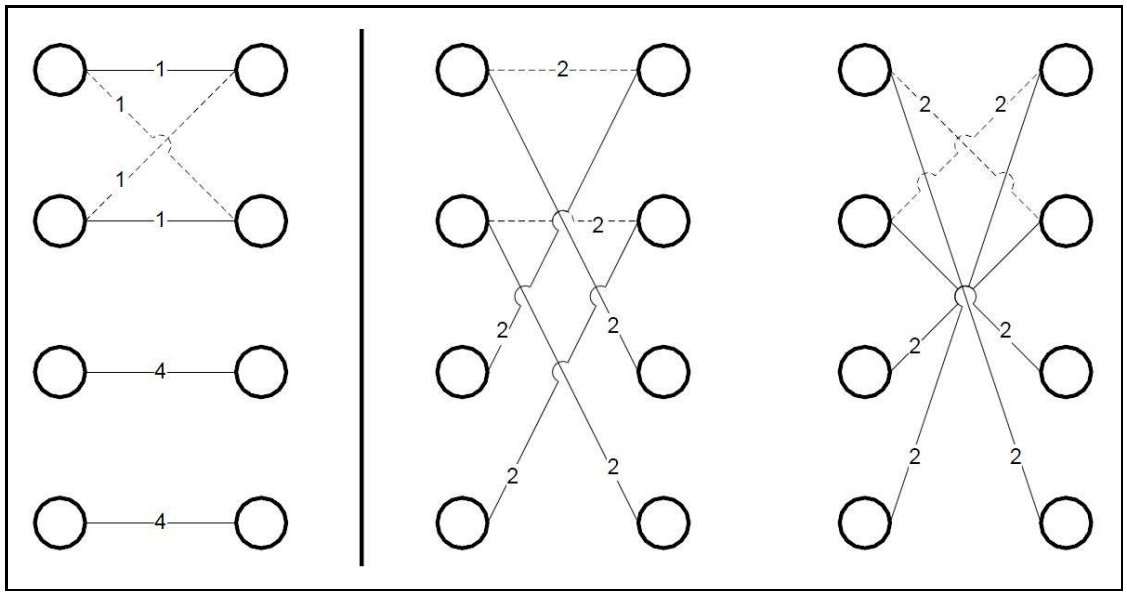


Figure 12: An *interleaved instance* for $K = n = 2$ (only nonzero arcs are shown). Thick lines show first-stage and second-stage myopic solutions.

second stage. Therefore, the weight of the assignment returned by `EGA-I` is at least $3nK$. Since the weight of the optimal assignment for 'split' instances is $3nK$ and `EGA` returns the best of `EGA-I` and `EGA-II`, `EGA` finds the optimal solution. □

Proposition 9 *The weights of the assignments obtained by `GA`, `GAE`, and `EGA` for interleaved instances are $n(2K+1)$, $n(2K+1)$, and $3nK$, respectively.*

Proof: Similar to the proof of Proposition 8. □

We want to conclude this section by emphasizing the importance of the constructed problem instances as they demonstrate that both `GA` and `GAE` can give results significantly away from optimal, while `EGA` returns the optimal solution for both of these classes of test instances.

6.5 Computational Experiments

6.5.1 Setup

Five classes of test instances are used in our computational experiments. The first two classes are similar to the ones used in [52], which allow us to provide an unbiased comparison of `GAE` and `EGA`.

Uncorrelated Instances: All the edge weights are drawn from $N(10, 15)$, the normal distribution with mean 10 and standard deviation 15.

$$w_{ij} \sim N(10, 15), \ \forall i, j.$$
$$q_{ij}^k \sim N(10, 15), \ \forall i, j, k.$$

If the generated weight is negative, then it is set to zero. All scenarios have the same probability.

Correlated Instances: For these instances, the second stage weights are correlated.

$$w_{ij} \sim N(10, 15),\ \forall i, j.$$
$$q_{ij} \sim N(10, 15),\ \forall i, j.$$
$$q_{ij}^k \sim q_{ij} + N(0, 5),\ \forall i, j, k.$$

If the generated weight is negative, then it is set to zero. All scenarios have the same probability.

The intuition behind the next class of test instances is to have the necessary optimality condition satisfied for nearly all unit cardinality sets, but possibly violated for subsets with two agents and two jobs. As it is demonstrated later, both `GA` and `GAE` fail to identify such rather simple weight dependencies.

Pairwise-Correlated Instances: Unlike *correlated* instances, the correlation in these instances is not just on a single agent and job, but on pairs of agents and jobs.

$$w_{ij} = \begin{cases} N(200, 40) & \text{for} \quad i \equiv 0 \mod (3), \\ N(140, 30) & \text{for} \quad i \equiv 1 \mod (3) \text{ and } i \leq j \leq i+1, \\ N(140, 30) & \text{for} \quad i \equiv 2 \mod (3) \text{ and } i-1 \leq j \leq i, \\ N(10, 15) & \text{o/w.} \end{cases}$$

$$q_{ij}^k = \begin{cases} N(200, 40) & \text{for} \quad k \equiv 0 \mod (2),\ i \equiv 1 \mod (3),\ \text{and } j = i, \\ N(200, 40) & \text{for} \quad k \equiv 1 \mod (2),\ i \equiv 1 \mod (3),\ \text{and } j = i+1, \\ N(200, 40) & \text{for} \quad k \equiv 0 \mod (2),\ i \equiv 2 \mod (3),\ \text{and } j = i, \\ N(200, 40) & \text{for} \quad k \equiv 1 \mod (2),\ i \equiv 2 \mod (3),\ \text{and } j = i-1, \\ N(10, 15) & \text{o/w.} \end{cases}$$

Next we provide details for *split-like* and *interleaved-like* instances. Both classes are based on the instances introduced in Section 6.4.4 with modifications that are aimed at "randomizing" their structures. In particular, we add a third class of agents and jobs to the 'split' and 'interleaved' instances with uniformly generated assignment weights. Thus, we have $3n$ agents, $3n$ jobs, and K scenarios, $K \leq n$. Let G_1, G_2, and G_3 be the sets of first, second, and third n agents and jobs, respectively.

Split-like Instances: We let

$$w_{ij} = \begin{cases} U[900/K, 1000/K] & \text{for} \quad (i,j) \in G_1\ , \\ U[500, 1000] & \text{for} \quad (i,j) \in G_2\ , \\ U[100, 1000] & \text{for} \quad (i,j) \in G_3\ , \\ U[2, 10] & \text{o/w.} \end{cases}$$

$$q_{ij}^k = \begin{cases} U[800, 900] & \text{for} \quad (i,j) \in G_1 \text{ and } \ i + k - 1 \equiv j \mod (n) \\ U[100, 500] & \text{for} \quad (i,j) \in G_2\ , \\ U[100, 1000] & \text{for} \quad (i,j) \in G_3\ , \\ U[2, 10] & \text{o/w.} \end{cases},$$

Interleaved-like Instances: We let

$$w_{ij} = \begin{cases} U[900/K, 1000/K] & \text{for} \quad (i,j) \in G_1\ , \\ U[4000, 5000] & \text{for} \quad (i,j) \in G_2 \text{ and } \ i = j\ , \\ U[100, 1000] & \text{for} \quad (i,j) \in G_3\ , \\ U[2, 10] & \text{o/w.} \end{cases}$$

$$q_{ij}^k = \begin{cases} U[800,900] & \text{for} \quad (i,j) \in G_1 \text{ and } \; i+k-1 \equiv j \mod (n) \;, \\ U[1500,2000] & \text{for} \quad i \in G_1, \; j \in G_2, \text{ and } \; i+k-1 \equiv j-n \mod (n) \;, \\ U[1500,2000] & \text{for} \quad i \in G_2, \; j \in G_1, \text{ and } \; i+k-1 \equiv j \mod (n) \;, \\ U[100,1000] & \text{for} \quad (i,j) \in G_3 \;, \\ U[2,10] & \text{o/w.} \end{cases}$$

The probability for each scenario is set to be $1/K$ for each instance from either class.

In our computational experiments, we use CPLEX 12.2 [82]. The algorithms are coded in C++ and implemented on a Windows XP based machine with Intel Xeon 3 GHz processor and 3GB RAM. In our experiments we consider problems with 2, 3, 5, 10, and 20 scenarios and 10, 20, 50, 100, and 200 agents/jobs. For each of these 25 configurations, we conduct 10 replications and report their averages.

6.5.2 Results and Discussion

We report statistics for CPLEX solver as well as `GA`, `GAE`, and `EGA` algorithms. The first two columns, as can be seen in Table 4, are self explanatory. We provide average running time (in seconds) for CPLEX in *Cplex T* column. We have enforced a time limit of 3600 seconds on CPLEX and an average running time of 3600 seconds in this column implies that CPLEX is unable to solve all IP formulations directly to optimality. In such cases, when an integral solution is not available, we use the LP relaxation solution for comparison purpose. The next three columns report the percentage deviation from the CPLEX solution for `GA-I`, `GA-II`, and the overall running time for `GA` in seconds. Next we provide the percentage deviation of the solution returned by `GAE-I` from the CPLEX solution. Since `GAE-II` is the same as `GA-II`, we do not provide such information. However, we report the combined time of `GAE-I` and `GAE-II` under the *GAE T* column. The next three columns are results obtained by `EGA`. Finally, we provide results for `EGA-II` with local search and the time spent for the local search procedure, excluding the time for obtaining `EGA-II` solution prior to the local search procedure. We have marked the best results in bold in all tables.

Results for the *uncorrelated* instances are given in Table 4. It can be observed that CPLEX has difficulty in solving large instances whereas the running time does not exceed 5 seconds for all other algorithms. As expected, `EGA` finds the best results in all cases. `EGA` solution yields a significant improvement over `GA` solution and is reasonably better than `GAE` solution. `EGA-II` (with and without local search) is successful in improving the second-stage myopic solution.

Table 5 summarizes results for the *correlated* instances. Both `EGA-I` and `EGA-II` find nearly optimal solutions and local search further improves the solution of `EGA-II`. `GAE-I` also performs very well on these instances, which is expected due to the structure of these test instances. Note that if an edge has a large weight in one scenario, then it should have a large weight in all scenarios. Thus, most of the time, it is better to make assignments for the agents and jobs incident to such edges in the second stage. Since `GAE` checks the necessary optimality condition for subsets of unit cardinality, its success on these instances is expected.

Results for the *pairwise-correlated* instances are summarized in Table 6. Both `GA` and `GAE` perform rather poorly. On the other hand, `EGA` (especially `EGA-I`) is successful in detecting correlation between pairs of agents and jobs. Since correlation is between pairs of jobs but not larger subsets, local search is also successful and is able to find an optimal solution in almost all cases.

Results for the *split-like* instances are summarized in Table 7. One can observe that CPLEX runs out of time for larger instances. Solutions found by `GA` and `GAE` are poor. In fact, `GA` and `GAE` find almost the same solutions. On the other hand, solution of `EGA-I` is only 1% worse than the CPLEX solution on average. `EGA-II` performs rather poorly for large instances; however the local search procedure is able to eliminate this deviation as shown in *EGA-II LS*. The reason that `EGA-I`

Table 4: Results for uncorrelated instances.

Scenarios	Agents	Cplex T	GA-I	GA-II	GA T	GAE-I	GAE T	EGA-I	EGA-II	EGA T	EGA-II LS	EGA-II LS T
2	**10**	0	9.60	10.24	0	**2.00**	0	**2.00**	6.36	0	3.72	0
	20	0	10.03	9.15	0	5.33	0	5.33	4.36	0	**2.71**	0
	50	0	6.88	6.95	0	4.55	0	4.55	3.88	0	**3.22**	0
	100	1	6.24	6.28	0	4.80	0	4.80	3.23	0	**3.08**	0
	200	10	5.16	5.36	0	4.53	0	4.50	2.70	0	**2.67**	0
3	**10**	0	9.33	8.84	0	**3.48**	0	**3.48**	6.83	0	3.84	0
	20	0	6.87	8.05	0	**3.77**	0	**3.77**	6.15	0	4.96	0
	50	0	6.34	6.65	0	5.65	0	5.65	5.42	0	**4.79**	0
	100	7	5.37	5.10	0	5.03	0	4.83	3.65	0	**3.42**	0
	200	325	4.65	4.74	0	4.55	0	4.21	3.13	0	**3.09**	0
5	**10**	0	12.11	8.82	0	6.85	0	5.68	8.51	0	**2.63**	0
	20	0	7.85	6.74	0	6.63	0	5.71	6.55	0	**5.48**	0
	50	1	5.25	5.53	0	5.10	0	**3.97**	4.66	0	4.40	0
	100	119	4.22	4.10	0	4.20	0	3.51	3.55	0	**3.36**	0
	200	2113	3.98	3.68	0	3.98	0	3.51	3.06	2	**3.02**	0
10	**10**	0	8.46	8.16	0	6.21	0	5.65	7.51	0	**2.53**	0
	20	0	4.42	6.56	0	4.26	0	**3.96**	6.56	0	4.81	0
	50	26	4.63	4.81	0	4.58	0	**3.74**	4.40	0	4.07	0
	100	1536	4.26	3.58	0	4.26	0	**3.31**	3.41	0	3.40	0
	200	3600	3.73	3.49	0	3.73	0	3.40	2.93	3	**2.82**	0
20	**10**	0	4.47	9.87	0	**2.34**	0	**2.34**	9.87	0	4.39	0
	20	0	5.86	4.39	0	5.47	0	4.50	4.39	0	**3.92**	0
	50	96	4.42	4.03	0	4.42	0	**3.31**	4.03	0	3.57	0
	100	2391	3.81	3.80	0	3.81	0	**3.32**	3.71	0	3.71	0
	200	3600	0.43	0.76	1	0.43	1	0.23	0.52	5	**0.20**	0

Table 5: Results for correlated instances.

Scenarios	Agents	Cplex T	GA-I	GA-II	GA T	GAE-I	GAE T	EGA-I	EGA-II	EGA T	EGA-II LS	EGA-II LS T
2	**10**	0	6.89	2.49	0	0.32	0	0.19	0.13	0	**0.09**	0
	20	0	4.65	1.90	0	0.56	0	0.55	0.13	0	**0.12**	0
	50	0	4.98	1.18	0	0.61	0	0.60	0.23	0	**0.20**	0
	100	0	4.80	1.05	0	0.73	0	0.72	0.15	0	**0.13**	0
	200	2	4.76	0.82	0	0.81	0	0.80	0.16	0	**0.15**	0
3	**10**	0	5.65	1.70	0	0.73	0	0.31	0.18	0	**0.15**	0
	20	0	4.86	1.24	0	0.69	0	0.57	0.19	0	**0.17**	0
	50	0	4.78	0.67	0	1.00	0	1.00	0.15	0	**0.15**	0
	100	0	4.54	0.66	0	1.18	0	1.18	0.22	0	**0.21**	0
	200	3	4.64	0.46	0	1.22	0	1.22	0.15	0	**0.14**	0
5	**10**	0	4.57	1.06	0	1.13	0	0.95	0.41	0	**0.27**	0
	20	0	4.20	0.65	0	1.20	0	0.87	0.16	0	**0.15**	0
	50	0	4.27	0.45	0	1.43	0	1.43	0.19	0	**0.15**	0
	100	0	4.60	0.32	0	1.61	0	1.59	0.15	0	**0.14**	0
	200	5	4.48	0.24	0	1.63	0	1.63	0.09	0	**0.09**	0
10	**10**	0	4.26	0.50	0	1.17	0	1.17	0.14	0	**0.13**	0
	20	0	4.12	0.46	0	1.84	0	1.42	0.28	0	**0.20**	0
	50	0	4.05	0.25	0	1.97	0	1.97	0.13	0	**0.11**	0
	100	1	4.23	0.20	0	2.02	0	1.82	0.10	0	**0.08**	0
	200	10	4.22	0.14	0	2.03	0	1.38	0.07	1	**0.07**	0
20	**10**	0	3.44	0.28	0	1.71	0	0.80	0.10	0	**0.07**	0
	20	0	3.90	0.19	0	1.91	0	1.18	0.10	0	**0.07**	0
	50	0	4.07	0.10	0	2.21	0	1.33	0.06	0	**0.05**	0
	100	3	4.00	0.11	0	2.23	0	1.51	0.07	0	**0.06**	0
	200	2526	4.12	0.06	1	2.39	1	0.52	**0.04**	2	**0.04**	0

Table 6: Results for pairwise-correlated instances.

Scenarios	Agents	Cplex T	GA-I	GA-II	GA T	GAE-I	GAE T	EGA-I	EGA-II	EGA T	EGA-II LS	EGA-II LS T
2	**10**	0	11.35	36.30	0	10.91	0	3.43	**0.00**	0	**0.00**	0
	20	0	14.14	31.56	0	13.29	0	3.04	0.44	0	**0.00**	0
	50	0	15.31	28.55	0	14.51	0	4.21	1.16	0	**0.00**	0
	100	0	16.34	27.45	0	15.13	0	4.55	0.42	0	**0.00**	0
	200	1	16.38	26.36	0	14.92	0	4.49	0.63	0	**0.00**	0
3	**10**	0	13.87	35.07	0	9.62	0	3.29	**0.00**	0	**0.00**	0
	20	0	14.65	30.45	0	11.18	0	4.46	3.48	0	**0.00**	0
	50	0	14.66	28.54	0	10.55	0	3.56	4.02	0	**0.00**	0
	100	0	16.08	27.41	0	11.20	0	5.53	2.37	0	**0.00**	0
	200	1	15.66	26.12	0	11.04	0	4.88	0.60	0	**0.00**	0
5	**10**	0	14.57	35.08	0	13.72	0	5.54	**0.00**	0	**0.00**	0
	20	0	14.87	30.51	0	12.86	0	3.75	5.23	0	**0.00**	0
	50	0	15.46	28.32	0	14.22	0	4.12	3.65	0	**0.00**	0
	100	0	15.42	27.57	0	13.88	0	4.19	3.84	0	**0.01**	0
	200	2	15.97	26.31	0	13.83	0	4.25	4.76	0	**0.00**	0
10	**10**	0	14.87	36.65	0	14.71	0	2.32	**0.00**	0	**0.00**	0
	20	0	13.14	30.83	0	12.96	0	0.97	14.76	0	**0.00**	0
	50	0	14.82	28.53	0	14.57	0	2.26	18.68	0	**0.00**	0
	100	1	15.72	27.79	0	15.29	0	2.85	9.22	0	**0.00**	0
	200	5	15.60	26.66	0	15.16	0	2.41	8.70	1	**0.01**	0
20	**10**	0	14.87	35.35	0	14.70	0	2.99	3.92	0	**0.00**	0
	20	0	16.04	30.10	0	15.48	0	4.28	8.80	0	**0.00**	0
	50	0	15.67	28.71	0	15.00	0	3.25	14.11	0	**0.00**	0
	100	2	16.17	27.28	0	15.88	0	2.61	19.12	0	**0.00**	0
	200	2524	15.91	26.18	1	15.50	1	2.57	15.70	3	**0.00**	0

is successful on this class of instances is that it is the only algorithm that can move the whole set G_1 of agents and jobs to the second stage as this set of agents and jobs does not satisfy the necessary optimality condition. It takes only 5 seconds for `EGA` to find a very good solution to the largest problem instance, whereas CPLEX is considerably slower.

Table 8 reports results for the *interleaved-like* instances. This time `EGA-II` is the best of all the algorithms considered. Its solution is only about 0.3% worse than the CPLEX solution. Since `EGA-II` is already very successful, we do not expect much improvement from the local search heuristic. It is also interesting to notice that contrary to the case for `EGA`, the first-stage solution is better than the second-stage solution for `GA` and `GAE`. In terms of time requirements, `EGA` requires at most 3 seconds for all test instances.

In summary, we should point out that the results for the *pairwise-correlated* as well as *split-like* and *interleaved-like* instances indicate that if the weight dependencies between subsets of agents and jobs become more complicated (e.g., in comparison with the *correlated* instances) then `GA` and `GAE` algorithms fail to correctly identify the proper assignments between such subsets of agents and jobs. This is due to the fact that these algorithms check the necessary optimality conditions only for subsets of size 1 and n. On the other hand, `EGA` is specifically designed to locate some of such subsets of agents and jobs, thus significantly improving the quality of the obtained solutions.

6.6 Concluding Remarks

In this chapter we discuss several greedy approximation algorithms for the 2SSLA problem. The proposed necessary optimality condition unifies two recent greedy approximation algorithms from the literature, and aid in the development of a more advanced approach. While `EGA` preserves the approximation guarantees of `GAE`, we are not able to prove whether `EGA` provides a better approximation bound. However, analytical observations and computational results indicate that `EGA` has strictly better results on some rather broad classes of the two-stage stochastic linear assignment problem.

As future research directions, one can use the proposed necessary optimality condition to develop new algorithms with better approximation guarantees, consider the extension to the multi-stage stochastic linear assignment problem, or concentrate on the problems with stochastic right-hand sides, e.g., when multiple jobs can be performed by the same agent. Furthermore, the results of the reported computational experiments indicate that the integrality gap is very small for most of the considered test instances. Thus, development of approximation algorithms based on the LP relaxation of the original integer program is among promising research directions. We are also currently working extending these results for some classes of nonlinear assignment problems.

Table 7: Results for split-like instances.

Scenarios	Agents	Cplex T	GA-I	GA-II	GA T	GAE-I	GAE T	EGA-I	EGA-II	EGA T	EGA-II LS	EGA-II LS T
2	**10**	0	22.01	8.41	0	15.39	0	3.17	1.75	0	**0.85**	0
	20	0	17.23	11.89	0	13.89	0	2.00	3.66	0	**1.03**	0
	50	0	14.41	15.60	0	13.30	0	1.20	6.93	0	**1.04**	0
	100	0	13.63	16.73	0	13.19	0	**0.73**	6.22	0	0.78	0
	200	5	12.73	17.12	0	12.57	0	**0.33**	6.71	0	0.51	0
3	**10**	0	25.43	4.01	0	21.20	0	2.67	0.94	0	**0.41**	0
	20	0	19.99	10.12	0	18.75	0	**1.12**	5.78	0	1.50	0
	50	0	19.04	14.62	0	18.85	0	**0.71**	9.69	0	1.35	0
	100	1	18.75	16.22	0	18.71	0	**0.33**	9.28	0	0.98	0
	200	27	18.32	16.84	0	18.32	0	**0.18**	7.94	0	0.51	0
5	**20**	0	23.33	9.29	0	22.70	0	**0.37**	7.81	0	2.10	0
	50	0	23.27	14.43	0	23.21	0	**0.43**	11.84	0	1.73	0
	100	5	23.38	16.19	0	23.37	0	**0.22**	11.02	0	0.77	0
	200	860	22.91	16.79	0	22.91	0	**0.11**	10.18	1	0.48	0
10	**50**	2	26.98	14.19	0	26.97	0	**0.25**	13.50	0	0.58	0
	100	120	26.77	16.10	0	26.77	0	**0.07**	13.97	0	0.41	0
	200	3600	26.41	16.80	0	26.41	0	**0.11**	12.86	2	0.34	0
20	**100**	2892	28.52	16.20	0	28.52	0	**0.16**	15.84	0	0.45	0
	200	3600	28.18	16.77	1	28.18	1	**0.10**	15.38	5	0.27	0

Table 8: Results for interleaved-like instances.

Scenarios	Agents	Cplex T	GA-I	GA-II	GA T	GAE-I	GAE T	EGA-I	EGA-II	EGA T	EGA-II LS	EGA-II LS T
2	**10**	0	6.55	30.21	0	5.67	0	5.67	0.57	0	**0.30**	0
	20	0	6.44	29.45	0	5.68	0	5.68	0.69	0	**0.50**	0
	50	0	5.92	29.57	0	5.76	0	5.75	0.32	0	**0.28**	0
	100	0	5.77	29.35	0	5.70	0	5.70	0.17	0	**0.17**	0
	200	3	5.62	29.30	0	5.60	0	5.60	0.09	0	**0.09**	0
3	**10**	0	9.66	30.69	0	8.59	0	8.53	0.86	0	**0.22**	0
	20	0	9.01	29.31	0	8.40	0	8.32	0.81	0	**0.62**	0
	50	0	8.57	28.99	0	8.45	0	8.42	0.28	0	**0.26**	0
	100	1	8.34	29.24	0	8.32	0	8.31	0.19	0	**0.18**	0
	200	12	8.20	29.36	0	8.20	0	8.19	**0.09**	0	**0.09**	0
5	**20**	0	10.72	29.29	0	10.36	0	10.36	0.69	0	**0.54**	0
	50	0	10.35	29.51	0	10.32	0	10.24	**0.21**	0	**0.21**	0
	100	4	10.38	29.43	0	10.37	0	10.33	**0.14**	0	**0.14**	0
	200	984	10.24	29.40	0	10.24	0	10.22	**0.06**	0	**0.06**	0
10	**50**	1	12.01	28.75	0	11.99	0	11.90	**0.14**	0	**0.14**	0
	100	65	11.94	29.11	0	11.94	0	11.90	**0.09**	0	**0.09**	0
	200	3600	11.83	29.19	0	11.83	0	11.82	**0.07**	1	**0.07**	0
20	**100**	3273	12.75	28.96	0	12.75	0	12.74	**0.11**	0	**0.11**	0
	200	3600	12.62	29.12	1	12.62	1	12.61	**0.06**	3	**0.06**	0

7 Multiple-Ratio Fractional Programming Problems

This chapter is mostly based on the results from:

- O. Ursulenko, S. Butenko, O.A. Prokopyev, "A Global Optimization Algorithm for Solving the Minimum Multiple Ratio Spanning Tree Problem," Technical report, 2010.

7.1 Introduction

A *fractional combinatorial optimization problem* is defined as follows:

$$\min_{\mathbf{x}\in\mathcal{X}} \frac{f(\mathbf{x})}{g(\mathbf{x})}, \tag{188}$$

where $\mathcal{X} \subseteq \{0,1\}^p$ is a set of certain combinatorial structures, and f and g are real-valued functions defined on $\mathcal{X}$. In addition, it is common to assume that $g(\mathbf{x}) > 0$ for all $\mathbf{x} \in \mathcal{X}$ [129].

One of the classical fractional combinatorial optimization problems is the *minimum ratio spanning tree* (MRST) problem [35], which is defined as follows. Consider a graph $G = (V, E)$ with the set V of n vertices and the set E of m edges. Given a pair of numbers (a_{ij}, b_{ij}) for each edge $(i, j) \in E$, find a spanning tree τ^*, which solves

$$\min_{\tau\in\mathcal{T}} \frac{\sum_{(i,j)\in\tau} a_{ij}}{\sum_{(i,j)\in\tau} b_{ij}}, \tag{189}$$

where $\mathcal{T}$ denotes the set of all spanning trees of G.

The practical applications of this problem include the *minimal cost-reliability ratio spanning tree* problem [36], where the functions in the numerator and the denominator of (189) represent the cost and the reliability of the spanning tree $\tau \in \mathcal{T}$, respectively. This problem can be solved in polynomial time using $O(|E|^{5/2} \log\log |V|)$ arithmetic operations [36, 37, 85]. Closely related classes of problems, where $\mathcal{X}$ is a cycle, a path, or a cut in graph G also admit polynomial time solution approaches [8, 105, 128, 129]. An example of such a problem is the *minimum cost-to-time ratio cycle* problem, also known as the *tramp steamer problem* [8]. A short survey on fractional combinatorial optimization problems and related solution approaches can be found in [129].

Recently, Skiscim and Palocsay [140, 141] have considered a generalization of the MRST problem, where the objective function is given by the sum of two ratios. The resulting *two ratio minimum spanning tree* (TRMST) problem is defined as follows. Consider a graph $G = (V, E)$ with the set V of n vertices and the set E of m edges. Given a set of 4 real positive numbers $(a_{ij}, b_{ij}, c_{ij}, d_{ij})$ for each edge $(i, j) \in E$, find a spanning tree τ^*, which solves

$$\min_{\tau\in\mathcal{T}} \frac{\sum_{(i,j)\in\tau} a_{ij}}{\sum_{(i,j)\in\tau} b_{ij}} + \frac{\sum_{(i,j)\in\tau} c_{ij}}{\sum_{(i,j)\in\tau} d_{ij}}, \tag{190}$$

where $\mathcal{T}$ denotes the set of all spanning trees of G.

A closely related class of combinatorial optimization problems is optimization of the ratio of two linear 0–1 functions:

$$\max_{\mathbf{x}\in\{0,1\}^n} f(\mathbf{x}) = \frac{a_0 + \sum_{i=1}^n a_i x_i}{b_0 + \sum_{i=1}^n b_i x_i}. \tag{191}$$

This problem is a special case of (188) and is usually referred to as a *single-ratio hyperbolic 0-1 programming problem* or *single-ratio fractional 0–1 programming problem* [25]. In a generalization

of this problem one considers the sum of ratios of linear 0–1 functions in the objective:

$$\max_{\mathbf{x}\in\{0,1\}^n} f(\mathbf{x}) = \sum_{r=1}^{k} \frac{a_{r0} + \sum_{i=1}^{n} a_{ri}x_i}{b_{r0} + \sum_{i=1}^{n} b_{ri}x_i}, \tag{192}$$

This problem is known as the *multiple-ratio hyperbolic (fractional) 0-1 programming* problem [127, 145]. A short survey of the literature dealing with the fractional 0–1 programming problems can be found in [125]. Applications of constrained and unconstrained versions of these problems can be found in service systems design [51], facility location [145], query optimization in data bases and information retrieval [77], data mining [29], etc.

Both the minimum ratio spanning tree problem and the single-ratio hyperbolic 0–1 programming problem are polynomially solvable if the denominator is always positive, but become NP-hard if the denominator can take both positive and negative values [77, 126, 140]. On the other hand, their multiple-ratio versions (190) and (192) are NP-hard for two ratios, even if all denominators are always positive [127, 140]. Some other complexity aspects of unconstrained single- and multiple-ratio fractional 0–1 programming problems, including complexity of uniqueness, approximability and local search, are addressed in [126, 127].

Generally speaking, multiple-ratio problems appear in the case of multiple fractional performance metrics that need to be optimized, e.g., a fleet of cargo ships in the tramp steamer problem. Related discussion can be found in [39, 135, 140, 141] and references therein. Analogously with the definition of the multiple-ratio hyperbolic 0–1 programming problem, the *multiple-ratio fractional combinatorial optimization* (MRFCO) problem is defined as

$$\min_{\mathbf{x}\in\mathcal{X}} \sum_{i=1}^{k} \frac{f_r(\mathbf{x})}{g_r(\mathbf{x})}, \tag{193}$$

where $\mathcal{X} \subseteq \{0,1\}^p$ is a set of certain combinatorial structures, and f_r and g_r, $r = 1, \ldots, k$, are real-valued function defined on $\mathcal{X}$.

Another possible application of MRFCO problems is to consider the original single-ratio problem in a *stochastic environment*. Suppose the input data can be described by a discrete number of possible scenarios with corresponding probabilities p_s, which is a typical assumption in stochastic optimization literature [23]. Assume also that the original metric is given by $f_s(\mathbf{x})/g_s(\mathbf{x})$ for each scenario s, $s = 1, \ldots, S$. Then designing combinatorial structure $\mathbf{x} \in \mathcal{X}$ with the minimum expected cost reduces to the MRFCO problem:

$$\min_{\mathbf{x}\in\mathcal{X}} \sum_{s=1}^{S} p_s \frac{f_s(\mathbf{x})}{g_s(\mathbf{x})}. \tag{194}$$

Obviously, the TRMST problem mentioned above is a simple example of the MRFCO problem. Then the multiple-ratio version of the MRST problem is formulated as follows. Let $G = (V, E)$ be a graph with the set V of n vertices and the set E of m edges. Given k pairs of real positive numbers (a^1_{ij}, b^1_{ij}), (a^2_{ij}, b^2_{ij}), …, (a^k_{ij}, b^k_{ij}) for each edge $(i, j) \in E$, the *minimum multiple-ratio spanning tree* (MMRST) problem is to find a spanning tree τ^*, which solves

$$\min_{\tau\in\mathcal{T}} \sum_{r=1}^{k} \frac{\sum_{(i,j)\in\tau} a^r_{ij}}{\sum_{(i,j)\in\tau} b^r_{ij}}, \tag{195}$$

where $\mathcal{T}$ denotes the set of all spanning trees of G. Note that, similarly to [141], we assume that all the coefficients in the pairs (a^1_{ij}, b^1_{ij}), (a^2_{ij}, b^2_{ij}), …, (a^k_{ij}, b^k_{ij}) are positive for each arc $(i, j) \in A$.

The MMRST problem may naturally arise, e.g., in applications where one is looking for an optimal connected configuration in a network that serves k users, $r = 1, \ldots, k$, each of which has its own set of "cost" and "benefit" pairs $\{(a_{ij}^r, b_{ij}^r) : (i,j) \in E\}$ associated with edges in E.

In this chapter we develop a global branch-and-bound approach for solving the MMRST problem based on representing the problem in the *image space* pioneered by Falk and Palocsay for general fractional programming [53, 54]. The suggested algorithm has evolved from the ideas behind the work on two-ratio minimum spanning trees by Skiscim and Palocsay [140].

The image space of the feasible set $\mathcal{T}$ [54] is obtained via introducing a mapping $M : \mathcal{T} \to \mathbb{R}^k$, such that

$$Y = \left\{ M(x) \equiv \left(\frac{a_1^T x}{b_1^T x}, \frac{a_2^T x}{b_2^T x}, \ldots, \frac{a_k^T x}{b_k^T x} \right)^T : \; x \in \mathcal{T} \right\}. \tag{196}$$

The idea of the image space became popular in research related to solving the problems involving the sum of ratios. One reason is that using the image space may significantly reduce the computational burden when $k << n$, which is usually the case in practical applications. This especially applies to our case, since for combinatorial problems like MST the dimension of the original feasible region is often extremely large. Another reason is that, when translated to $\mathbb{R}^k$, the MMRST problem (195) is equivalent to the linear program

$$\begin{array}{rl} \min & e^T y \\ \text{subject to} & y \in conv(M(\mathcal{T})), \end{array} \tag{197}$$

where e denotes the corresponding vector of all ones. Unfortunately, neither we have a description of $conv(M(\mathcal{T}))$ nor there exists a systematic way of generating its facets or extreme points. It may be possible, however, to build a sort of an approximation of $conv(Y)$, which would be accurate enough in the neighborhood of an optimal extreme point y^* to guarantee a solution as close to y^* as needed. This is precisely the idea our algorithm is based on.

The rest of this chapter is organized as follows. Section 7.2 provides a detailed description of the developed global optimization algorithm and the proof of its convergence. The computational results are discussed in Section 7.3. Section 7.4 outlines some directions for future research. For graph theory definitions used in the chapter and for a recent detailed bibliography of fractional programming we refer the reader to [8] and [143], respectively.

7.2 A Global Optimization Approach

This section develops a global optimization approach for the MMRST problem. Its first subsection provides the description of the proposed algorithm and establishes convergence, while the remaining two subsections address two important aspects of the algorithm, namely, solving a subproblem and partitioning the feasible region, respectively.

7.2.1 Description and convergence of the main algorithm

In order to proceed with description of the algorithm, let us introduce some additional notation that we use throughout the rest of the chapter. Recall the definition of the image $Y \triangleq M(\mathcal{T})$ of the feasible set $\mathcal{T}$ of the MMRST problem introduced in (196), where $M : \mathcal{T} \to \mathbb{R}^k$ is given by $M(x) \equiv \left(\frac{a_1^T x}{b_1^T x}, \frac{a_2^T x}{b_2^T x}, \ldots, \frac{a_k^T x}{b_k^T x} \right)^T$ for any $x \in \mathcal{T}$. Given $x \in \mathcal{T}$, we will denote by $M_r(x)$ the r-th ratio $a_r^T x / b_r^T x$. Given $y \in Y$, we will denote by $M^{-1}(y)$ the inverse image $\{x \in \mathcal{T} : M(x) = y\}$. Note that since $\mathcal{T}$ is finite, Y is also finite. For a rectangular region $Q = \{y \in \mathbb{R}^k : l_r \le y_r \le u_r, r = 1, \ldots, k\}$,

we denote the vector $l = (l_1, \ldots, l_k)$ by $L(Q)$, and its r-th component by $L_r(Q)$. Similarly, $U(Q)$ and $U_r(Q)$ denote u and u_r, respectively.

On each step j of our algorithm, an approximation of the portion of $conv(Y)$ containing optimal solution y^* is given by a set of rectangular regions $S^j = \{Q_1^j, \ldots, Q_t^j\}$, such that for all steps j and i, where $j < i$, we have

1. $y^* \in \bigcup_{Q \in S^j} Q$;
2. $\bigcup_{Q \in S^i} Q \subseteq \bigcup_{Q \in S^j} Q$;
3. $\bar{y}^j \in \bigcup_{Q \in S^j} Q$ and $\bar{y}^i \in \bigcup_{Q \in S^i} Q$ are available s.t. $e^T y^* \leq e^T \bar{y}^i \leq e^T \bar{y}^j$.

Note that $e^T L(Q_p^j)$ provides a lower bound on the optimal objective of (197) over the rectangle Q_p^j. Without loss of generality, we can assume that on every step j the rectangular regions in the set S^j are sorted in the nondecreasing order of such lower bounds, i.e., we have $e^T L(Q_p^j) \leq e^T L(Q_q^j)\ \forall p < q$. Then $e^T L(Q_1^j)$ provides the lower bound on (197) available from the approximation S^j. Let us denote this lower bound by $\underline{z}^j$, and the current upper bound, which is the best feasible solution found so far, by $\bar{z}$. S^{j+1} is obtained from S^j by reducing Q_1^j and/or partitioning it into two subregions. The reduction is done similarly to [140], by solving the following subproblem for a particular ratio $r \in \{1, \ldots, k\}$:

$$\min\{y_r : y \in Y \cap Q_1^j, y_s \leq u_s, s = 1, \ldots, k\} \tag{198}$$

where

$$u_s = \max\{y_s : y \in Q_1^j, e^T y \leq \bar{z}\} = \bar{z} - \underline{z}^j + L_s(Q_1^j),\ s = 1, \ldots, k.$$

Let $\tilde{y}$ be an optimal solution to (198). Then Q_1^j may be reduced to

$$P = \{y \in Q_1^j : y_s \leq u_s, s = 1, \ldots, k, s \neq r, y_r \geq \tilde{y}_r\}$$

without discarding any $y \in Y \cap Q_1^j$ that are no worse than the best incumbent solution to (197). If $\tilde{y}_r > L_r(Q_1^j)$, then $\underline{z}^{j+1}$ will be a better lower bound than $\underline{z}^j$. Certainly, P is discarded from further consideration if $e^T L(P) \geq \bar{z}$. Otherwise, $\tilde{y}$ may improve on the current incumbent solution. If $\tilde{y}_r = L_r(Q_1^j)$, then P is partitioned into $P' = \{y \in P : y_h \leq (L_h(P) + \tilde{y}_h)/2\}$ and $P'' = \{y \in P : y_h \geq (L_h(P) + \tilde{y}_h)/2\}$, where

$$h = \arg\max\{|\tilde{y}_s - L_s(P)| : s = 1, \ldots, k\}. \tag{199}$$

Thus $\tilde{y}$ becomes separated from $L(P')$, making the next iteration likely to improve $\underline{z}$. Of course, (198) does not have to be solved when $\tilde{y} \in P$ such that $\tilde{y}_r = L_r(Q_1^j)$ is already known from previous steps of the algorithm.

The formal description of the algorithm is provided in Algorithm 3. Note that $\mathcal{T}$ is passed to the main procedure implicitly through the description of graph G. We discuss how the subproblems (198) are solved along with some other important details in the following subsections.

Theorem 7 *Algorithm 3 converges in a finite number of steps.*

Algorithm 3:

Require: G; $a_r, b_r \in \mathbb{R}^n, r = 1, \ldots, k$; $0 < \epsilon < 1$.

Ensure: $\bar{x}$, an ϵ-optimal solution to (195).

$y^r \leftarrow \arg\min\{y_r : y \in Y\}, r = 1, \ldots, k$;
$\bar{y} \leftarrow \arg\min\{e^T y : y \in \{y^1, \ldots, y^k\}\}$;
$\bar{z} \leftarrow e^T \bar{y}$;
$Q \leftarrow \{y \in \mathbb{R}^k : y_r \geq y_r^r, r = 1, \ldots, k\}$;
$S \leftarrow \{Q\}$;
Choose $r \in \{1, \ldots, k\}$;
repeat
  $Q \leftarrow$ the first set in S;
  Remove Q from S;
  $\underline{z} \leftarrow e^T L(Q)$;
  $P \leftarrow \{y \in Q : y_s \leq \bar{z} - \underline{z} + L_s(Q), s = 1, \ldots, k\}$;
  $\tilde{y} = \arg\min\{y_r : y \in Y \cap P\}$;
  if $e^T \tilde{y} < \bar{z}$ **then**
    $\bar{z} \leftarrow e^T \tilde{y}$;
    $\bar{y} \leftarrow \tilde{y}$;
  end if
  if $\tilde{y}_r = L_r(P)$ **then**
    Choose $h \in \{1, \ldots, k\} (h \neq r)$ that maximizes $\tilde{y}_h - L_h(P)$;
    $P' \leftarrow \{y \in P : y_h \leq (L_h(P) + \tilde{y}_h)/2\}$;
    $P'' \leftarrow \{y \in P : y_h \geq (L_h(P) + \tilde{y}_h)/2\}$;
    if $e^T L(P'') < \bar{z}$ **then**
      $S \leftarrow S \cup \{P''\}$;
    end if
  else
    $P' \leftarrow \{y \in P : y_r \geq \tilde{y}_r\}$;
  end if
  if $e^T L(P') < \bar{z}$ **then**
    $S \leftarrow S \cup \{P'\}$;
  end if
until $\bar{z} - \underline{z} \leq \epsilon \bar{z}$
return $\bar{x} \in M^{-1}(\bar{y})$;

Proof: Let $\bar{z}^j$ denote the value of $\bar{z}$ after j steps of the algorithm. The algorithm terminates when $\bar{z}^j - \underline{z}^j \leq \epsilon \bar{z}$. Suppose that the stopping criterion is not satisfied in a finite number of steps, i.e., the algorithm generates infinite sequences of bounds $\{\underline{z}^j : j \geq 1\}$ and $\{\bar{z}^j : j \geq 1\}$. Since $\{\underline{z}^j : j \geq 1\}$ is monotonously nondecreasing, $\{\bar{z}^j : j \geq 1\}$ is monotonously nonincreasing, and $\underline{z}^j \leq \bar{z}^j$ for any $j \geq 1$, both sequences must converge:

$$\lim_{j \to \infty} \underline{z}^j = \underline{z}^*, \lim_{j \to \infty} \bar{z}^j = \bar{z}^*, \text{ and } \underline{z}^* < \bar{z}^*.$$

The last inequality is strict because of the assumption that we do not have a finite convergence of the algorithm. Consider an arbitrary $\delta > 0$. We will show that there exists $\hat{j}$ such that for any $j \geq \hat{j} : \bar{z}^j - \underline{z}^j \leq \delta$, thus obtaining a contradiction.

Note that finiteness of Y guarantees that $\underline{z}$ improves after a finite number of steps, and the lower bound can increase only due to one of the following two reasons:

1. $\tilde{y}_r > L_r(P)$, in which case the lower bound increases by $y_r - L_r(P)$;
2. P' is not added to S, i.e., $e^T L(P') \geq \bar{z}$, in which case the increase in lower bound value would be $(\tilde{y}_h - L_h(P))/2$.

Due to finiteness of Y it is possible to choose $\delta_1 < \min\{|y'_r - y''_r| : y', y'' \in Y, y'_r \neq y''_r\}$, $\delta_2 \leq \min\{\delta_1, \delta/(2k)\}$, and due to convergence of $\{\underline{z}^j : j \geq 1\}$ there exists $\hat{j}$ such that for any $j \geq \hat{j}$ we have $|\underline{z}^j - \underline{z}^{j+1}| < \delta_2$. On the other hand, if $\underline{z}^j$ in the algorithm increases because $\tilde{y}_r > L_r(P)$ then the increase must be at least δ_2. Thus, if $j \geq \hat{j}$, $\underline{z}^j$ can increase only due to the second reason, and the corresponding increase $(\tilde{y}_h - L_h(P))/2$, where h is defined in line 18 of Algorithm 3, must be less than δ_2. Since h maximizes $\tilde{y}_s - L_s(P), s = 1, \ldots, k$ and y_h is a feasible solution, we have

$$\bar{z}^j - \underline{z}^j \leq e^T \tilde{y} - e^T L(P) \leq k(\tilde{y}_h - L_h(P)) < 2k\delta_2 \leq \delta.$$

Thus, $\bar{z}^* = \underline{z}^*$, and we obtain the contradiction with our assumption that the stopping criterion is not satisfied in a finite number of steps. The finite convergence follows. □

7.2.2 Solving the subproblem

Computational complexity of each iteration of the algorithm described above is defined by the complexity of solving the subproblem (198), therefore it is imperative to solve this problem effectively. Returning to the original variable x, for a rectangular region $Q \in \mathbb{R}^k$ it is formulated as

$$\min \; a_r^T x / b_r^T x \tag{200}$$
$$\text{subject to} \quad x \in \mathcal{T} \cap \mathcal{B}, \tag{200a}$$

where $\mathcal{B}$ defines Q in terms of x:

$$(a_i - U_i(Q) b_i)^T x \leq 0, \quad i = 1, \ldots, k; \tag{200b}$$
$$(L_i(Q) b_i - a_i)^T x \leq 0, \quad i = 1, \ldots, k. \tag{200c}$$

The *constrained minimum ratio spanning tree (CMRST)* problem (200) above is a generalization of the capacity-constrained version of the MST problem. Unfortunately, the latter problem is NP-hard, as shown by Aggarwal et al. [6], even in the case of one constraint. Unless we specifically mention otherwise, $L_r(Q)$ and $U_r(Q)$ in (200b)-(200c) should be assumed $-\infty$ and ∞, respectively, i.e., $k = 2$ refers to a single constraint case of (200).

An effective branch-and-bound approach is suggested in [6] for the MST problem with a single capacity constraint. This approach can be directly extended to our problem when $k = 2$, but because it heavily exploits the ability to obtain solutions that satisfy the capacity constraint, further generalization for $k > 2$ is difficult, if at all possible. In fact, the case of multiple capacity constraints in such classical combinatorial optimization problems as the MST problem and the shortest path problem is not addressed in the literature. Therefore, we have developed our own branch-and-bound approach for solving the general CMRST problem when $k \geq 2$.

Each node $\mathcal{N}$ of our branch-and-bound tree is characterized by the sets $F^0_{\mathcal{N}} = \{e \in E(G) : x_e \text{ is fixed to } 0\}$ and $F^1_{\mathcal{N}} = \{e \in E(G) : x_e \text{ is fixed to } 1\}$. To obtain a good lower bound on the objective in each node, we dualize the constraints (200b) and (200c) and solve the fractional

Lagrangian dual introduced by Gol'stein in [71]. Assume, without loss of generality, that $r = 1$ in (200). Then the respective Lagrangian dual problem is defined as

$$\max_{v \geq 0} \min_{x \in \mathcal{T}} \mathcal{L}(x, v), \tag{201}$$

where $v \in \mathbb{R}^{2k-2}$ and

$$\mathcal{L}(x, v) = \max_{v \geq 0} \min_{x \in \mathcal{T}} \left(\frac{a_1^T x}{b_1^T x} + \sum_{r=2}^{k} \frac{v_{r-1}(a_r - U_r(Q)b_r)^T x}{b_1^T x} + \sum_{r=2}^{k} \frac{v_{k+r-2}(L_r(Q)b_r - a_r)^T x}{b_1^T x} \right). \tag{202}$$

We can solve (201), e.g., via some subgradient optimization algorithm [108]. We employ the Kelley's cutting plane method, since for moderate k it converges fast for piecewise linear functions in practice. Each new cutting plane generated by the Kelley's method corresponds to a tree $x \in \mathcal{T}$, which may or may not be feasible for our CMRST problem. Depending on whether this is the case, they will be used differently in computing a lower bound for the CMRST problem.

We adopt a branching rule similar to the one introduced in [6]. Suppose that solving the dual problem in node $\mathcal{N}$ yields a solution $\hat{x} \in \mathcal{T}$ feasible to (200), and let $e_1, \ldots, e_p$ ($p \leq m-1$) be all edges of the tree corresponding to $\hat{x}$ that are not fixed in node $\mathcal{N}$. We produce p child nodes $\mathcal{M}_1, \ldots, \mathcal{M}_p$ of $\mathcal{N}$ by additionally fixing some of those edges at each child node. Specifically, a child node $\mathcal{M}_j$ ($j = 1, \ldots, p$) is created by additionally fixing j edges out of $e_1, \ldots, e_p$ according to the rule:

$$\begin{aligned} F^0_{\mathcal{M}_j} &= F^0_{\mathcal{N}} \cup \{e_j\}; \\ F^1_{\mathcal{M}_j} &= F^1_{\mathcal{N}} \cup \{e_1,\ e_2,\ \ldots,\ e_{j-1}\}. \end{aligned} \tag{203}$$

Note that if $F^1_{\mathcal{N}}$ is a forest, then it is guaranteed that $F^1_{\mathcal{M}_j}$ is a forest. If several trees feasible to (200) are available in $\mathcal{N}$, then we choose a tree that yields the best objective value.

However, it is possible that the procedure that solves (201) does not encounter a solution feasible to (200). Then we use a different criterion for choosing the edges to branch upon. Let $\bar{v}$ be the optimal solution to (201), and the trees $t_1, \ldots, t_w$ define the hyperplanes that are tangent to the lower epigraph of $\mathcal{L}(v) = \min_{x \in \mathcal{T}} \mathcal{L}(x, v)$ at $\bar{v}$ for some $w \geq 1$. Since $epi\mathcal{L} \subset \mathbb{R}^{2k-1}$, to define $\bar{v}$ uniquely we need at least $2k-1$ hyperplanes. Thus, eliminating $w - 2k + 2$ of the w trees guarantees increase in the optimal value of $\mathcal{L}(v)$. Therefore, the branching should be performed on the edges of those particular trees.

However, it may be difficult to obtain *all* hyperplanes tangent to the lower epigraph of $\mathcal{L}(v)$ at $\bar{v}$. Instead, we branch on the edges of the trees $t_1, \ldots, t_\alpha$, corresponding to the *last* α hyperplanes produced by Kelley's method to approximate the epigraph of $\mathcal{L}(v)$. We choose p' edges occurring most frequently in $t_1, \ldots, t_\alpha$, that are not yet fixed, and produce p' child nodes according to the rule (203). Clearly, because in this case there is no guarantee for a child node $\mathcal{M}_j$ that $F^1_{\mathcal{M}_j}$ is a forest, we have to check this fact, and discard the node if it is not.

Solving (201) via the Kelley's method, in turn, involves solving a sequence of problems of the form

$$\min_{x \in \mathcal{T}} a^T x / b^T x, \tag{204}$$

which is polynomially solvable. We solve (204) using the Dinkelbach's method [50], which, again, involves solving a sequence of MST problems. Consequently, to derive a lower bound for (200), we examine a sequence of spanning trees of G, each of them being a feasible solution to the original MMRST problem (195). Therefore, as we obtain each spanning tree, we examine the value of the original objective that it yields, and improve the upper bound $\bar{z}$ whenever possible.

7.2.3 Partitioning the feasible region

There is a subtle, yet extremely important from the computational perspective, difference between the cases $k = 2$ and $k > 2$. To solve (197) for $k = 2$, one can take advantage of the fact that alternating $r = 1$ and $r = 2$ in

$$\min\ y_r \tag{205}$$
$$\text{subject to}\quad L_i(Q) \le y_i \le U_i(Q), \quad i \ne r \tag{205a}$$
$$y \in Y \tag{205b}$$

virtually rules out the necessity to partition $Q \subset \mathbb{R}^2$. Note also that $L_i(Q)$ may be set to $-\infty$, and thus efficient procedures suggested in [6, 76] for solving (205) with only one side constraint may be utilized. In fact, this is the strategy used in the algorithm by Skiscim and Palocsay [140]. Indeed, suppose that y^1 is the solution to (205) for $r = 1$ and

$$Q = \{y \in \mathbb{R}^2 : (l_1, l_2) \le y \le (u_1, u_2)\}.$$

Now Q is reduced to

$$Q' = \{y \in \mathbb{R}^2 : (y^1_1, l_2) \le y \le (e^T y^1 - l_2, y^1_2)\}.$$

Let y^2 be a solution to (205) for $r = 2$ and $Q = Q'$.

If $e^T y^2 < e^T y^1$, then we can further reduce Q' to

$$Q'' = \{y \in \mathbb{R}^2 : (y^1_1, y^2_2) \le y \le (y^2_1, e^T y^2 - y^1_1)\}$$

thus forcing $y^1 \notin Q''$, since $e^T y^2 < e^T y^1 \Rightarrow e^T y^2 - y^1_1 < y^1_2$. Now that y^1 is separated from $L(Q'')$, we can again solve (205) for $r = 1$ and $Q = Q''$ to further improve the bounds on the optimal objective of (197).

If $e^T y^2 > e^T y^1$, then we can reduce Q' to

$$Q''' = \{y \in \mathbb{R}^2 : (y^1_1, y^2_2) \le y \le (e^T y^1 - y^2_2, y^1_2)\}$$

forcing $y^2 \notin Q'''$, and we can proceed with solving (205) for $r = 2$ and $Q = Q'''$.

The only case when the algorithm cannot proceed is $e^T y^2 = e^T y^1$. In [140] the authors restart the procedure by improving the upper bound, thus reducing Q' and forcing both y^1 and y^2 outside of the resulting rectangle. To achieve this, either a local search is performed, or, if the local search fails to improve an incumbent solution, the procedure is applied recursively to $\{y \in Q' : y_s \le (L_s(Q) + U_s(Q))/2\}$, $s = 1, 2$ until either a better incumbent is found or ϵ-optimality of the current incumbent is proved.

Consider now the case $k > 2$. Let $Q \subset \mathbb{R}^k$ such that

$$L_s(Q) = \min\{y_s \in \mathbb{R} : y \in Y \cap Q\},$$

with y^s being the respective optimal image point, $s = 1, \ldots, k$; and

$$U_s(Q) = \bar{z} - \sum_{s'=1, s' \ne s}^{k} L_{s'}(Q), s = 1, \ldots, k,$$

where $\bar{z} = \min\{e^T y^s, s = 1, \ldots, k\}$.

It is likely that $y^s \in Q'$ for all $s = 1, \ldots, k$ when $k > 2$. This case is analogous to $e^T y^2 = e^T y^1$ for $k = 2$ above, and the procedure by Sciskim and Palocsay [140] outlined above stalls. Improvement

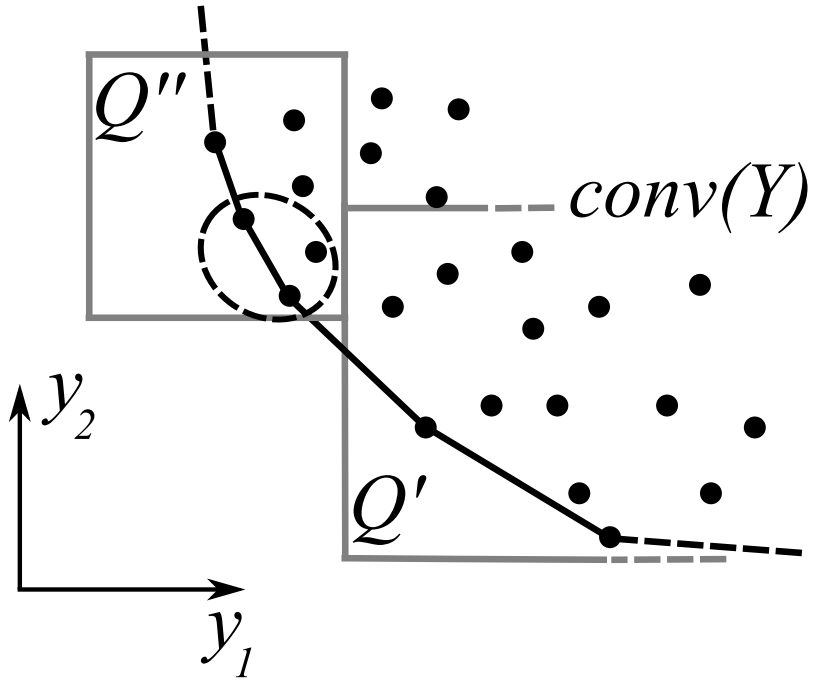


Figure 13: A two-dimensional illustration of condition (206) where $r = 1$.

of the upper bound, unless it is large enough (which cannot be guaranteed), does not restart the procedure. Therefore, for $k > 2$ partitioning Q is a vital step for the algorithm to proceed. Moreover, it turns out that the way the feasible region is partitioned has a significant impact on the computational performance of the algorithm. In particular, we would like to avoid solving (200) with finite $L_r(Q)$. Suppose such subproblem may have to be solved and

$$\exists\, Q', Q'' \in S:\ L_r(Q') \geq U_r(Q''),\ L_s(Q') \leq L_s(Q'') \leq U_s(Q')\ \text{ for some } s \neq r, \tag{206}$$

i.e., the regions Q' and Q'' are positioned as shown on Figure 13 with $r = 1$ and $s = 2$.

As the following proposition implies, the situation described by the condition (206) may lead to extremely inefficient computations. Assume that $\mathcal{B}$ is defined as in (200b)-(200c), $L_r(Q) > -\infty$, and $\mathcal{L}(x, v)$ is the fractional Lagrangian function of (200) obtained via dualizing the constraints defining $\mathcal{B}$. Then the following proposition is true.

Proposition 10 *Let $conv(\mathcal{T}) \cap \mathcal{B} \neq \emptyset$, and $\tilde{x} \in \mathcal{T}$ be such that $a_r^T\tilde{x}/b_r^T\tilde{x} < L_r(Q)$, and all other inequalities that define $\mathcal{B}$ are satisfied in $\tilde{x}$. Then*

$$\sup_{v \geq 0} \inf_{x \in \mathcal{T}} \mathcal{L}(x, v) = L_r(Q).$$

Proof: Take some $\bar{x} \in conv(\mathcal{T}) \cap \mathcal{B}$. Let $\hat{x} = \alpha\bar{x} + (1-\alpha)\tilde{x}$ for some $0 \leq \alpha < 1$ such that $a_r^T\hat{x}/b_r^T\hat{x} = L_r(Q)$. Since $\mathcal{B}$ is convex, such $\hat{x}$ exists. Moreover, it is an optimal solution to the linear relaxation of (200)

$$\min\ a_r^T x / b_r^T x \tag{207}$$

$$\text{subject to}\quad x \in conv(\mathcal{T}) \cap \mathcal{B}. \tag{207a}$$

Indeed, $x \in \mathcal{B}$ enforces the lower bound of $L_r(Q)$ on the objective, and this bound is achieved at $\hat{x}$. On the other hand, it follows from the results in fractional duality [24, 71, 134] that

$$L_r(Q) = \inf\{a_r^T x / b_r^T x : x \in conv(\mathcal{T}) \cap \mathcal{B}\} = \sup_{v \geq 0} \inf\{\mathcal{L}(x, v) : x \in conv(\mathcal{T})\}.$$

Since $\mathcal{L}(x, v)$ is quasiconcave for any fixed $v \geq 0$, it achieves its minimum over $conv(\mathcal{T})$ in some x^* that is a vertex of $conv(\mathcal{T})$. Thus $x^* \in \mathcal{T}$ and

$$\sup_{v \geq 0} \inf\{\mathcal{L}(x, v) : x \in \mathcal{T}\} = \sup_{v \geq 0} \inf\{\mathcal{L}(x, v) : x \in conv(\mathcal{T})\} = L_r(Q).$$

□

Suppose that Q', Q'' are defined as in (206), and $T \subset \mathcal{T}$ is such that

$$\forall x \in T \quad M_r(x) \in Q'' \text{ and } L_s(Q') \leq M_s(x) \leq U_s(Q'), s = 1, \ldots, k, s \neq r.$$

The example displayed on Figure 13 shows $M(T)$ as the image points encircled by a dash line. Then, if the CMRST subproblem (200) with the box constraints defined by Q' is solved via the procedure described in subsection 7.2.2, the lower bound on the optimal value of (200) obtained in *all* nodes of the branch-and-bound tree will be equal to $L_r(Q')$ until at least one edge is excluded *for each* $x \in T$. Not only *this may be a weak bound*; what is worse, it leaves the branching process without direction for choosing the next node to process, thus dramatically increasing run time. *To rule out the possibility of such a situation to occur, we do not alternate the index* r*, but choose it to be fixed* in Algorithm 3. This way, the boxes can only be split by hyperplanes that are parallel to the r-th coordinate axis. Hence, since we start with a single box, the projections of boxes in S^j at any step j of the algorithm onto any coordinate axes other than r-th never overlap.

It should be clear, that the run time of the main algorithm does depend on the choice of r, as it depends on the shape of $conv(Y)$. It may be chosen, for example, by running a few iterations of the algorithm for every $r = 1, \ldots, k$, and choosing the ratio along which the lower bound progresses faster.

7.3 Computational Experiments

7.3.1 Setup

All algorithms are implemented in C++ using Microsoft Visual Studio 2003 environment.We rely on the Boost Graph Library [139] implementation of adjacency list to represent graphs, and the Mersenne Twister MT19937 [103] random number generator implementation from the Boost Random Number Library [104]. The experiments were performed on a computer with Intel® Core™ 2 Duo 3.16 GHz CPU and 3.23 GB of RAM.

The computational experiments were carried out for $k = 1, \ldots, 5$. We considered two types of test instances: complete graphs and connected random graphs. For all graphs the parameters $a_{ij}^1, \ldots, a_{ij}^k$, $b_{ij}^1, \ldots, b_{ij}^k$ for each edge (i, j) of the graph are uncorrelated and follow standard uniform distribution.

7.3.2 Results and Discussion

Tables 9 and 10 summarize the computational performance of the developed global optimization algorithm for complete graph instances and sparse connected random graphs, respectively. Probability of an edge in the latter type of instances is set to 0.1 when $|V| = 20$, and to 0.05 otherwise. We tested a batch of five instances for each reported pair (k, n). The target gap value is set to 1%,

Table 9: Performance on complete graph instances

k	n	steps	run time (sec.)	gap (%)
2	10	12.4	0.1	0.6
	15	20.0	0.6	0.9
	20	26.0	2.6	0.9
	30	37.0	16.8	0.9
	40	28.6	37.0	0.9
	50	39.0	87.0	0.9
	80	65.4	1053.0	1.0
	100	65.8	2941.0	3.9
3	10	51.0	1.1	0.9
	15	104.2	14.6	0.9
	20	190.4	108.0	1.0
	30	495.6	1801.0	1.0
	40	322.0	3465.0	1.9
	50	46.0	3600.0	16.2
4	10	318.0	19.8	0.9
	15	999.4	441.0	1.0
	20	1773.2	3424.0	1.9
	30	198.6	3600.0	17.7
5	10	1534.0	181.0	0.9
	15	4101.2	3600.0	2.9
	20	710.0	3600.0	8.9

and computation time is limited to 1 hour. Average run time as well as average final gap values are reported for each batch. In addition to run time and gap, we report the average number of steps (*i.e.*, subproblems solved) performed by the algorithm in order to reach the final gap value, or until the allotted time expires.

It is evident that performance of the global optimization approach depends on both k and $|\mathcal{T}|$, which have their impact on how difficult it is to build the approximation of $conv(Y)$ that is accurate enough. Furthermore, computational complexity of each iteration also depends on both of these factors. As expected, the results suggest that k primarily affects the number of iterations, and that $|\mathcal{T}|$ mostly affects the time per iteration. An encouraging empirical conclusion can be drawn from Figure 14, which presents convergence of bounds on the optimal objective value for the hardest tested instances. It turns out that an optimal or near-optimal solution is found by the algorithm early, and most of the time is spent on proving quality of an incumbent. This tendency is even more obvious for easier instances. Most likely this should be contributed to a large number of trees examined on each step of the algorithm. Therefore, when the size of the instance does not allow to prove near-optimality in a reasonable time, the suggested algorithm may still be used as a good heuristic.

In general, the developed global optimization procedure shows consistently good performance on the instances with small and medium graph sizes and relatively small number of ratios in the objective function. However, the considered approach needs substantial improvement in order to guarantee a near-optimal solution in reasonable time for large scale instances and large number of ratios in the objective.

Table 10: Performance on sparse random graph instances

k	n	steps	run time (sec.)	gap (%)
2	20	5.8	0.1	0.8
	40	6.0	0.3	0.8
	60	16.6	8.9	0.9
	80	20.6	46.9	0.9
	100	50.6	371.0	0.9
	120	54.0	996.3	1.0
	140	63.6	2677.0	1.2
	160	50.0	3600.0	3.5
3	20	19.4	0.25	0.7
	40	34.0	6.8	0.9
	60	141.2	387.7	0.9
	80	184.6	3000.6	1.1
	100	49.6	3600.0	6.3
4	20	23.4	0.6	0.8
	40	210.0	87.0	0.9
	60	386.2	2610.0	1.7
	80	39.6	3600.0	10.6
5	20	154.0	3.3	1.0
	40	610.0	208.0	1.0
	60	243.6	3600.0	6.0

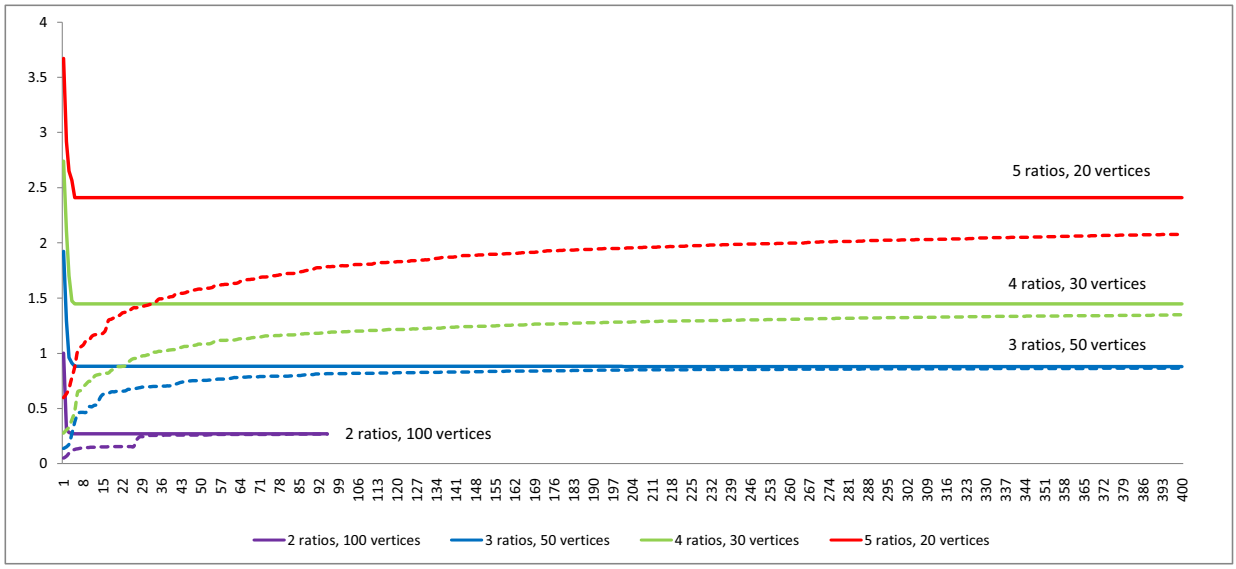


Figure 14: Convergence of lower and upper bounds, represented by dashed and solid lines, respectively, for the hardest complete graph instances.

7.4 Concluding Remarks

We have proposed a global optimization algorithm for solving the MMRST problem. The developed method has evolved from the ideas used by Skiscim and Palocsay [140] to solve a special case of MMRST. The algorithm approximates the convex hull of the images of the spanning trees by subdividing the image space and minimizing a single ratio in the obtained subregions. Extensive computational tests reveal positive sides of the considered approach, as well as its limitations.

Several directions of the further research seem to be apparent from our results. Firstly, the optimization algorithm can be adapted to other combinatorial problems with the considered form of the objective, as long as the linear version of the problem can be solved rather efficiently. Secondly, instead of solving the constrained minimum ratio spanning tree subproblem to optimality on every iteration, it may be beneficial to underestimate its optimal value, provided that we can guarantee convergence of the bounds for such method. Also, if we could find a way to emphasize proving the quality of an incumbent solution at a reasonable computational cost, we would perhaps significantly increase overall performance of the global optimization procedure.

8 Irregular Polyomino Tiling via Integer Programming

This chapter is mostly based on the results from:

- S. Karademir, O.A. Prokopyev, "Irregular Polyomino Tiling via Integer Programming," working paper, 2011.

8.1 Introduction

A phased array antenna is composed of many stationary antenna elements. Phase shift and time delay are the key ideas behind electronically steering the beam in phased array antennas. This technology replaces mechanically steered array antenna designs. Details on theoretical foundations of antennas and review of current antenna technology can be found in [56, 98, 122]. Ideally one would like to have controls (phase shift or time delay depending on the application [99]) at element level but it is too expensive to implement that many controls. Therefore, a group of elements are used to form a 'subarray' which is treated and controlled as an oversized element.

Quantization sidelobes' occur due to periodicity introduced by identical rectangular subarrays used in practice. Simply speaking, a sidelobe is a beam (typically of a smaller magnitude) with a direction other than the main beam direction of the antenna. Such undesired radiation reduces the quality of the pattern generated by antenna. It has been shown that using irregular *polyomino*-shaped subarrays in design of phased array antennas results in a significant reduction of quantization lobes [99, 100]. Figure 15 from [100] illustrates that when polyominoes are used as subarray shapes, sidelobe quantization is reduced to white noise level and the main beam is substantially more significant.

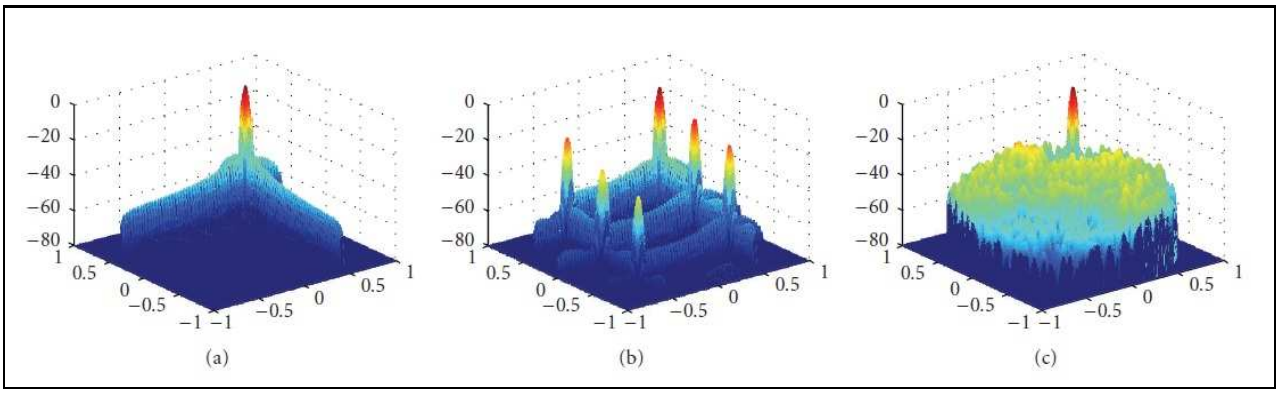


Figure 15: Comparison of 3D sidelobe profiles with time delay at: (a) element level, (b) rectangular subarray level, and (c) polyomino shaped subarray level [100].

In *combinatorial geometry*, a polyomino is a generalization of the *domino* and is created by connecting certain numbers of equal-sized squares [26, 69]. Fig. 16 shows the first five families of polyominoes. The number of different polyominoes, excluding rotations and reflection, increases very fast as n, the number of squares used, grows. There are 2, 108, and 63600 different polyominoes for n equal to 3, 7, and 12, respectively. *Polyomino tiling* is computationally difficult since even deciding whether a rectangular box can be exactly tiled by a set of given rectangles is NP-complete [44]. Enumeration is the only technique that is typically applied in practice. Furthermore, as previously stated, using 'irregular' polyomino tilings results in a major improvement in array antenna performance [99, 100]. Therefore, another challenge that we face is to define a proper

measure of '*irregularity*' of a tiling. We tackle these problems using mixed integer programming approach incorporating the concept of information-theoretic entropy. In particular, 'irregular' polyomino tiling problem can be modeled as an entropy maximizing set partitioning/covering problem, which can be formulated as a nonlinear mixed integer program (MIP).

This chapter briefly reviews our current progress in developing the solution approaches for solving the 'irregular' polyomino tiling problem. In particular, the remainder of this chapter is organized as follows. In Section 8.2 we describe our metric for measuring the irregularity of a tiling. Section 8.3 develops respective mathematical programming formulations. Section 8.4 describes heuristic procedures that can be used to solve large-scale tiling problems. Finally, Section 8.5 motivates our current work.

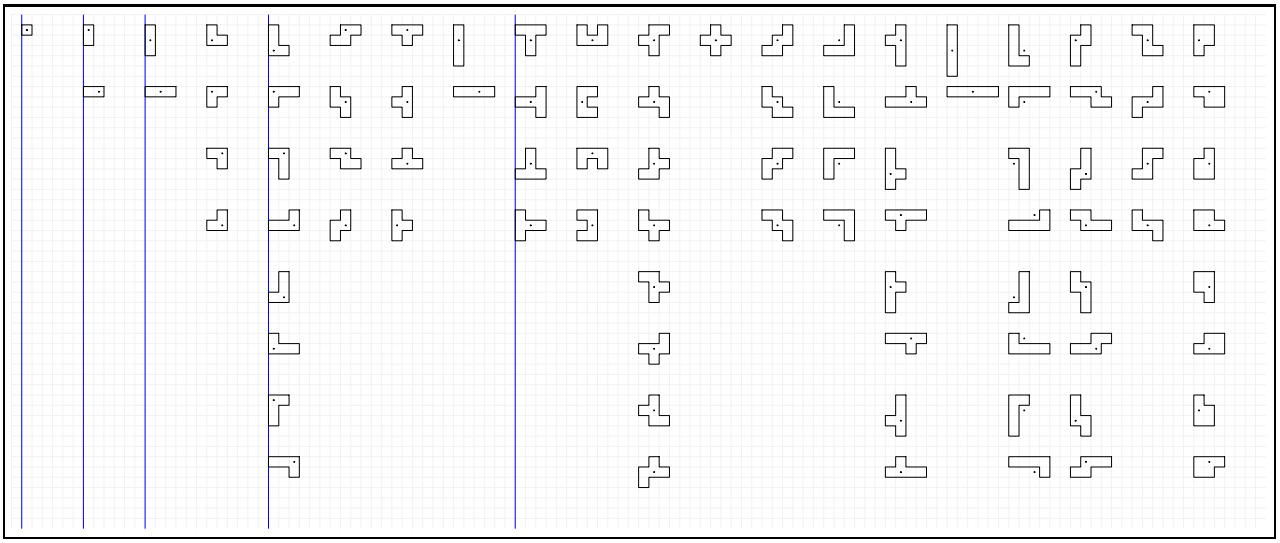


Figure 16: Monomino, domino, triomino, tetromino, and pentomino families.

8.2 Information Theoretic Entropy as a Measure of "Irregularity"

We need a metric that can measure the *irregularity* of a given tiling. Subsequently, it can be incorporated into the mathematical programming models as an objective function. We use the *Information Theoretic Entropy* concept. Though this concept has found applications in optimization and graph theory, we believe our work is the first to use it as a measure of irregularity in the framework of the polyomino tiling problem.

Let X be a discrete random variable with n outcomes. Furthermore, let p_i be the probability of the i^{th} outcome. Then the information theoretic entropy of X, $H(X)$ is defined as

$$H(X) = -\sum_i p_i \lg(p_i) \ .$$

If $p_i = 0$, the value of the term $p_i \log p_i$ is assumed to be zero, which is consistent with $\lim_{p \to 0+} p \log p = 0$. It is easy to verify that $H(X)$ is concave and its maximum occurs when $p_i = \frac{1}{n}$ for all i. This implies that the entropy attains its maximum for the uniform distribution. To link this result to the entropy concept from the statistical mechanics, consider each outcome of X to be a *micro-state* that some system may occupy. The less we know about the micro-state that the system occupies, the larger is the entropy of the system. Then, intuitively if each state may occur with equal probability then our knowledge about the system is minimal; hence, its entropy

is maximized. Conversely, consider a random variable with a single outcome having probability 1. Then we have complete knowledge about the system state; thus, its entropy is 0.

Assume each polyomino is a solid body with some *center of gravity*. Figure 16 illustrates the center of gravity for each polyomino as a black dot inside, or outside, of each polyomino. For simplicity we assume that each center of gravity is located exactly at the center of one of the squares.

Intuitively, we expect that for regular tilings the locations of centers of gravities also have some regular pattern. In particular, most of them should be located in certain rows and/or columns. According to this observation, we define probability distribution for rows and columns of any tiling as follows: row i has probability $\frac{r[i[}{2T}$ and column j has probability $\frac{c[j]}{2T}$, where $r[i]$ and $c[j]$ are the numbers of centers of gravity located in row i and column j, respectively; T is the total number of polyominoes used for the considered tiling. Since each center of polyomino is counted exactly once for rows and exactly once for columns, it easy to verify the validity of this probability distribution. Figure 17 stands as a proof of concept for our argument. Observe how centers of gravity are aligned when entropy is minimized. For the maximization case, one can easily notice that the centers of gravity form almost uniform distribution.

As mentioned above, uniform distribution has the maximum entropy. Therefore,

$$H(X) \leq H(\text{Uniform}) = -\sum_{i=1}^{n} \frac{1}{n} \log\left(\frac{1}{n}\right) = log(n) \ .$$

This theoretical upper bound is extremely important since it allows us to evaluate the performance of the developed approximation or heuristic algorithms. Observe that for the board in Figure 17 we have $\log(40) = 3.6889$, implying that the obtained tiling in the maximization case is close to the optimal solution.

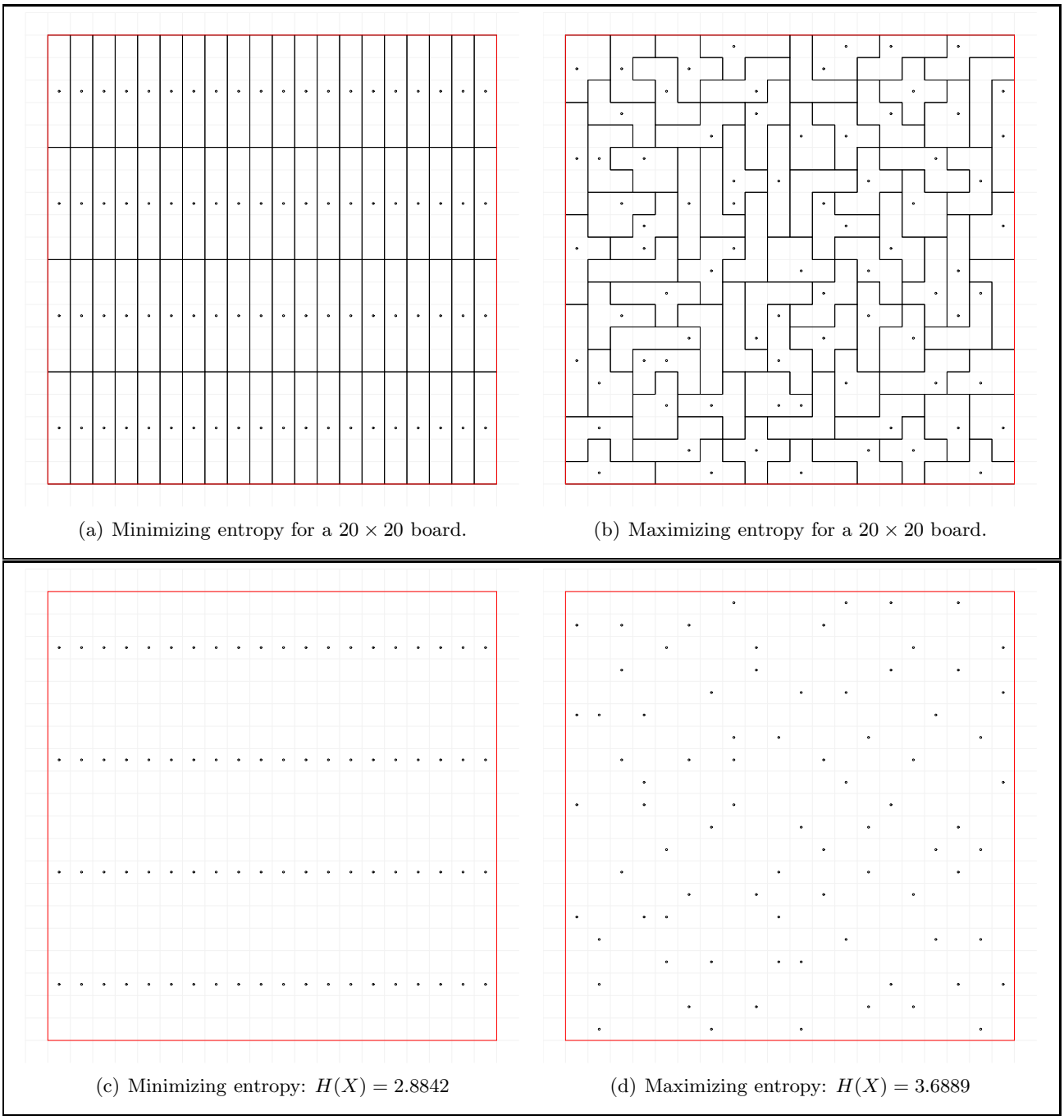
8.3 Mathematical Programming Formulations

8.3.1 Formal Setup

Consider a rectangular set of equal-sized squares (located next to each other) and a polyomino that covers some of the squares. Observe that the polyomino type (e.g., domino) and the location of its north-western corner completely determine squares covered by this polyomino. Hence, it is natural to model our polyomino tiling problem as a set partitioning or a set covering problem: all squares form the ground set and each polyomino with the location of its north-western corner describes some subset of the ground set. Before proceeding with the mathematical programming formulations, we define the notation and provide the formal problem statement.

We will refer to the rectangular set of squares that is required to be tiled as a *board*. We can visualize the board as a matrix since referring to its *rows* and *columns* is natural. Thus, (r, c) denotes the square at the intersection of row r and column c. For a standard board we define two regions: the "frame" and the "center." The *frame* consists of a fixed number of rows and columns that form the boundaries of the board. In general, frame is not required to be tiled exactly. The *center* of the board (inside the frame) consists of squares that need to be covered by exactly one polyomino. If a *perfect* tiling of an $m \times n$ board is required, then the center of the respective board is of size $m \times n$.

To avoid the necessity of mentioning two different parameters for each board, whenever we refer to its dimensions $m \times n$, the values of m and n define the size of the center of the board. Furthermore, we will say that a given tiling is *perfect* if none of the squares of the board frame are covered. Finally, we can state our problem more formally:



(a) Minimizing entropy for a 20×20 board.

(b) Maximizing entropy for a 20×20 board.

(c) Minimizing entropy: $H(X) = 2.8842$

(d) Maximizing entropy: $H(X) = 3.6889$

Figure 17: Validating entropy concept on pentomino family.

INPUT: A set of polyominoes P with $|P| = K$ members, $m \times n$ board B, and the type of tiling: perfect or imperfect.
OUTPUT: The most irregular (according to the metric defined above) tiling of board B.

Next, we develop mathematical programming formulations for this problem.

8.3.2 Nonlinear Set Partitioning Formulation

In the remainder of this chapter we assume that each polyomino covers exactly the same number of squares of the board (i.e., all polyominos have the same area size). In general this assumption can be easily relaxed.

Next we introduce the following notation.

- Define 0–1 variable $x^k_{pq} = 1$ iff the north-western corner of a polyomino of type k is located at (p, q). For example, a monomino located at (p, q) would cover a single square $\{(p, q)\}$; similarly, a vertical domino located at (p, q) would cover $\{(p, q),\ (p + 1, q)\}$.
- Let r_i and c_j be continuous decision variables that denote the number of centers of gravity in row i and column j, respectively. In fact, r and c are auxiliary variables that are completely determined by the values of x variables.
- Let I_{ij} be the set of all triples (k, p, q) such that if a polyomino of type k located at (p, q) then it covers (i, j).
- Let R_i be the set of triples (k, p, q) such that if a polyomino of type k is located at (p, q) then its center of gravity is located in row i.
- Let C_j be the set of triples (k, p, q) such that if a polyomino of type k is located at (p, q) then its center of gravity is located in column j.
- Let T be the total number of polyominoes used for tiling the board. In fact, due to our assumption T is constant since its value can be easily derived from the area to be covered and the size of the used polyomino family. For instance, an exact tiling of 8×8 board using tetromino family requires $T = 8 * 8/4 = 16$ polyominoes.

Given this notation, we provide the following nonlinear mixed integer programming (MIP) formulation of the exact tiling problem:

$$
\begin{array}{lll}
\mathrm{P}_{NL}\text{: } \min & \displaystyle\sum_i \frac{r_i}{2T} \lg\left(\frac{r_i}{2T}\right) + \sum_j \frac{c_j}{2T} \lg\left(\frac{c_j}{2T}\right) & \\
\text{s.to} & \displaystyle\sum_{(kpq)\in I_{ij}} x^k_{pq} \;=\; 1 & \forall\ i, j \\
& \displaystyle r_i = \sum_{(kpq)\in R_i} x^k_{pq}, & \forall\ i \\
& \displaystyle c_j = \sum_{(kpq)\in C_j} x^k_{pq} & \forall\ j \\
& x^k_{pq} \in \{0, 1\},\ r_i,\ c_j \geq 0 & \forall\ i, j, p, q, k.
\end{array}
$$

8.3.3 Linear Set Partitioning Formulation

We use *value disjunctions* to reformulate P_{NL} as a linear MIP that can be tackled using any standard mixed integer programming solver (e.g., CPLEX).

- Define 0–1 variable $r_{it} = 1$ iff there are exactly t centers of gravity in row i.
- Similarly, define 0–1 variable $c_{jt} = 1$ iff there are exactly t centers of gravity in column j.

Then we obtain the following linear MIP denoted as P_L.

$$
\begin{array}{lll}
\mathrm{P}_L: \quad \min & \displaystyle\sum_{i=1}^{m}\sum_{t=1}^{T}\left(\frac{t}{2T}\lg\frac{t}{2T}\right) r_{it} + \sum_{j=1}^{n}\sum_{t=1}^{T}\left(\frac{t}{2T}\lg\frac{t}{2T}\right) c_{jt} & \\
\text{s.to} & \displaystyle\sum_{(kpq)\in I_{ij}} x^k_{pq} \;=\; 1 & \forall\ i,\ j \\
& \displaystyle\sum_{t=1}^{T} t\ r_{it} = \sum_{(kpq)\in R_i} x^k_{pq} & \forall\ i \\
& \displaystyle\sum_{t=1}^{T} t\ c_{jt} = \sum_{(kpq)\in C_j} x^k_{pq} & \forall\ j \\
& \displaystyle\sum_{t=0}^{T} r_{it} = 1 & \forall\ i \\
& \displaystyle\sum_{t=0}^{T} c_{jt} = 1 & \forall\ j \\
& x^k_{pq},\ r_{it},\ c_{jt} \in \{0,1\} & \forall\ i,j,p,q,k,t.
\end{array}
$$

If there are K polyomino types used to tile an $m \times n$ board, we would have $O(Kmn)$ variables. Note also that both formulations can be easily modified to handle imperfect tilings. Figure 18 provides a perfect tiling of 10×10 board and an imperfect tiling of 9×11 board (there is no perfect tiling for this board size) using tetromino family.

Proposition 11 *Variables r_{it} and c_{jt} in P_L can be relaxed to be nonnegative, i.e., formulation P_L is locally ideal.*

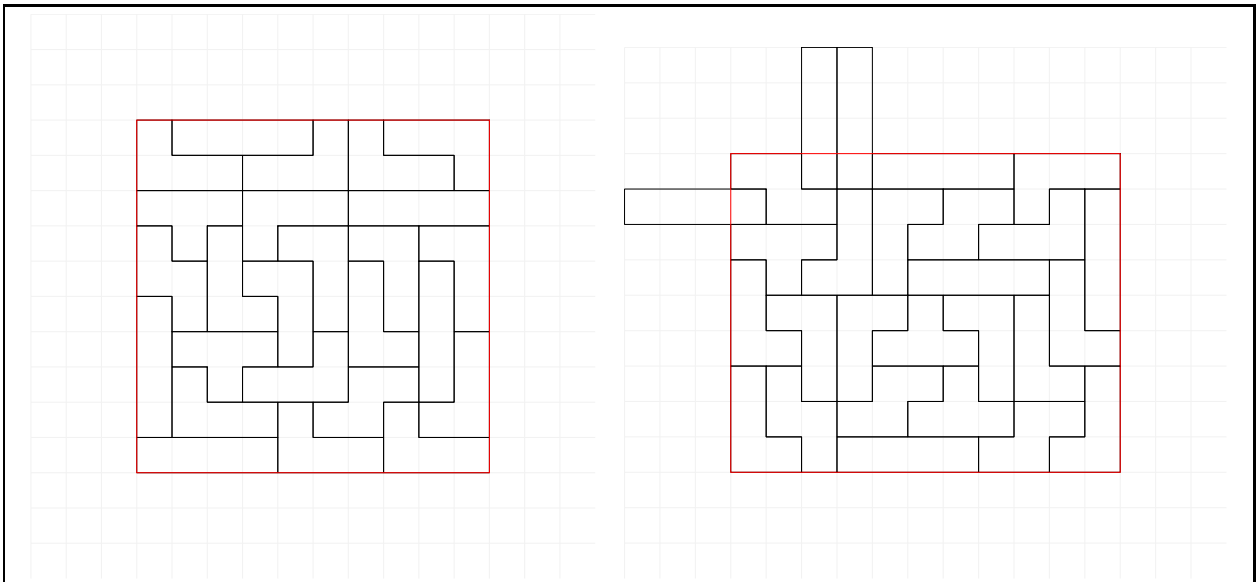
Proof: Consider the second derivative of the function $\phi(x) = x\lg(x)$ which is continuous on $(0, 1]$:

$$\phi''(x) = \frac{1}{x} > 0\ .$$

Hence, $\phi(x)$ is *strictly convex*. Now consider the strictly convex function $\phi_T(x) = \frac{x}{T}\lg(\frac{x}{T})$ defined on $x \in (0, T]$. Using constraints of the model ($\sum t\ r_t = \Delta \le T$ and $\sum r_t = 1$) and the Jensen's Inequality,

$$
\begin{aligned}
\frac{\Delta}{T}\lg(\frac{\Delta}{T})\cdot 1 \;=\; \phi_T(\Delta) \;&=\; \phi_T\left(\frac{\sum t\ r_t}{\sum r_t}\right) \\
&<\; \frac{\sum r_t\phi_T(t)}{\sum r_t} \\
&=\; \sum r_t\phi_T(t) \;=\; \sum \frac{t}{T}\lg\left(\frac{t}{T}\right) r_t
\end{aligned}
$$

whenever more than one r_t is nonzero. Thus, the lower bound on the left hand side is attained only when $r_\Delta = 1$. In the derivation above, we drop indices i and j for simplicity. □



(a) Perfect tiling of 10×10 board.

(b) Imperfect tiling of 9×11 board.

Figure 18: Optimal tilings using the tetromino family.

8.4 Heuristics

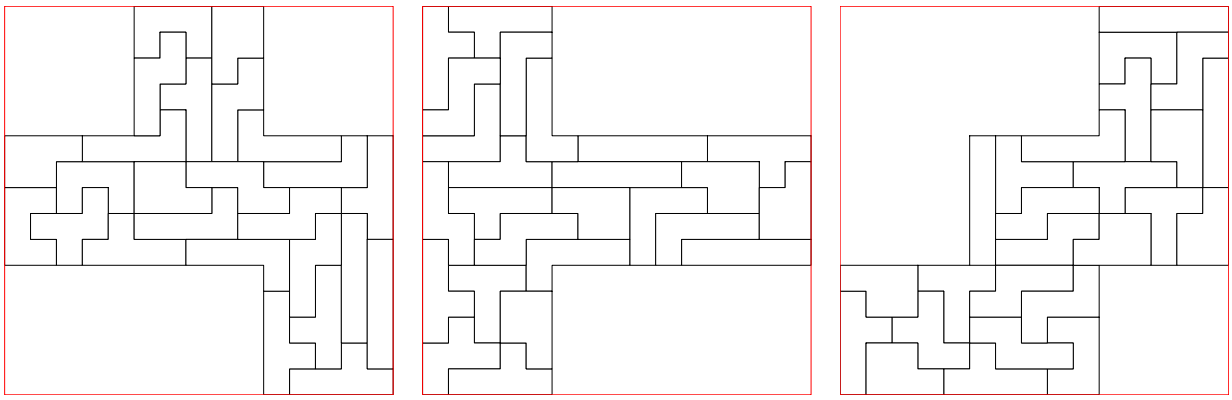
Since the obtained linear MIP can be used to solve exactly tiling problems of rather small sizes, we also develop a heuristic approach that can be applied to tile large-scale boards. In the algorithms described below solving P_L for small board sizes serves as a subprocedure. In order to simplify the description of the developed algorithms, we illustrate them with some simple examples.

8.4.1 Heuristic Procedure: Zoom-in

Consider a board that consists of a single square. One could enlarge the board by replacing the square by a 2×2 board and tile it using 4 unit squares. In general, given board B (not necessarily rectangular) we can enlarge it replacing each unit square of B by another rectangular board of size $a \times b$. We refer to this procedure as "zoom-in" with level $z = (a, b)$, or $z = (a \times b)$. If B is rectangular of size $m \times n$, then we obtain a board of size $m \cdot a \times n \cdot b$.

Consider a set of polyominoes P. Assume that there exists a "zoom" level (a, b) (obtained replacing each square with an $a \times b$ rectangular board) such that each polyomino in P can be tiled exactly with polyominoes from P. Figure 19 illustrates this concept providing several pentominoes tiled with other pentominoes at (5×5) "zoom" level. The respective tilings are obtained solving formulation P_L exactly.

For any given initial tiling of size $m \times n$, the "zoom-in" procedure can be used to generate tilings of $m \cdot a^x \times n \cdot b^x$ for any positive integer x. Procedure Zoom-in below provides the pseudo-code of the algorithm.



(a) 'F' rotated 90°. (b) 'T' rotated 270°. (c) 'W'.

Figure 19: Several pentominoes at (5×5) "zoom" level.

Procedure Zoom-in

Input: $m \times n$ board B tiled exactly (e.g., via solving P_L) by some set of polyominoes P_1, required "zoom" level $z = (a, b)$, and exact tilings (e.g., also obtained via solving P_L) of each polyomino in P_1 with some other set of polyominoes P_2 (possibly same as P_1) at the "zoom" level z.

```
1 begin Zoom-in
2     Replace each unit square of B with an a × b board.
3     Denote the final enlarged board be B_z.
4     Denote by p_z in B_z a polyomino p ∈ P_1 enlarged by z after the "zoom-in".
5     foreach p_z in B_z do
6         Let p_z^* be the perfect tiling of p_z using P_2.
7         Replace p_z with p_z^* in B_z.
8 return Final tiling.
```

8.4.2 Heuristic Procedure: Magnify

Consider a set of polyominoes P and some $m \times n$ board tiled exactly solving formulation P_L. If one requires tilings of other board sizes, procedure Magnify is designed to serve this aim. Unfortunately, the resulting tiling may be not necessarily perfect.

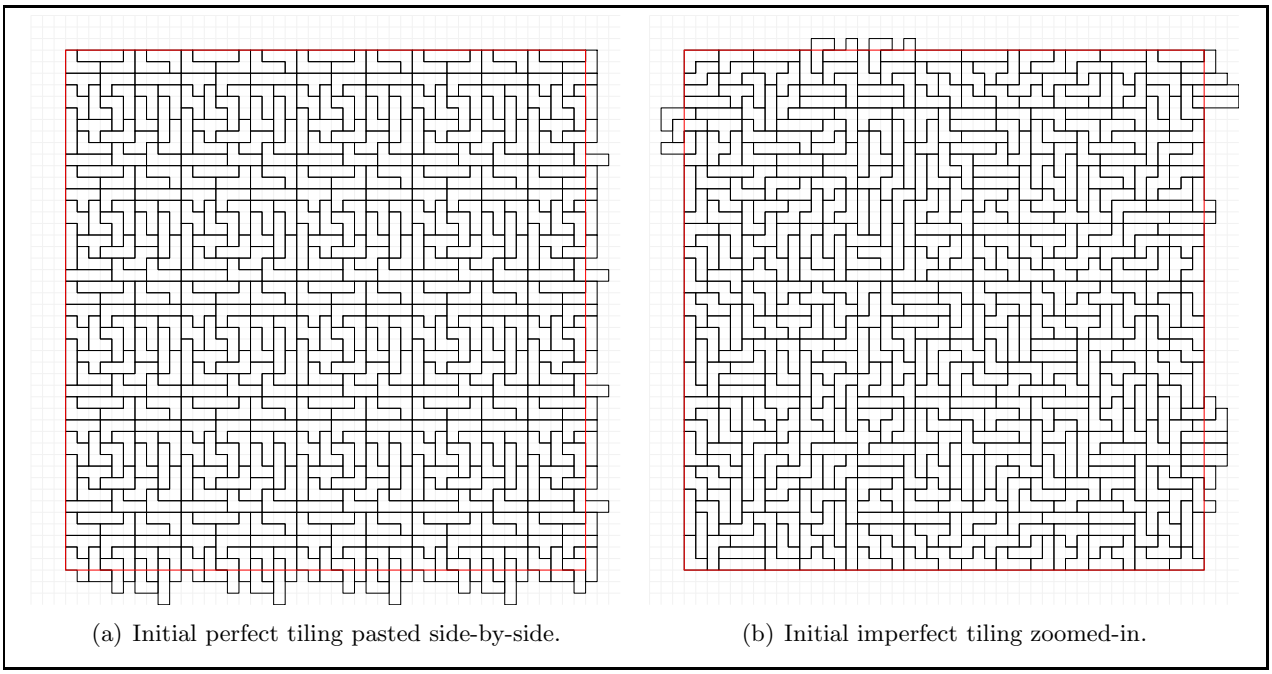
Assume that we are required to tile a board of size $M \times N$, where $M >> m$ and $N >> n$. Suppose there exists a "zoom" level (a, b) (i.e., obtained replacing each square with an $a \times b$ rectangular board) such that each polyomino in P can be tiled exactly with polyominoes from P. One can simply first enlarge the original tiling until $m \cdot a^x \geq M$ and $n \cdot b^x \geq N$, and then drop the polyominoes that are completely outside the board of the required size. Otherwise, we can simply paste the copies of the initial $m \times n$ tiling side by side until we cover the board above the required size and then drop the polyominoes that are completely outside.

Figure 20 shows the solutions of Figure 18 pasted side-by-side and "zoomed-in." Initially, the tiling of Figure 18(a) is enlarged to 50×50 and tiling of Figure 18(b) is enlarged to 45×55. Both tilings then reduced to 45×45.

Procedure Magnify

Input: (1) Required tiling dimensions ($M \times N$). (2) Either (2a) a small (not necessarily perfect) tiling of size $m \times n$ together with the perfect tiling of each polyomino at some "zoom" level ($a \times b$) exists or (2b) a small perfect tiling of size ($m \times n$) exists.

if *Tiling of each polyomino at "zoom" level* $a \times b$ *exists* **then**
  Let x be the smallest integer satisfying $M \leq m \cdot a^x$ and $N \leq n \cdot b^x$
  "Zoom-in" x times (Alg. Zoom-in)
  Draw a rectangle of size $M \times N$ inside the obtained tiling
  Drop any polyomino that lies completely outside the $M \times N$ rectangle
else
  Let y and x be the smallest integers satisfying $M \leq m \cdot y$ and $N \leq n \cdot x$
  Create a horizontal strip by pasting x copies of the $m \times n$ perfect tiling side by side
  Paste y copies of the obtained strip vertically to get $(m \cdot y) \times (n \cdot x)$ perfect tiling
  Draw a rectangle of size $M \times N$ inside the obtained tiling
  Drop any polyomino that lies completely outside the $M \times N$ rectangle
return *Final tiling.*



(a) Initial perfect tiling pasted side-by-side.

(b) Initial imperfect tiling zoomed-in.

Figure 20: Initial solutions of Figure 18 magnified.

8.4.3 Heuristic Procedure: Retile

Procedure Retile is an important part of our heuristic. Consider board B and assume some of its tilings is given (e.g., obtained via any of the procedures described above). Let (r, c) be some square of B. Consider a smaller board S of size $d \times d$ centered at (r, c). There are some polyominoes that cross the boundaries of S and others that are completely inside S. We may fix the polyominoes crossing the boundaries and retile the area covered by the others. Retiling starts with the existing tiling of S (i.e., polyominoes that are completely inside S form a feasible solution for S); therefore,

we are always guaranteed to have a solution, which is at least as good as the original one. Therefore, we possibly may improve irregularity of a very large tiling by solving smaller MIPs.

Procedure Retile

Input: Tiling of $m \times n$ board B, d ($d \leq \min\{m, n\}$), (r, c) for S, and the type of retiling (perfect vs. imperfect).

1 Draw a square S of dimension $d \times d$ centered at (r, c) of B.
2 Divide S into three regions: Center, CoveredFrame, and FreeFrame.
3 Center consists of squares in S that: (i) are covered by polyominoes that are completely in S and (ii) do not belong to the frame of B.
4 CoveredFrame consists of squares in S that: (i) do not belong to the Center and (ii) do not belong to the frame of B.
5 FreeFrame consists of squares in S that do not belong to the frame of B.
6 Mark the Center as *to be covered exactly*,
7 Mark the CoveredFrame as *not to be covered*,
8 **if** *retiling type is 'perfect'* **then**
9 Mark the FreeFrame as *to be "penalized" if covered*.
10 **else**
11 Mark the FreeFrame as *to be packed on*.
12 Using the appropriate entropy maximizing mathematical model, tile S.
13 **return** *Final tiling.*

8.4.4 Heuristic Procedure: Smoothen

We can slightly modify the objective function of formulation P_L "penalizing" for tiled squares in the frame of board B. Procedure Smoothen takes as its input an imperfect tiling (possibly a very large one) of B. Then it attempts to obtain a perfect tiling of B by "moving" along its frame and retiling B while "penalizing" for squares in the frame of B that are tiled.

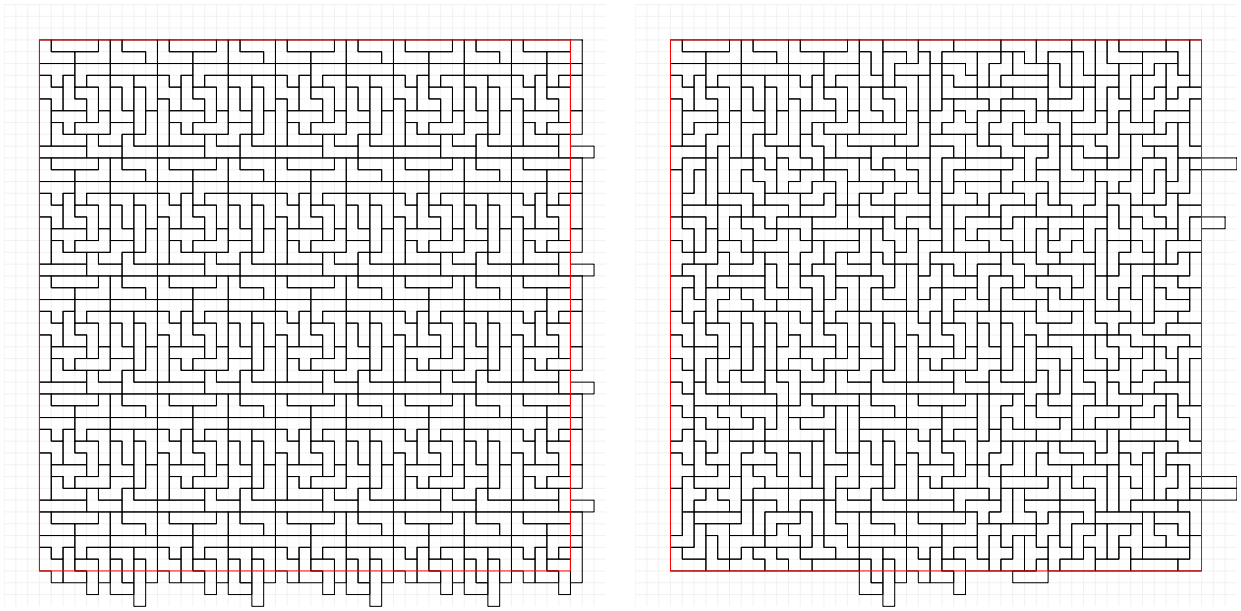
Procedure Smoothen

Input: An imperfect tiling of $m \times n$ board B.

1 Let $d \times d$, $d \leq \min\{m, n\}$, be a square that can be tiled in a short time using standard MIP solvers.
2 Consider the following closed rectangular path P on B :
$(d, d) \rightarrow (d, n-d) \rightarrow (m-d, n-d) \rightarrow (m-d, d) \rightarrow (d, d)$.
3 Let δ be the step size along the path. `// Usually` $\delta \leq d/2$
4 Using the $d \times d$ board and step size δ, retile B perfectly (Alg. Retile) along the path P.
5 **return** *Final tiling.*

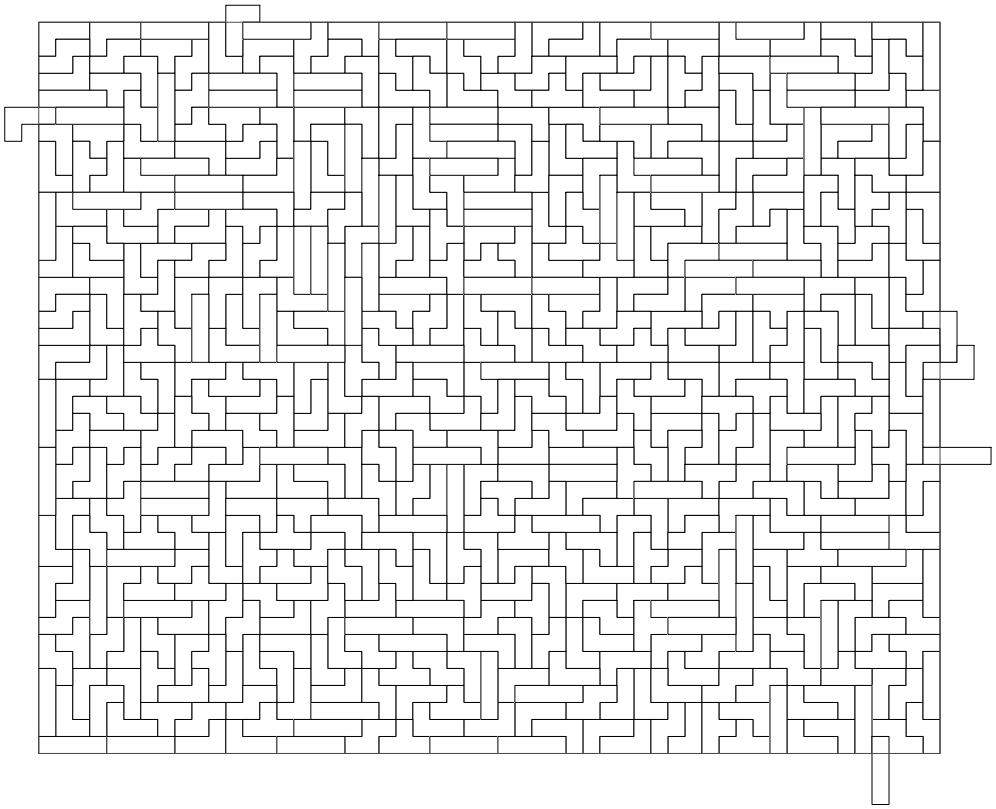
8.4.5 Heuristic Procedure: Randomize

Procedure Randomize is another building block of our heuristic. Given some tiled board B (typically very large), it traverses along B and applies Procedure Retile to "randomize" (i.e., increasing irregularity according to the developed metric) the subregions in B. Figure 21 illustrates application of Procedures Retile and Randomize starting from the tiling of Figure 20(a) to obtain a near-perfect irregular tiling.



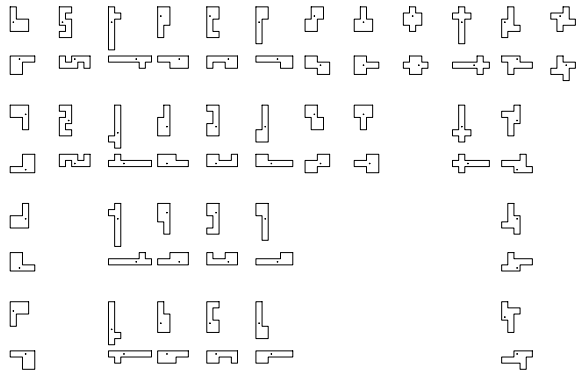
(a) Perfect tiling from 18(a) pasted side-by-side: 6 % gap with respect to the theoretical upper bound on irregularity.

(b) Tiling from 21(a) randomized: 0.2 % gap with respect to the theoretical upper bound on irregularity.

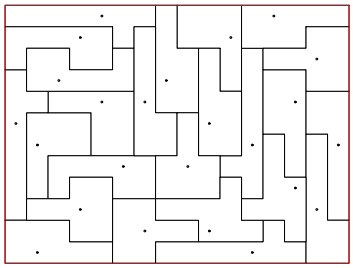


(c) Tiling from 21(b) smoothed along the boundaries.

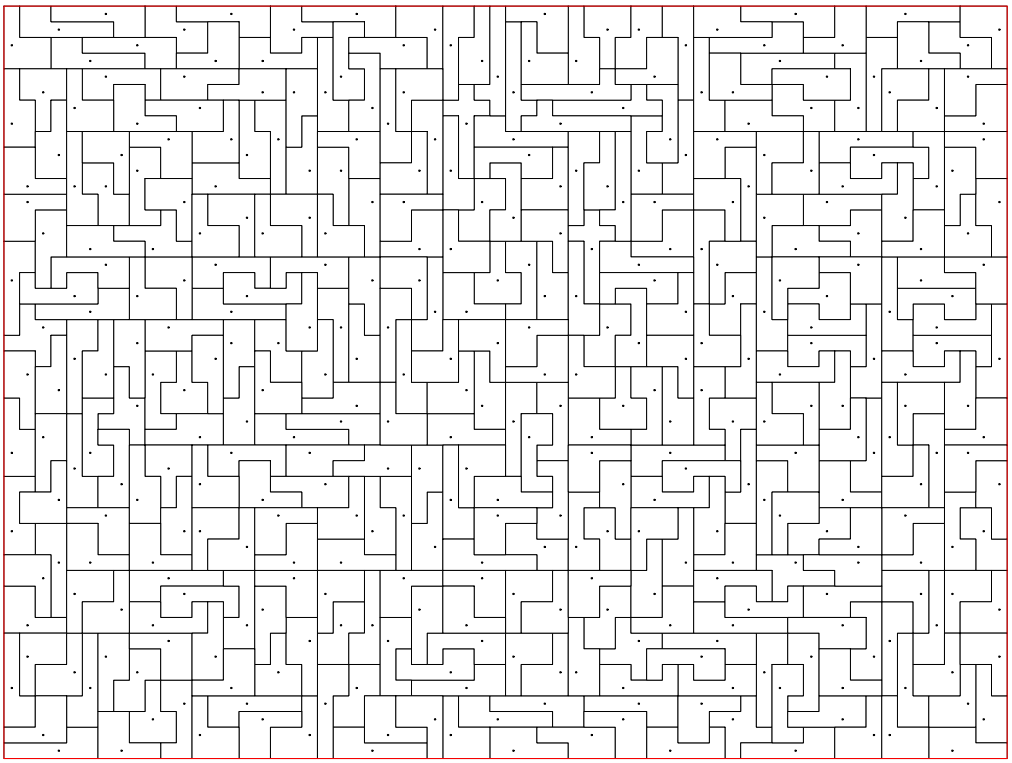
Figure 21: Perfect tiling of Figure 18(a) magnified, randomized and smoothed.



(a) Members of octomino family that are used.

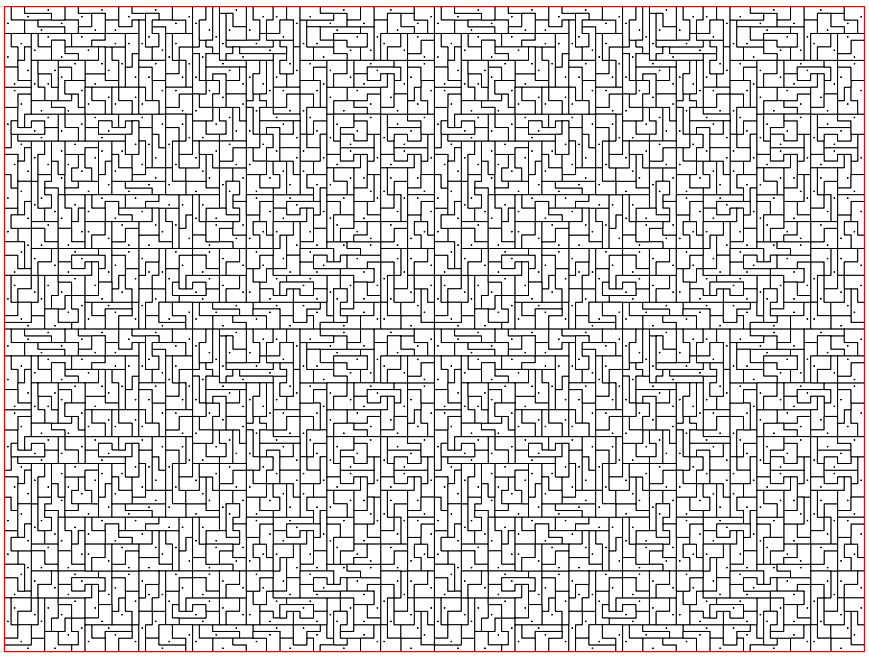


(b) 12×16 tiling.

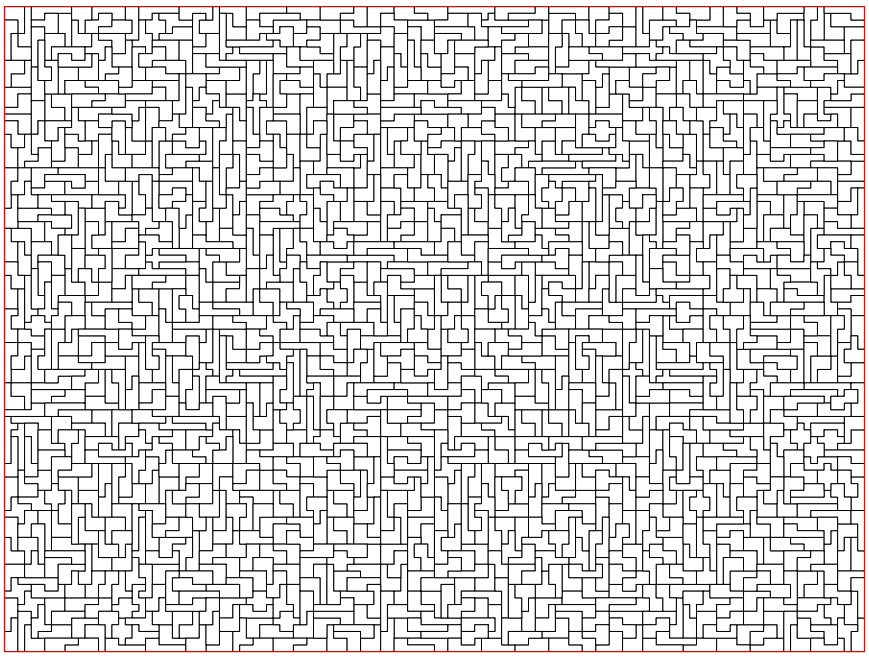


(c) Tiling in (b) "zoomed-in" by $(4, 4)$.

Figure 22: 12×16 board magnified to 48×64 board in less than 1 minute.



(a) Tiling from 22 (c) pasted side-by-side to obtain 96×128 board.



(b) Tiling from 23(a) randomized. Final gap 0.37 % with respect to the theoretical upper bound on irregularity.

Figure 23: Final tiling: original 12×16 board magnified to 96×128 and randomized.

Procedure Randomize

Input: Tiled $m \times n$ board B and the type of the required tiling (perfect vs. imperfect).

1 Let $d \times d$, $d \leq \min\{m, n\}$, be a square that can be tiled in a short time using standard MIP solvers.

2 Let δ be the step size. `// Usually` $\delta \leq d/2$

3 Consider the following path P on B :
$(d, d) \to (d, d+\delta) \to \ldots \to (d, n-d) \to (d+\delta, d) \to \ldots \to (d+\delta, n-d) \to \ldots \to (m-\delta, n-\delta)$.
`// If` $(n-2d)/\delta$ `is not integral:` $\ldots (d, d+\delta\lfloor (n-2d)/\delta \rfloor) \to (d, n-d) \ldots$

4 Using the $d \times d$ board and step size δ, retile B (Pro. Retile) along the path P enforcing the required type of tiling.

5 **return** *Final tiling.*

Figures 22 and 23 illustrates application of the developed heuristic on another problem instance. Comparison to the theoretical upper bound proves that the algorithm is rather successful in obtaining an irregular tiling.

8.5 Current Work and Concluding Remarks

The above described heuristic can be applied to obtain arbitrarily large tilings, though not necessarily perfect. Observe that exact MIP formulation P_L is at the core of the algorithm. Therefore, we are currently working on developing more advanced exact (!) solution approaches that will be able to solve exact tiling problems for larger problem sizes. There are three distinct research directions:

(i) another set partitioning formulation (we are currently performing some preliminary computational tests);

(ii) a better branching strategy (specifically using constraints of P_L);

(iii) a more advanced branch-and-price algorithm.

The PI expects that the first paper on the topic will be submitted for publication within next two-three months. The target journal is *INFORMS Journal on Computing*.

9 Participants

The project participants whose work was supported by this grant are:

- PI: Oleg Prokopyev (University of Pittsburgh);
- co-PI: Nan Kong (Purdue);
- Graduate students (fully or partially supported during last three years): Osman Ozaltin, Serdar Karademir, Behdad Behesti, Anahita Khojandi (all from University of Pittsburgh) and Zhen Zhu (Purdue).

Other major collaborators (i.e., co-authors on some of the papers below that were result of the work in the framework of this project) included: O. Ursulenko, S. Butenko (both from Texas A& M University), S. Rebennack (Colorado School of Mines), A.J. Schaefer and L.M. Maillart (both from University of Pittsburgh).

10 Publications

The following list includes published or accepted for publication journal articles:

- O.Y. Ozaltin, O.A. Prokopyev, A.J. Schaefer, "Two-Stage Quadratic Integer Programs with Stochastic Right-Hand Sides," *Mathematical Programming*, accepted for publication, 2010.
- O.Y. Ozaltin, O.A. Prokopyev, A.J. Schaefer, M.S. Roberts, "Optimizing the Societal Benefits of the Annual Influenza Vaccine: A Stochastic Programming Approach," *Operations Research*, accepted for publication, 2010.
- O.Y. Ozaltin, O.A. Prokopyev, A.J. Schaefer, "The Bilevel Knapsack Problem with Stochastic Right-Hand Sides," *Operations Research Letters*, Vol. 38/4 (2010), pp. 328–333.
- O.A. Prokopyev, N. Kong, D.L. Martinez-Torres, "The Equitable Dispersion Problem," *European Journal of Operational Research*, Vol. 197 (2009), pp. 59-67.
- J. Rajgopal, Z. Wang, A.J. Schaefer, O.A. Prokopyev, "Integrated Design and Operation of Remnant Inventory Supply Chains under Uncertainty," *European Journal of Operational Research*, Vol. 214/2 (2011), pp. 358-364.
- O.A. Prokopyev, S. Butenko, A.C. Trapp, "Checking Solvability of Systems of Interval Linear Equations and Inequalities via Mixed Integer Programming," *European Journal of Operational Research*, Vol. 199/1 (2009), pp. 117-121.

The following list includes technical reports (articles that are submitted for publication to the refereed journals, or under revision):

- S. Karademir, O.A. Prokopyev, N. Kong, "On Greedy Approximation Algorithms for a Class of Two-Stage Stochastic Assignment Problems," Technical report, 2011.
- Z. Zhu, N. Kong, O.A. Prokopyev, "A New Lagrangian Decomposition Based Approach for Quadratic Binary Programs," Technical Report, 2011.

- S. Rebennack, O.A. Prokopyev, "Two-Stage Stochastic Minimum $s-t$ Problem: Formulations and Complexity," Technical Report, 2011.
- O. Ursulenko, S. Butenko, O.A. Prokopyev, "A Global Optimization Algorithm for Solving the Minimum Multiple Ratio Spanning Tree Problem," Technical report, 2011.
- A. Khojandi, L.M. Maillart, O.A. Prokopyev, "Optimal Planning of Life-Depleting Maintenance Activities," Technical report, 2011.

The following list includes of articles that the PIs expect to be submitted for publication within next several months:

- Z. Zhu, N. Kong, O.A. Prokopyev, "Two-Stage Stochastic Quadratic Binary Program with Recourse: A Dual Decomposition Approach," working paper, 2011.
- S. Karademir, O.A. Prokopyev, "Irregular Polyomino Tiling via Integer Programming," working paper, 2011.

11 Presentations

Invited seminars at other universities given by the PIs included:

- "Discovering Underlying Structure with Integer Programming," BMERC Seminar Series, Department of Biomedical Engineering, Boston University, October 1, 2010.
- "Two-Stage Quadratic Integer Programs with Stochastic Right-Hand Sides," Texas A&M University, Department of Industrial Engineering, March 24, 2011.
- "Optimizing the Societal Benefits of the Annual Influenza Vaccine," 2010 University of Florida-Air Force Research Lab Summer Seminar Series, University of Florida, REEF, June 15, 2010.

Major invited conference presentations by the PIs and graduate students included:

- "Two-stage quadratic integer programs with stochastic right-hand sides," Mixed Integer Programming (MIP) Workshop, Waterloo, Canada, June 20-23, 2011.
- "The Bilevel Knapsack Problem with Stochastic Right-Hand Sides," 3rd International Conference on the Dynamics of Information Systems, Gainesville, FL, February 16 - 18, 2011.
- "Two-stage quadratic integer programs with stochastic right-hand sides," 12th International Conference on Stochastic Programming, Halifax, Canada, August 16-20, 2010.
- "On Approximation Algorithms for a Class of Stochastic Assignment Problems," 2nd International Conference on the Dynamics of Information Systems, Destin, FL, February 3 - 5, 2010.
- "A Lagrangian Decomposition Based Branch and Bound Approach to Quadratic 0–1 Programs," 2010 INFORMS Annual Meeting, Austin, TX, November 7-10, 2010.
- "Algorithm Study for Two-stage Stochastic Quadratic 0-1 Recourse Problem," 2010 INFORMS Annual Meeting, Austin, TX, November 7-10, 2010.

- "On Greedy Approximation Algorithms for a Class of Stochastic Assignment Problems," 2010 INFORMS Annual Meeting, Austin, TX, November 7-10, 2010.
- "The Bilevel Knapsack Problem with Stochastic Right-hand Sides," 2010 INFORMS Annual Meeting, Austin, TX, November 7-10, 2010.
- "Two-stage Quadratic Integer Programs with Stochastic Right-hand Sides," 2010 INFORMS Annual Meeting, Austin, TX, November 7-10, 2010.
- "Optimal Strain Selection for the Annual Influenza Vaccine," 2010 INFORMS Annual Meeting, Austin, TX, November 7-10, 2010.
- "Exact Approaches for Solving Minimum Ratio Spanning Trees with Multiple-Ratios," IIE Annual Conference, Miami, FL, May 30 - June 3, 2009.
- "Exact Approaches for Solving Minimum Ratio Spanning Trees with Multiple-Ratios," 2009 INFORMS Annual Meeting, San Diego, CA, October 11-14, 2009.
- "On Global Optimization of Two-stage Stochastic Integer Programs," 2009 INFORMS Annual Meeting, San Diego, CA, October 11-14, 2009.
- "On Two-Stage Quadratic Integer Programs with Stochastic Right-Hand Side," 2009 INFORMS Annual Meeting, San Diego, CA, October 11-14, 2009.

References

[1] A.-Ghouila-Houri. Caracterisation des matrices totalement unimodulaires. *C. R. Academy of Sciences of Paris*, 254:1192–1194, 1962.

[2] J. Abello, S. Butenko, P.M. Pardalos, and M. Resende. Finding independent sets in a graph using continuous multivariable polynomial formulations. *Journal of Global Optimization*, 21:111–137, 2001.

[3] W.P. Adams and R.J. Forrester. A simple approach for generating concise linear representations of mixed 0-1 polynomial programs. *Operations Research Letters*, 33(1):55–61, 2005.

[4] W.P. Adams and R.J. Forrester. Linear forms of nonlinear expressions: New insights on old ideas. *Operations Research Letters*, 35(4):510–518, 2007.

[5] W.P. Adams and H.D. Sherali. A tight linearization and an algorithm for zero-one quadratic programming problems. *Management Science*, 32(10):1274–1290, 1986.

[6] V. Aggarwal, Y.P. Aneja, and K.P.K Nair. Minimal spanning tree subject to a side constraint. *Computers and Operations Research*, 9:287–296, 1982.

[7] S. Ahmed, M. Tawarmalani, and N. V. Sahinidis. A finite branch and bound algorithm for two-stage stochastic integer programs. *Mathematical Programming*, 100(2):355–377, 2004.

[8] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[9] R.K. Ahuja, J.B. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research*, 27(10):917–934, 2000.

[10] B. Alidaee, G.A. Kochenberger, and A. Ahmadian. 0-1 Quadratic programming approach for optimum solutions of two scheduling problems. *International Journal of Systems Science*, 25(2):401–408, 1994.

[11] K. Allemand, K. Fukuda, T.M. Liebling, and E. Steiner. A polynomial case of unconstrained zero-one quadratic optimization. *Mathematical Programming*, 91(1):49–52, 2001.

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

[13] K.M. Anstreicher and N.W. Brixius. A new bound for the quadratic assignment problem based on convex quadratic programming. *Mathematical Programming*, 89(3):341–357, 2001.

[14] G.C. Armour and E.S. Buffa. A heuristic algorithm and simulation approach to relative location of facilities. *Management Science*, 9(2):294–309, 1963.

[15] F. Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A: Mathematical and General*, 15:3241, 1982.

[16] F. Barahona. A solvable case of quadratic 0-1 programming. *Discrete Applied Mathematics*, 13(1):23–26, 1986.

[17] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36(3):493–513, 1988.

[18] C. Berge. *Theori des graphes et ses applications*. Dunod, Paris, 1958.

[19] A. Billionnet, M.-C. Costa, and A. Sutter. An efficient algorithm for a task allocation problem. *Journal of the ACM*, 39:502–518, 1992.

[20] A. Billionnet and S. Elloumi. Best reduction of the quadratic semi-assignment problem. *Discrete Applied Mathematics*, 109(3):197–213, 2001.

[21] A. Billionnet, A. Faye, and É. Soutif. A new upper bound for the 0-1 quadratic knapsack problem. *European Journal of Operational Research*, 112(3):664–672, 1999.

[22] A. Billionnet and É. Soutif. An exact method based on Lagrangian decomposition for the 0-1 quadratic knapsack problem. *European Journal of Operational Research*, 157(3):565–575, 2004.

[23] J.R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer, 1997.

[24] G.R. Bitran and T.L. Magnanti. Duality and sensitivity analysis for fractional programs. *Operations Research*, 24:675–699, 1976.

[25] E. Boros and P. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123:155–225, 2002.

[26] P. Brass, W. Moser, and J. Pach. *Research Problems in Discrete Geometry*. Springer, New York, 2005.

[27] L. Brotcorne, S. Hanafi, and R. Mansi. A dynamic programming algorithm for the bilevel knapsack problem. *Operations Research Letters*, 37(3):215–218, 2009.

[28] R. E. Burkard. Location with spatial interactions: The quadratic assignment problem. In P.B. Mirchandani and R.L. Francis, editors, *Discrete Location Theory*. Wiley, San Mateo, CA, 1991.

[29] S. Busygin, O.A. Prokopyev, and P.M. Pardalos. Feature selection for consistent biclustering via fractional 0–1 programming. *Journal of Combinatorial Optimization*, 10:7–21, 2005.

[30] P. Camion. Caracterisation des matrices unimodulaires. *Cahiers Centre Etudes Rech.*, 5(4), 1963.

[31] P. Camion. Characterization of Totally Unimodular Matrices. *Proceedings of the American Mathematical Society*, 16(5):1068–1073, 1965.

[32] A. Caprara, D. Pisinger, and P. Toth. Exact solution of the quadratic knapsack problem. *INFORMS Journal on Computing*, 11(2):125–137, 1999.

[33] C. C. Carøe and R. Schultz. Dual decomposition in stochastic integer programming. *Operations Research Letters*, 24:37–45, 1999.

[34] C. C. Carøe and J. Tind. L-shaped decomposition of two-stage stochastic programs with integer recourse. *Mathematical Programming*, 83(1–3):451–464, 1998.

[35] R. Chandrasekaran. Ratio spanning trees. *Networks*, 7:335–342, 1977.

[36] R. Chandrasekaran, Y.P. Aneja, and K.P.K. Nair. Minimal cost reliability ratio spanning tree. *Ann. Discrete Math.*, 11:53–60, 1981.

[37] R. Chandrasekaran and A. Tamir. Polynomial testing of the query is $a^b \geq c^d$? with application to finding a minimal cost reliability ratio spanning tree. *Discrete Applied Mathemtaics*, 9:117–123, 1984.

[38] P. Chardaire and A. Sutter. A decomposition method for quadratic zero-one programming. *Management Science*, 41(4):704–712, 1995.

[39] D.Z. Chen, O. Daescu, Y. Dai, N. Katoh, X. Wu, and J. Xu. Optimizing the sum of linear fractional functions and applications. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 707–716, New York, 2000. ACM.

[40] P. Chretienne. A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints. *European Journal of Operational Research*, 43:225–230, 1989.

[41] B. Colson, P. Marcotte, and G. Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256, 2007.

[42] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23:864–894, 1994.

[43] J. W. Daniel. Stability of the solution of definite quadratic programs. *Mathematical Programming*, 5(1):41–53, 1973.

[44] E.D. Demaine and M.L. Demaine. Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. *Graphs and Combinatorics*, 23:195–208, 2007.

[45] S. Dempe. *Foundations of bilevel programming*. Dordrecht: Kluwer Academic, 2002.

[46] S. Dempe and K. Richter. Bilevel programming with knapsack constraints. *Central European Journal of Operations Research*, 8(2):93–107, 2000.

[47] M. A. H. Dempster, M. L. Fisher, L. Jansen, B. J. Lageweg, J. K. Lenstra, and A. H. G. Rinnooy Kan. Analytical evaluation of hierarchical planning systems. *Operations Research*, 29:707–716, 1981.

[48] K. Dhamdhere, V. Goyal, R. Ravi, and M. Singh. How to Pay, Come What May: Approximation Algorithms for Demand-Robust Covering Problems. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 367–378, 2005.

[49] K. Dhamdhere, R. Ravi, and M. Singh. On stochastic minimum spanning trees. In *Proceedings of the 11th International Conference on Integer Programming and Combinatorial Optimization*, 2005.

[50] W. Dinklebach. On nonlinear fractional programming. *Management Science*, 13:492–498, 1967.

[51] S. Elhedhli. Exact solution of a class of nonlinear knapsack problems. *Operations Research Letters*, 33:615–624, 2005.

[52] B. Escoffier, L. Gourves, J. Monnot, and O. Spanjaard. Two-stage stochastic matching and spanning tree problems: Polynomial instances and approximation. *European Journal of Operations Research*, 205:19–30, 2010.

[53] J.E. Falk and S.W. Palocsay. Optimizing the sum of linear fractional functions. pages 221–258. Princeton University Press, Princeton, NJ, 1992.

[54] J.E. Falk and S.W. Palocsay. Image space analysis of generalized fractional programs. *Journal of Global Optimization*, 4:63–88, 1994.

[55] U. Feige, D. Peleg, and G. Kortsarz. The dense k-subgraph problem. *Algorithmica*, 29(3):410–421, 2001.

[56] A.J. Fenn, D.H. Temme, W.P. Delaney, , and W.E. Courtney. The development of phased-array radar technology. *Lincoln Laboratory Journal*, 12(2):321–340, 2000.

[57] M.L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 50(12S):1861–1871, 2004.

[58] FL. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Can. J. Math.*, 8:399–404, 1956.

[59] FL. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, New Jersey, 1962.

[60] R.J. Forrester and H.J. Greenberg. Quadratic binary programming models in computational biology. *Algorithmic Operations Research*, 3(2):110–129, 2008.

[61] A. Frieze, A. Flaxman, and M. Krivelevich. On the random 2-stage minimum spanning tree. *Random Structures and Algorithms*, 28:24–36, 2006.

[62] H. K. Fung, M. S. Taylor, and C. A. Floudas. Novel formulations for the sequence selection problem in de novo protein design with flexible templates. *Optimization Methods and Software*, 22(1):51–71, 2007.

[63] A. M. Geoffrion. Lagrangean relaxation for integer programming. *Mathematical Programming Studies*, pages 82–114, 1974.

[64] F. Glover. Improved linear integer programming formulations of nonlinear integer problems. *Management Science*, 22(4):445–460, 1975.

[65] F. Glover, B. Alidaee, C. Rego, and G. Kochenberger. One-pass heuristics for large-scale unconstrained binary quadratic problems. *European Journal of Operational Research*, 137(2):272–287, 2002.

[66] F. Glover, G.A. Kochenberger, and B. Alidaee. Adaptive memory tabu search for binary quadratic programs. *Management Science*, 44(3):336–345, 1998.

[67] F. Glover and E. Woolsey. Further reduction of zero-one polynomial programming problems to zero-one linear programming problems. *Operations Research*, 21:156–161, 1973.

[68] F. Glover and E. Woolsey. Converting the 0-1 polynomial programming problem to a 0-1 linear program. *Operations Research*, 22(1):180–182, 1974.

[69] S.W. Golomb. *Polyominoes: Puzzles, Patterns, Problems, and Packings*. Princeton University Press, 1994.

[70] D. Golovin, V. Goyal, and R. Ravi. *STACS 2006*, volume 3884/2006 of *Lecture Notes in Computer Science*, chapter Pay Today for a Rainy Day: Improved Approximation Algorithms for Demand-Robust Min-Cut and Shortest Path Problems, pages 206–217. Springer, Berlin / Heidelberg, 2006.

[71] E.G. Gol'stein. Dual problems of convex and fractionally-convex programming in functional spaces. *Soviet Math. Dokl.*, 8:212–216, 1967.

[72] S. K. Goyal and B. C. Giri. Recent trends in modeling of deteriorating inventory. *European Journal of Operations Research*, 134(1):1–16, 2001.

[73] F. Granot and J. Skorin-Kapov. Some proximity and sensitivity results in quadratic integer programming. *Mathematical Programming*, 47(2):259–268, 1990.

[74] M. Guignard and S. Kim. Lagrangean decomposition: A model yielding stronger bounds. *Mathematical Programming*, 39:215–228, 1987.

[75] A. Gupta, R. Ravi, and A. Sinha. Lp rounding approximation algorithms for stochastic network design. *Mathematics of Operations Research*, 32:345–364, 2007.

[76] G.Y. Handler and I. Zhang. A dual algorithm for the constrained shortest path problem. *Networks*, 10:293–310, 1980.

[77] P. Hansen, M. Poggi de Aragão, and C.C. Ribeiro. Hyperbolic 0–1 programming and query optimization in information retrieval. *Mathematical Programming*, 52:256–263, 1991.

[78] M. Held, P. Wolfe, and H.P. Crowder. Validation of subgradient optimization. *Mathematical programming*, 6(1):62–88, 1974.

[79] C. Helmberg and F. Rendl. Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Mathematical Programming*, 82(3):291–315, 1998.

[80] H.-X. Huang, P. M. Pardalos, and O. A. Prokopyev. Lower bound improvement and forcing rule for quadratic binary programming. *Computational Optimization and Applications*, 33(2–3):187–208, 2006.

[81] L.D. Iasemidis, P.M. Pardalos, J.C. Sackellares, and D.S. Shiau. Quadratic binary programming and dynamical system approach to determine the predictability of epileptic seizures. *Journal of Combinatorial Optimization*, 5(1):9–26, 2001.

[82] ILOG S.A. and ILOG Inc. *ILOG CPLEX 12.2*, September 2007.

[83] K. O. Jörnsten, M. Näsberg, and P. A. Smeds. Variable splitting – A new Lagragean relaxation approach to some mathematical programming models. Technical Report LiTH-MAT-R-85-04, Department of Mathematics, Linköping Institute of Technology, Sweden.

[84] M. Juenger, A. Martin, G. Reinelt, and R. Weismantel. Quadratic 0/1 optimization and a decomposition approach for the placement of electronic circuits. *Mathematical Programming*, 63(1):257–279, 1994.

[85] D. Jungnickel. *Graphs, networks and algorithms*. Springer, 2007.

[86] W. K. Klein Haneveld and M. H. van der Vlerk. Optimizing electricity distribution: Using two-stage integer recourse models. In *Stochastic Optimization: Algorithms and Applications*, pages 137–154. Kluwer Academic Publishers, 2001.

[87] J.L. Klepeis, C.A. Floudas, D. Morikis, C.G. Tsokos, and J.D. Lambriss. Design of peptide analogues with improved activity using a novel de novo protein design approach. *Industrial and Engineering Chemistry Research*, 43(14):3817–3826, 2004.

[88] N. Kong and A. Schaefer. A factor 1/2 approximation algorithm for two-stage stochastic matching problems. *European Journal of Operational Research*, 81:387–394, 2006.

[89] N. Kong, A. J. Schaefer, and B. Hunsaker. Two-stage integer programs with stochastic right-hand sides: a superadditive dual approach. *Mathematical Programming*, 108(2):275–296, 2006.

[90] N. Kong and A.J. Schaefer. A factor 1/2 approximation algorithm for two-stage stochastic matching problem. *European Journal of Operational Research*, 172:740–746, 2006.

[91] H.W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[92] G. Laporte, F. V. Louveaux, and H. Mercure. The vehicle routing problem with stochastic travel times. *Transportation Science*, 26:161–170, 1992.

[93] D.J. Laughhunn. Quadratic binary programming with application to capital-budgeting problems. *Operations Research*, pages 454–461, 1970.

[94] E.L. Lawler. The quadratic assignment problem. *Management Science*, 9(4):586–599, 1963.

[95] A. Lodi, K. Allemand, and T.M. Liebling. An evolutionary heuristic for quadratic 0-1 programming. *European Journal of Operational Research*, 119(3):662–670, 1999.

[96] J. A. De Loera, D. Haws, R. Hemmecke, P. Huggins, B. Strumfels, and R. Yoshida. Short rational functions for toric algebra and applications. *Journal of Symbolic Computation*, 38(2):959–973, 2004.

[97] E.M. Loiola, N.M.M. de Abreu, P.O. Boaventura-Netto, P. Hahn, and T. Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690, 2007.

[98] R.J. Mailloux. Phased array theory and technology. *Proceedings of the IEEE*, 70(3):246–302, 1982.

[99] R.J. Mailloux, S.G. Santarelli, and T.M. Roberts. Wideband arrays using irregular (polyomino) shaped subarrays. *Electronics Letters*, 42(18):11–12, 2006.

[100] R.J. Mailloux, S.G. Santarelli, T.M. Roberts, and D. Luu. Irregular polyomino-shaped subarrays for space-based active arrays. *International Journal of Antennas and Propagation*, 2009, 2009. Article ID 956524, 9 pages.

[101] K. Martin. *Large Scale Linear and Integer Optimization: A Unified Approach*. Kluwer Academic Press, 1999.

[102] R. K. Martin. *Large Scale Linear and Integer Optimization: A Unified Approach*. Kluwer Academic Publishers, Boston, MA, 1999.

[103] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998.

[104] J. Maurer. The boost random number library. http://www.boost.org/doc/libs/1_38_0/libs/random/index.html. Accessed 29 April 2009.

[105] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4:414–424, 1979.

[106] A. Migdalas, P. M. Pardalos, and P. Värbrand. *Multilevel optimization: algorithms and applications.* Norwell: Kluwer Academic Publishers, 1998.

[107] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, NewYork, NY, 1988.

[108] Yu. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course.* Springer, 2004.

[109] M. Oral and O. Kettani. A linearization procedure for quadratic and cubic mixed-integer problems. *Operations Research*, 40(S1):109–116, 1990.

[110] M. Oral and O. Kettani. Reformulating nonlinear combinatorial optimization problems for higher computational efficiency. *European Journal of Operational Research*, 58(2):236–249, 1992.

[111] G. Palubeckis. Heuristics with a worst-case bound for unconstrained quadratic 0-1 programming. *Informatica*, 3:225–240, 1992.

[112] G. Palubeckis. Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. *Annals of Operations Research*, 131(1):259–282, 2004.

[113] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Dover Publications,Inc., 1998.

[114] P. M. Pardalos, F. Rendl, and H. Wolkowicz. The quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems, DIMACS Series*, pages 1–42. American Mathematical Society, Providence, RI, 1994.

[115] P. M. Pardalos and G. P. Rodgers. Computational aspects of a branch and bound algorithm for quadratic zero–one programming. *Computing*, 45(2):131–144, 1990.

[116] P.M. Pardalos. Construction of test problems in quadratic bivalent programming. *ACM Transactions on Mathematical Software (TOMS)*, 17(1):74–87, 1991.

[117] P.M. Pardalos and S. Jha. Graph separation techniques for quadratic zero-one programming. *Computers & Mathematics with Applications*, 21(6-7):107–113, 1991.

[118] P.M. Pardalos and S. Jha. Complexity of uniqueness and local search in quadratic 0-1 programming. *Operations Research Letters*, 11(2):119–123, 1992.

[119] P.M. Pardalos, O.A. Prokopyev, O.V. Shylo, and V.P. Shylo. Global equilibrium search applied to the unconstrained binary quadratic optimization problem. *Optimization Methods and Software*, 23(1):129–140, 2008.

[120] P.M. Pardalos and G.P. Rodgers. A branch and bound algorithm for the maximum clique problem. *Computers & Operations Research*, 19(5):363–375, 1992.

[121] P.M. Pardalos and J. Xue. The maximum clique problem. *Journal of Global Optimization*, 4(3):301–328, 1994.

[122] D. Parker and D.C. Zimmermann. Phased arrays - part I: Theory and architectures. *IEEE Transactions on Microwave Theory and Techniques*, 50(13):678–687, 2002.

[123] J.C. Picard and H.D. Ratliff. Minimum cuts and related problems. *Networks*, 5(4):357–370, 1975.

[124] D. Pisinger. The quadratic knapsack problem – A survey. *Discrete Applied Mathematics*, 155:623–648, 2007.

[125] O.A. Prokopyev. Fractional zero-one programming. In C. Floudas and P.M. Pardalos, editors, *Encyclopedia of Optimization*, pages 1091–1094. Springer, 2009.

[126] O.A. Prokopyev, H.-Z. Huang, and P.M. Pardalos. On complexity of unconstrained hyperbolic 0–1 programming problems. *Operations Research Letters*, 33:312–318, 2005.

[127] O.A. Prokopyev, C. Meneses, C.A.S. Oliveira, and P.M. Pardalos. On multiple-ratio hyperbolic 0–1 programming problems. *Pacific Journal of Optimization*, 1:327–345, 2005.

[128] T. Radzik. Parametric flows, weighted means of cuts, and fractional combinatorial optimization. In P.M. Pardalos, editor, *Complexity in Numerical Optimization*, pages 351–386. World Scientific, 1993.

[129] T. Radzik. Fractional combinatorial optimization. In C.A. Floudas and P.M. Pardalos, editors, *Encyclopedia of Optimization*, volume 2, pages 159–161. Kluwer Academic Publishers, 2001.

[130] G. Reinelt. Effiziente Algorithmen 1. Lecture notes. Institut für Informatik, Universität Heidelberg, 2001.

[131] R. T. Rockafellar and R.J.-B. Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16:1–23, 1991.

[132] C. H. Rosa and Ruszczyński. On augmented Lagrangian decomposition methods for multistage stochastic programs. *Annals of Operations Research*, 64:289–309, 1996.

[133] A. Ruszczyński and A. Shapiro, editors. *Handbooks in OR & MS: Stochastic Programming, Vol. 10*. Elsevier, 2003.

[134] S. Schaible. Duality in fractional programming: A unified approach. *Operations Research*, 24:452–461, 1976.

[135] S. Schaible and J. Shi. Fractional programming: the sum-of-ratios case. *Optimization Methods and Software*, 18:219–229, 2003.

[136] R. Schultz. On structure and stability in stochastic programs with random technology matrix and complete integer recourse. *Mathematical Programming*, 70(1):73–89, 1995.

[137] R. Schultz, L. Stougie, and M. H. van der Vlerk. Solving stochastic programs with integer recourse by enumeration: A framework using Gröbner basis reductions. *Mathematical Programming*, 83(1–3):229–252, 1998.

[138] H.D. Sherali and W.P. Adams. *A Reformulation-linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.

[139] J. Siek and L.-Q. Lee. The boost graph library. http://www.boost.org/doc/libs/1_38_0/libs/graph/doc/index.html. Accessed 29 April 2009.

[140] C.C. Skiscim and S.W. Palocsay. Minimum spanning trees with sums of ratios. *Journal of Global Optimization*, 19:103–120, 2001.

[141] C.C. Skiscim and S.W. Palocsay. The complexity of minimum ratio spanning tree problems. *Journal of Global Optimization*, 30:335–346, 2004.

[142] Y. Soun and K. Truemper. Single Commodity Representation of Multicommodity Networks. *SIAM. J. on Algebraic and Discrete Methods*, 1:348–358, 1980.

[143] I.M. Stancu-Minasian. A sixth bibliography of fractional programming. *Optimization*, 55:405–428, 2006.

[144] A. Tamir. On totally unimodular matrices. *Networks*, 6(4):373–382, 1976.

[145] M. Tawarmalani, S. Ahmed, and N. Sahinidis. Global optimization of 0–1 hyperbolic programs. *Journal of Global Optimization*, 24:385–416, 2002.

[146] D. Towsley. Allocating programs containing branches and loops within a multiple processor system. *IEEE Transactions on Software Engineering*, SE-12:1018–1024, 1986.

[147] K. Truemper. *Matroid Decomposition*. Academic Press, Boston, revised edition leibniz, plano, texas edition, 1998.

[148] R. J-B. Wets. Stochastic programs with fixed recourse: The equivalent deterministic problem. *SIAM Review*, 16:309–339, 1974.

[149] L.A. Wolsey. *Integer Programming*. John Wiley & Sons, 1998.

[150] X. Zhao, PB Luh, and J. Wang. Surrogate gradient algorithm for Lagrangian relaxation. *Journal of Optimization Theory and Applications*, 100(3):699–712, 1999.