Generation Approaches for Combinatorial Optimization Problems

Wei WU

Generation Approaches for Combinatorial Optimization Problems

Wei WU

Department of Computer Science and Mathematical Informatics
Graduate School of Information Science
Nagoya University
Nagoya 464-8601, Japan

March, 2017

Doctoral Dissertation

submitted to Graduate School of Information Science, Nagoya University

in partial fulfilment of the requirement for the degree of

DOCTOR OF INFORMATION SCIENCE

Preface

Over the last decade, combinatorial optimization is one of the most active topics in applied mathematics and theoretical computer science. Many important applications have been developed based on combinatorial algorithms in several areas, including transportation, production management, artificial intelligence, and software engineering. Indeed, combinatorial optimization has its roots from combinatorics and operations research aiming to solve real-life problems.

Combinatorial optimization can be viewed as a problem to search for the best element combination out of a set of discrete items. Some of the classical problems in combinatorial optimization, such as spanning trees, shortest paths, matching, are polynomial-time solvable, while still a variety of them, such as the traveling salesman problem, generalized assignment problem, set covering problem, are known to be NP-hard.

Most problems in combinatorial optimization can be formulated naturally as linear programs or integer programs, both of which can be represented by a matrix. However, in many practical applications, the corresponding matrices become so large (sometimes exponential in the size of the instance) that it is impractical to represent all rows or columns of them. Row generation and column generation have proven to be two of the most successful approaches for solving such kind of large-scale combinatorial optimization problems effectively.

In this thesis, we describe basic ideas, theoretical results, and practical applications related to row/column generation approaches. We also summarize major similarities and differences between row generation and column generation approaches.

Benders decomposition, known as a typical row generation approach, was proposed by Benders five decades ago. In Benders decomposition, the variables of the original problem is separated into two sets for a two-stage iterated approach. In the first stage, a master problem is solved on the first set of variables, and the obtained solution in the first stage is treated as the input for a second stage problem, which we call a subproblem. In the second stage, the subproblem determines the feasibility of the first stage solution, and it updates the current master problem by adding new constraint (row) until no constraints (rows) can be generated. Logic-based Benders decomposition can be viewed as a generalization of the classical Benders method, in which a master problem solves a relaxation problem with some constraints removed from the original problem, and a subproblem aims to generate a critical constraint (row). A variety of real-life applications are successfully solved via classical or logic-based Benders decomposition, including robust optimization, planning and scheduling, resource management, and network design.

Column generation approach was proposed almost at the same time as Benders decomposition by Dantzig and Wolfe. This technique was first brought into real application by Gilimore and Gomory for solving the cutting stock problem. Similar to the row generation approach, in the first stage, a master problem in the column generation approach is solved with a subset of all columns from the original problem. In the second stage, a subproblem determines the optimality of the first stage solution, and it updates the current master problem by adding new columns until no column can be generated. Numerous column generation applications are described in the literature, including airline crew pairing problem, various vehicle routing problems, and bin packing and cutting-stock problems.

In most cases, warm-starts for initial master problems and designing algorithms for subproblems have been considered as two main components to elaborate in solving real-life problems. In this thesis, we apply the row and column generation techniques in solving robust optimization and airline crew pairing problems, incorporating various ideas in these two main components to improve the overall convergence speed.

For the case of robust optimization, we consider combinatorial optimization problems under the min-max regret criteria with interval objective coefficients. We design a Logic-based Benders decomposition approach and also proposed many state-of-the-art exact and heuristic algorithms, some of which are partially based on Benders cuts. Moreover, we propose a new heuristic approach based on a row generation approach, which we call an iterated dual substitution algorithm. We analyze the performance of the developed Logic-based Benders decomposition approach and compared it with other existing algorithms.

For the case of airline crew pairing problem, We propose a branch-and-cut method based upon a resource constrained dynamic programming for a subproblem. Computational results are given for a number of large-scale instances with up to 10,000 flights.

We hope that the descriptions and ideas we discuss in this thesis will contribute to providing fundamental motivation and designing generation approaches for some other real-life applications.

March, 2016
Wei WU

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor professor Mutsunori Yagiura for your enthusiastic guidance, patient support of my Ph.D. study, and for continuous encouragement. You helped me in all the time of research and taught me how to become a qualified researcher from my master course. I could not have imagined having a better supervisor for my Ph.D study.

I would like to express thanks to associate professor Hediki Hashimoto and assistant professor Yannan Hu for your priceless comments and suggestions on both research as well as on my personal annoyance. I would also like to express my special appreciation and thanks to professor Silvano Martello and professor Manuel Iori for your warm inspirations and valuable advice on research of robust optimization. I gratefully thank professor Tomio Hirata for your generous help and insightful comments.

My sincere thanks also goes to professor Yo Matsubara and professor Takafumi Kanamori for being my thesis committee, reading my thesis, and providing me with an inspiration in many ways.

I thank all members in Yagiura and Hirata laboratories for their understanding and precious friendship. I would especially like to thank Nitori International Scholarship Foundation and Ushio Scholarship. Without your financial support it would not be possible to finish this thesis.

Last but not least, I would like to thank my family for all of the sacrifices that you have made on my behalf.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

This chapter starts with the general background of the thesis in Section 1.1. We provide a technical review of topics relevant to row generation approaches in Section 1.1.1, including Benders decomposition, logic-based Benders decomposition, and cutting plane method. We review successful applications of column generation and provide some historical and recent contributions in Section 1.1.2. The Structure of this thesis is presented in Section 1.2.

1.1 General Background

Over the last decade, combinatorial optimization is one of the most active topics in applied mathematics and theoretical computer science [59]. A diverse range of real world problems can be naturally modeled as combinatorial optimization problems in different areas. Indeed, combinatorial optimization has its roots from combinatorics and operations research aiming to solve real-life problems.

Combinatorial optimization can be viewed as a problem to search for the best element combination out of a set of discrete items. Some of the classical problems in combinatorial optimization, such as the spanning trees, shortest paths, matching, are polynomial-time solvable, while still many of them, such as the traveling salesman problem, generalized assignment problem, set covering problem, are known to be NP-hard [80].

A large portion of the problems have their linear programming or integer programming formulation, both of which can be represented by a matrix. However, in many practical applications, the matrix becomes so large (sometimes exponential in the size of the instance) that it is impractical to explicitly represent all rows or columns of it. Generation approaches including row generation and column generation have proven to be successful for solving such kind of large-scale combinatorial optimization problems effectively.

1.1.1 Row Generation

The classical *Benders decomposition*, originally proposed by Benders in the 1960s [17], devised a row generation approach for exploiting the structure of mixed integer programming (MIP). The classical Benders decomposition divides the variables in an original problem (MIP) into two sets, continuous variables and integer ones. The problem with integer variables temporarily fixed is significantly easier to solve than the original problem. A master problem is defined to obtain a solution for the integer variables. Then, a subproblem problem with only continuous variables is defined to generate constraints that are violated by the obtained integer solution (i.e., constraints that cut from the feasible region, the obtained integer solution), and the generated constraints are then added to the current master problem. The procedure repeats until no valid constraint exists when solving the current subproblem. Geoffrion, by using non-linear convex duality theory, generalized Benders decomposition to a broader class of problems in which the subproblem need not to be a linear program [42]. Hooker and Ottosson introduced another extension called *logic-based Benders decomposition* [51]. The subproblem in the logic-based Benders decomposition needs not to be a specific form: it can be a MIP [52] and a feasibility-checking problem [100]. The classical and extended Benders decomposition have been widely used in a large variety of applications, such as planning and scheduling [25, 20, 50, 65, 71], routing [64, 71, 88] and supply chain optimization [33, 78, 98, 109]. For a state-of-the-art analysis of the existing literature, we refer the interested reader to Rahmaniani et al. [87].

The *cutting plane method* can also be seen as a row generation approach. Gomory first proposed this idea to solve MIP [45] (see [57] for convex programming problems). The cutting plane method first solves the LP relaxation of a MIP problem. If the obtained solution is not integral, the method generates constraints that cut the obtained solution but do not cut any integral solution for the original problem. The procedure iterates until an integral optimal solution is obtained. However, such a pure cutting plane method is considered impractical for large-scale instances due to numerical instability, as well as ineffective because an extremely large number of cuts are required for convergence [46, 79, 82, 102]. The situation turned around when Balas et al. [10] adapted this method into the branch-and-bound framework in the 1990s. The basic idea involved can be traced back to Padberg and Rinaldi [79]. Most recent commercial MIP solvers use Gomory cuts in one way or another.

Sethi and Tompson introduced another row generation algorithm called *pivot and probe algorithm* (PAPA) [96], based on the observation that in randomly generated LP problems, only about 15 to 25% of all constraints are candidate constraints and about 70-90% of them are tight at the optimum [95]. Ben-Ameur and Neto proposed a constraint generation algorithm for large-scale linear programs [16]. This kind of row generation algorithm has been recently adopted to many combinatorial optimization problems, such as the two-machine flow shop problem [26], and the sensor resource management problem [103].

1.1.2 Column Generation

Ford and Fulkerson first proposed a column-reduced approach in solving a multicommodity flow problem almost six decades ago [39]. Two years later, Dantzig and Wolfe generalized their idea to design a strategy for solving any linear program [27]. Dantzig-Wolfe decomposition decompose the constraints into two types, master constraints and sub-problem constraints. This method generates columns as needed by solving one or more subproblems. Gilmore and Gomory first applied column generation in solving the *cutting stock problem* [43, 44], which is known as the first application to an integer programming problem. They proposed a model in which each column represents a "pattern" to cut a roll. Then each pattern is a solution of a subproblem (e.g., in their case a *knapsack problem*). A master problem is the LP relaxation of the cutting stock problem based on the generated cutting patterns. In their approach, master problems interact with subproblems to identify effective patterns. It terminates when subproblems cannot generate any effective pattern for the current master problem. Such a situation ensures that the obtained solution is optimal to the original LP relaxation. In fact, this optimal solution value is shown to be a highly tight bound on the IP problem [93]. Several works discussed the absolute gap between LP and IP, and it is observed that there exist the instances with the difference larger than one [69, 94]. However, it is still not known whether there exists an instance with this difference not less than two. In the last step, Gilmore and Gomory obtain an feasible solution by rounding up all the fractional variables in the optimal solution to the master problem. Over the last decade, many successful applications have been studied including the airline crew scheduling problem [47, 21], the operating room planning problem [36], and the vehicle routing problem [5, 66].

Note that a tighter bound is available by solving IP with generated columns. However, this IP can still be difficult to solve even without an exponential number of columns. On the other hand, for large-scale optimization problems in practice, most columns have their variable values equal zero in any optimal solution. Desrosiers et al. proposed a compatible scheme for the vehicle routing problem that overcome these difficulties by incorporating column generation algorithm in a branch-and-bound framework [32]. Barnhart et al. introduced a general technique for column generation and named it *branch-and-price* [13]. At the start of a branch-and-price method, a number of columns are reduced from the LP relaxation for saving computation time. Then, at each node of the search tree, columns may be added to the LP relaxation on demand. Cutting plane technique is considered to strengthen the relaxation of branch-and-price, and this is called branch-price-and-cut (see e.g., [12, 81]). Branch-and-price has many successful applications, such as the generalized assignment problem [92], the crew scheduling problem [31], the bin packing problem with conflicts [90], the vehicle routing problem [29, 48, 86], the nurse scheduling problem [15, 68], the multiple length cutting stock problem [3], and the capacitated facility location problem [58].

For a detailed survey and technical implements, we refer the interested reader to Lübbecke and Desrosiers [67], Desaulniers et al. [30], Wilhelm [101], and Nemhauser [76].

1.2 Thesis Structure

Following this introduction, the thesis is composed of seven chapters. In Chapter 2, we show the theoretical basis of row generation and explain the most famous row generation, Benders decomposition, as well as one of its generalizations logic-based Benders decomposition. In Chapter 3, we discuss column generation approaches and present two mathematical models for solving linear programs including Dantzig-Wolfe decomposition.

In Chapter 4, we consider the generalized assignment problem (GAP) with min-max regret criterion under interval costs. We propose exact algorithmic approaches, including a logic-based Benders decomposition approach, and two Benders cut driven branch-and-cut algorithms. Regarding these two branch-and-cut algorithms, we first examine a basic branch-and-cut and further introduce a more sophisticated algorithm that incorporates various methodologies, including Lagrangian relaxation and variable fixing. The resulting Lagrangian-based branch-and-cut algorithm performs satisfactorily on benchmark instances. We also computationally examine two heuristic methods: a fixed-scenario approach and a dual substitution algorithm. For the fixed-scenario approach, we show that solving the classical GAP under a median-cost scenario leads to a solution of the min-max regret GAP whose objective function value is within twice the optimal value.

Chapter 5 describes another application of the row generation technique. We consider the multidimensional knapsack problem (MKP) under min-max regret criterion. We propose a new heuristic framework, which we call the iterated dual substitution (IDS) algorithm. The IDS iteratively generates linear constraints (rows) based on a mixed integer programming model. Computational experiments on a wide set of benchmark instances are carried out, and the proposed iterated dual substitution algorithm performs best on all of the tested instances.

In Chapter 6, we introduce a crew pairing problem, which is a well-known practical use of column generation. We consider the crew pairing problem in airline scheduling that calls for assigning crew members in order to cover all flights with the minimum total person-days under the constraints that the schedule of each crew member does not violate given constraints on the total working time, flying time, and the number of landings. We formulate the problem as a set covering problem and apply a column generation approach to generate a candidate set of schedules. In the pricing step, we propose a branch-and-bound method based upon a resource constrained dynamic programming. Computational results are given for a number of large-scale instances with up to 10,000 flights.

Chapter 7 summarises our results in this thesis.

Chapter 2

Row Generation

In this chapter, we describe the basic idea of row generation or *constraint generation* approaches for combinatorial optimization problems. We outline a general framework of row generation approach in Section 2.1. We explain in Section 2.2 the motivation, concept and ideas of Benders decomposition, which is known as a typical row generation approach, and in Section 2.3, we extend this strategy to a more general approach, logic-based Benders decomposition, which can solve a much larger class of combinatorial optimization problems.

2.1 General Framework

The motivation of row generation is that many mathematical programming problems that contain too many constraints may not be solvable in their original formulations. Row generation approaches have been used in many well-known algorithms and have numerous applications. In these algorithms, a relaxation of the original problem (master problem) containing only a subset of the constraints is first solved. Then a separation procedure (in which a subproblem or a separation problem is solved) is invoked, which adds to the master problem a set of constraints that are violated by the current optimal solution. The procedure iterates until no violated constraint can be generated in subproblems. The basic framework of a row generation algorithm is shown in Algorithm 1.

Algorithm 1 Basic framework of a row generation approach

```
1: Initialize a constraint set $C := \emptyset$.
2: Initialize an initial master problem (e.g., a relaxation of the original problem with a
   subset of the constraints).
3: repeat
4:     Add all constraints in $C$ to the current master problem.
5:     Solve the current master problem, and obtain an optimal solution $x$.
6:     Generate a set of constraints $C$ that separates $x$ from the feasible region.
7: until $C$ is empty
8: return $x$.
```

2.2 Benders Decomposition

We present the classical version of the Benders decomposition for solving MIP problems. Let $\mathbb{Z}_{\geq 0}$ denote the set of nonnegative integer. We consider a basic MIP formulation:

$$\min \sum_{i=1}^{m} a_i x_i + \sum_{j=1}^{n} b_j y_j \tag{2.2.1}$$

$$\text{s.t.} \sum_{i=1}^{m} c_{ik} x_i + \sum_{j=1}^{n} d_{jk} y_j \geq f_k \qquad \forall k \in K \tag{2.2.2}$$

$$x_i \in \mathbb{Z}_{\geq 0} \qquad \forall i \in I \tag{2.2.3}$$

$$y_j \geq 0 \qquad \forall j \in J, \tag{2.2.4}$$

with m integer variables $x_i, \forall i \in I = \{1, 2, \ldots, m\}$, and n continuous variable $y_j, \forall j \in J = \{1, 2, \ldots, n\}$. In this form, there are $|K|$ constraints, where $K = \{1, 2, \ldots, l\}$. If all the integer variables are fixed to $\bar{x}$, the model can be optimized by solving the following linear programming (LP) problem $P(\bar{x})$:

$$\min \sum_{j=1}^{n} b_j y_j \tag{2.2.5}$$

$$\text{s.t.} \sum_{j=1}^{n} d_{jk} y_j \geq f_k - \sum_{i=1}^{m} c_{ik} \bar{x}_i \qquad \forall k \in K \tag{2.2.6}$$

$$y_j \geq 0 \qquad \forall j \in J. \tag{2.2.7}$$

Let $P(\bar{x})$ also denote the optimal value of problem (2.2.5)–(2.2.7). Then, the problem (2.2.1)–(2.2.4) can be reformulated as:

$$\min \sum_{i=1}^{m} a_i x_i + P(x) \tag{2.2.8}$$

$$\text{s.t.}\ \ x \text{ is feasible to problem } P(x) \tag{2.2.9}$$

$$x_i \in \mathbb{Z}_{\geq 0} \qquad \forall i \in I. \tag{2.2.10}$$

Let α_k denote the dual variable associated with the constraint in (2.2.6) for all $k \in K$. Then, the dual problem $D(\bar{x})$ of $P(\bar{x})$ can be expressed as

$$\max \sum_{k=1}^{l} \left(f_k - \sum_{i=1}^{m} c_{ik} \bar{x}_i \right) \cdot \alpha_k \tag{2.2.11}$$

$$\text{s.t.} \sum_{k=1}^{l} d_{jk} \alpha_k \leq b_j \qquad \forall j \in J \tag{2.2.12}$$

$$\alpha_k \geq 0 \qquad \forall k \in K. \tag{2.2.13}$$

We define $D(\bar{x})$ also be the opmial value of problem (2.2.11)–(2.2.13). Based on strong duality, $D(\bar{x}) = P(\bar{x})$. By replacing $P(\bar{x})$ with $D(\bar{x})$ in problem (2.2.8)–(2.2.8), problem (2.2.5)–(2.2.7) can be reformulated again as:

$$\min \sum_{i=1}^{m} a_i x_i + D(x) \tag{2.2.14}$$

$$\text{s.t.}\ \ x \text{ is feasible to problem } P(x) \tag{2.2.15}$$

$$x_i \in \mathbb{Z}_{\geq 0} \qquad \forall i \in I. \tag{2.2.16}$$

Notice that the feasible region of $D(\bar{x})$ is independent of the value $\bar{x}$. Thus, If the feasible region is not empty, this problem is either feasible or unbounded for any $\bar{x}$. Let R and Q be the set of extreme rays and extreme points of problem $D(\bar{x})$, respectively. The following inequalities must hold

$$\sum_{k=1}^{l} (f_k - \sum_{i=1}^{m} c_{ik} \bar{x}_i) \cdot \beta_k \leq 0 \qquad \forall \beta \in R \tag{2.2.17}$$

to satisfy the feasibility of problem $P(\bar{x})$. Consequently, problem (2.2.14)–(2.2.16) can be re-expressed as:

$$\min \sum_{i=1}^{m} a_i x_i + \max_{\alpha \in Q} \sum_{k=1}^{l} (f_k - \sum_{i=1}^{m} c_{ik} x_i) \cdot \alpha_k \tag{2.2.18}$$

$$\text{s.t.} \sum_{k=1}^{l} (f_k - \sum_{i=1}^{m} c_{ik} x_i) \cdot \beta_k \leq 0 \qquad \forall \beta \in R \tag{2.2.19}$$

$$x_i \in \mathbb{Z}_{\geq 0} \qquad \forall i \in I. \tag{2.2.20}$$

By introducing a new variable θ to (2.2.18)–(2.2.20), the Benders formulation of MIP could be expressed as:

$$\min \sum_{i=1}^{m} a_i x_i + \theta \tag{2.2.21}$$

$$\text{s.t.} \quad \sum_{k=1}^{l} (f_k - \sum_{i=1}^{m} c_{ik} x_i) \cdot \beta_k \leq 0 \qquad \forall \beta \in R \tag{2.2.22}$$

$$\sum_{k=1}^{l} (f_k - \sum_{i=1}^{m} c_{ik} x_i) \cdot \alpha_k \leq \theta \qquad \forall \alpha \in Q \tag{2.2.23}$$

$$x_i \in \mathbb{Z}_{\geq 0} \qquad \forall i \in I. \tag{2.2.24}$$

Constraints (2.2.22) and (2.2.23) are called feasibility cuts and optimality cuts, respectively. It is known that the polyhedron of any LP feasible region can be expressed by finite sets of extreme rays and extreme points. Hence, the sizes of R and Q are finite. However, it is not impractical in most cases to enumerate all the feasibility cuts and the optimality cuts. A row generation approach is proposed by Benders to deal with this difficulty.

A master problem in the classical Benders decomposition can be formulated as:

$$\min \sum_{i=1}^{m} a_i x_i + \theta \tag{2.2.25}$$

$$\text{s.t.} \quad \sum_{k=1}^{l} (f_k - \sum_{i=1}^{m} c_{ik} x_i) \cdot \beta_k \leq 0 \qquad \forall \beta \in \bar{R} \tag{2.2.26}$$

$$\sum_{k=1}^{l} (f_k - \sum_{i=1}^{m} c_{ik} x_i) \cdot \alpha_k \leq \theta \qquad \forall \alpha \in \bar{Q} \tag{2.2.27}$$

$$x_i \in \mathbb{Z}_{\geq 0} \qquad \forall i \in I, \tag{2.2.28}$$

where $\bar{R}$ and $\bar{Q}$ are subsets of R and Q, respectively. This formulation is a relaxation of the original MIP problem only considering a subset of the feasibility cuts and the optimality cuts. The classical Benders decomposition repeatedly solves the master problem, obtaining a solution $(\bar{x}, \bar{\theta})$. It then solves the subproblem $D(\bar{x})$, obtaining an optimal solution $\bar{\alpha}$ and its objective value. If $D(\bar{x})$ is bounded and its optimal value is greater than $\bar{\theta}$, $\bar{\alpha}$ is added to $\bar{Q}$ (i.e., the optimality cut defined by $\bar{\alpha}$ is added to (2.2.27)), obtaining an updated master problem. If $D(\bar{x})$ is unbounded, an extreme ray $\bar{\beta}$ of $D(\bar{x})$ is added to $\bar{R}$ (i.e., a feasibility cut is added to (2.2.26)). On the other hand, if the optimal value of $D(\bar{x})$ is less than or equal to $\bar{\theta}$, no cut can be generated to separate the current solution $\bar{x}$ from feasible region, (i.e., $\bar{x}$ is valid for all the constraints in (2.2.26) and (2.2.27)). The approach iterates until such a feasible solution is obtained.

2.3 Logic-Based Benders Decomposition

Hooker and Ottosson described the idea of the logic-based Benders decomposition [51]. They introduced that each subproblem in this Benders-like decomposition can be a generalized dual, so-called *inference dual*, that aims to seek the tightest possible bound on the optimal value of the master problem. In other words, a solution of the inference dual problem is a proof of the optimality for the original problem. The inference dual satisfies a trivial form of strong duality. However, in each non-final iteration of the logic-based Benders decomposition, the following property holds for a minimization problem:

$$\begin{aligned} \text{the optimal value of the master} &\leq \text{the optimal value of the original problem} \\ &\leq \text{the optimal value of the subproblem (inference dual).} \end{aligned}$$

Based on this property, in each iteration a Benders cut is generated for the next iteration.

Different from the classical Benders decomposition, the subproblems are not needed to be LP problems. In some cases, they can even be NP-hard problems, one of which is shown in Chapter 4.

Chapter 3

Column Generation

Column generation has been known as one of the most successful methods for solving combinatorial optimization problems. We explain the basic idea of column generation in Section 3.1. Section 3.2 describes a general approach for solving linear programming problems. This approach is widely used as an efficient algorithm for a variety of combinational optimization problems, such as airline crew scheduling and cutting stock problem. We explain another type of column generation, Dantzig-Wolfe decomposition, in Section 3.3.

3.1 General Framework

Column generation, analogous to row generation, has been proven to be an efficient approach for large-scale combinatorial optimization problems. This approach is mostly applied to the problems that contain a huge (in many cases exponential) number of variables compared to the number of constraints. For each variable, the corresponding coefficients in the objective function and in the constraint matrix are called *columns*. In a basic column generation approach, we consider a master problem by reducing a number of columns from the original problem rather than explicitly enumerating all the columns. Note that, different from row generation, any feasible solution to a master problem is also feasible for the original problem. A subproblem aims to find a column set from the original column set to improve the incumbent value. Then we add such obtained column(s) to the current master problem, and solve it again. This procedure iterates until no column can improve the incumbent value. Thus, the obtained optimal solution is an optimal solution to the original problem. Let C_{all} be the column set of the original problem. The basic framework of a column generation approach is described in Algorithm 2.

Algorithm 2 Basic framework of a column generation approach

```
1: Initialize an initial master problem with a column set $C(\subseteq C_{\mathrm{all}})$, for which a feasible
   solution exists.
2: repeat
3:     Add all columns in $C$ to the current master problem.
4:     Solve the current master problem, and obtain an optimal value and its solution $x$.
5:     Generate a set of columns $C$ from $C_{\mathrm{all}}$ to improve the incumbent value.
6: until $C$ is empty
7: return $x$.
```

3.2 Basic Column Generation for Linear Program

Let us consider the following linear program, called the *master problem*

$$\begin{aligned} \min \ & \sum_{j \in J} c_j x_j \\ \text{s.t.} \ & \sum_{j \in J} a_{ij} x_j \geq b_i && \forall i \in I \\ & x_j \geq 0 && \forall j \in J, \end{aligned}$$

where $J = \{1, 2, \ldots, n\}$ and $I = \{1, 2, \ldots, m\}$. Column generation starts with a *restricted master problem*, which contains only a subset $J' \subseteq J$ of columns:

$$\begin{aligned} \min \ & \sum_{j \in J'} c_j x_j \\ \text{s.t.} \ & \sum_{j \in J'} a_{ij} x_j \geq b_i && \forall i \in I \\ & x_j \geq 0 && \forall j \in J'. \end{aligned}$$

We consider the dual problem of the restricted master problem with dual valuable π:

$$\begin{aligned} \max \ & \sum_{i \in I} b_i \pi_i \\ \text{s.t.} \ & \sum_{i \in I} a_{ij} \pi_i \leq c_j && \forall j \in J' \\ & \pi_i \geq 0 && \forall i \in I. \end{aligned}$$

Denote by x^* and π^* optimal solutions to the primal and dual of the restricted master problem, respectively. A subproblem, also called a *pricing problem*, aims to search for a new column j with negative reduced cost to enter the column set J':

$$\min_{j \in J \setminus J'} \left\{ c_j - \sum_{i \in I} a_{ij} \pi_i^* \right\} \tag{3.2.1}$$

Note that a_{ij} is the variable in this pricing problem. Denote by j' the column corresponding to an optimal solution. If the optimal solution value of problem (3.2.1) is is less than zero, we add j' into J' and solve the restricted master problem with the updated column set J'; otherwise, the current optimal solution x^* is a feasible solution to the master problem. By using the LP duality theorem, this feasible solution x^* can be proven to be an optimal solution to the master problem. Observe that the total number of iterations is bounded by $|J|$, so convergence is assured.

3.3 Dantzig-Wolfe Decomposition

Dantzig-Wolfe Decomposition was known as a classical case using a column generation approach. Dantzig and Wolfe initially proposed Dantzig-Wolfe decomposition algorithm in 1960 for solving linear programming (LP) problems [27]. A significant set of practical problems in combinatorial optimization can be expressed as (integer) linear programming problems with sparse matrices, i.e., matrices such that a greater proportion of the elements are zero (see for example [54]). In modeling stage, many variables are defined but used only in a limited part of constraints. Thus, it is natural that we can formulate many problems in real-world applications with a block structure.

Given a constraint matrix

$$A = \{a_{ij} \mid \forall i \in I, \forall j \in J\},$$

we consider a pair of $(l+1)$-size partitions both on rows and columns denoted by $\{I_0, I_1, \dots, I_l\}$ and $\{J_0, J_1, \dots, J_l\}$, respectively, i.e.,

$$I = \bigcup_{k \in K \cup \{0\}} I_k \tag{3.3.2}$$

$$I_k \cap I_{k'} = \emptyset \qquad \forall k, k' \in K \cup \{0\} \text{ and } k \neq k' \tag{3.3.3}$$

$$J = \bigcup_{k \in \cup \{0\}} J_k \tag{3.3.4}$$

$$J_k \cap J_{k'} = \emptyset \qquad \forall k, k' \in K \cup \{0\} \text{ and } k \neq k', \tag{3.3.5}$$

where $K = \{1, 2, \ldots, l\}$. We can formulate a linear programming problem as

$$\min \sum_{j \in J} c_j x_j \tag{3.3.6}$$
$$\text{s.t.} \sum_{j \in J} a_{ij} x_j \geq b_i \qquad \forall i \in I_0 \tag{3.3.7}$$
$$\sum_{j \in J_k} a_{ij} x_j \geq b_i \qquad \forall i \in I_k,\ k \in K \tag{3.3.8}$$
$$x_j \geq 0 \qquad \forall j \in J. \tag{3.3.9}$$

Thus, the given constraint matrix are partitioned into blocks $A_{kk'} = \{a_{ij} \mid i \in I_k, j \in J_{k'}\}$ as visualized in Figure 3.1 without loss of generality. We call constraints (3.3.7) the linking

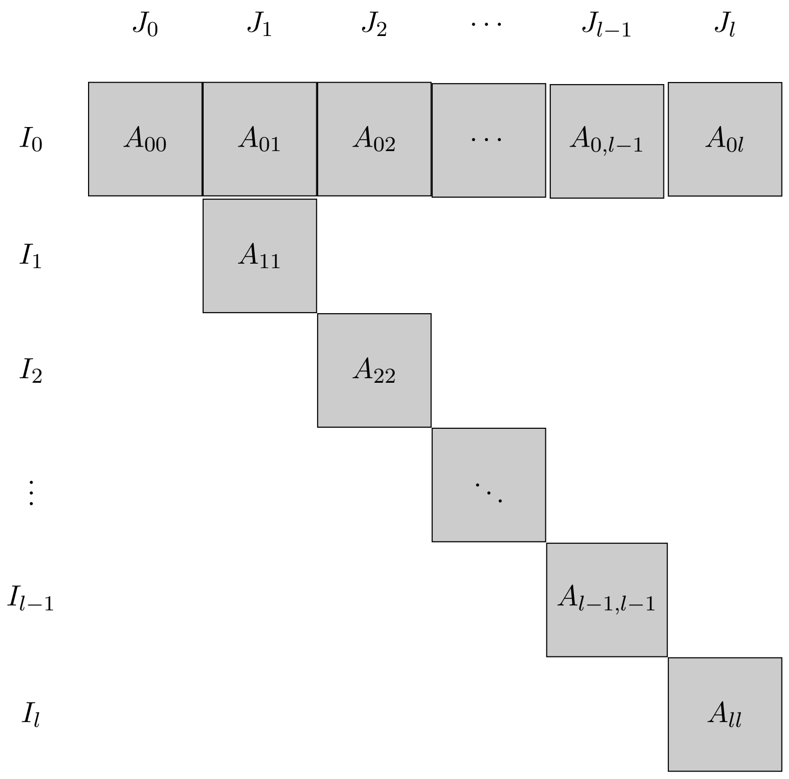


Figure 3.1: Block structure in Dantzig-Wolfe for LP

constraints. Notice that if we remove constraints (3.3.7) from the model (3.3.6)–(3.3.9), the problem is decomposed into l independent problems, i.e.,

$$\min \sum_{j \in J_k} c_j x_j \tag{3.3.10}$$

$$\sum_{j \in J_k} a_{ij} x_j \geq b_i \qquad \forall i \in I_k,\ k \in K \tag{3.3.11}$$

$$x_j \geq 0 \qquad \forall j \in J_k \tag{3.3.12}$$

for each $k \in \{1, 2, \ldots, l\}$. Let F be the feasible region (polyhedron) expressed by linear constraints (3.3.8). Denote by Q and R the set of extreme points and extreme rays of polyhedron F, respectively. Because of the convexity property [18], we know that any point $x \in F$ can be represented by a convex combination (Minkowski's representation)

$$x_j = \sum_{t\colon x^t \in Q} \lambda_t x_j^t + \sum_{t\colon x^t \in R} \mu_t x_j^t \qquad \forall j \in J \tag{3.3.13}$$

with

$$\sum_{t\colon x^t \in Q} \lambda_t = 1 \tag{3.3.14}$$

$$\lambda_t \geq 0 \qquad \forall x^t \in Q \tag{3.3.15}$$

$$\mu_t \geq 0 \qquad \forall x^t \in R. \tag{3.3.16}$$

Using (3.3.13), we can reformulate (3.3.6)–(3.3.9) as

$$\min \sum_{j \in J} c_j \left(\sum_{t\colon x^t \in Q} \lambda_t x_j^t + \sum_{t\colon x^t \in R} \mu_t x_j^t \right) \tag{3.3.17}$$

$$\text{s.t.} \sum_{j \in J} a_{ij} \left(\sum_{t\colon x^t \in Q} \lambda_t x_j^t + \sum_{t\colon x^t \in R} \mu_t x_j^t \right) \geq b_i \qquad \forall i \in I_0 \tag{3.3.18}$$

$$\sum_{t\colon x^t \in Q} \lambda_t = 1 \tag{3.3.19}$$

$$\lambda_t \geq 0 \qquad \forall x^t \in Q \tag{3.3.20}$$

$$\mu_t \geq 0 \qquad \forall x^t \in R. \tag{3.3.21}$$

Note that the variables in (3.3.17)–(3.3.21) are λ and μ. Thus, we can rearrange it as follows:

$$\min \sum_{t:\, x^t \in Q} \left(\sum_{j \in J} c_j x_j^t \right) \lambda_t + \sum_{t:\, x^t \in R} \left(\sum_{j \in J} c_j x_j^t \right) \mu_t \tag{3.3.22}$$

$$\text{s.t.} \sum_{t:\, x^t \in Q} \left(\sum_{j \in J} a_{ij} x_j^t \right) \lambda_t + \sum_{t:\, x^t \in R} \left(\sum_{j \in J} a_{ij} x_j^t \right) \mu_t \geq b_i \qquad \forall i \in I_0 \tag{3.3.23}$$

$$\sum_{t:\, x^t \in Q} \lambda_t = 1 \tag{3.3.24}$$

$$\lambda_t \geq 0 \qquad \forall x^t \in Q \tag{3.3.25}$$

$$\mu_t \geq 0 \qquad \forall x^t \in R. \tag{3.3.26}$$

Compared with (3.3.6)–(3.3.9), this reformulation reduced the number of constraints from $|I|$ to $|I_0|$, while the column size increased from $|J|$ to $|Q| + |R|$ which may be exponential to $|J|$. We call the form (3.3.22)–(3.3.22) the master problem, and consider a restricted master problem with a subset $Q' \subseteq Q$ and a subset $R' \subseteq R$ instead:

$$\min \sum_{t:\, x^t \in Q'} \left(\sum_{j \in J} c_j x_j^t \right) \lambda_t + \sum_{t:\, x^t \in R'} \left(\sum_{j \in J} c_j x_j^t \right) \mu_t \tag{3.3.27}$$

$$\text{s.t.} \sum_{t:\, x^t \in Q'} \left(\sum_{j \in J} a_{ij} x_j^t \right) \lambda_t + \sum_{t:\, x^t \in R'} \left(\sum_{j \in J} a_{ij} x_j^t \right) \mu_t \geq b_i \qquad \forall i \in I_0 \tag{3.3.28}$$

$$\sum_{t:\, x^t \in Q'} \lambda_t = 1 \tag{3.3.29}$$

$$\lambda_t \geq 0 \qquad \forall x^t \in Q' \tag{3.3.30}$$

$$\mu_t \geq 0 \qquad \forall x^t \in R'. \tag{3.3.31}$$

Let π and ρ be the dual variables for (3.3.28) and (3.3.29), respectively. The dual problem to this restricted master problem can be expressed as follows:

$$\max \sum_{i \in I_0} \pi_i b_i + \rho \tag{3.3.32}$$

$$\text{s.t.} \sum_{i \in I_0} \left(\sum_{j \in J} a_{ij} x_j^t \right) \pi_i + \rho \leq \sum_{j \in J} c_j x_j^t \qquad \forall x^t \in Q' \tag{3.3.33}$$

$$\sum_{i \in I_0} \left(\sum_{j \in J} a_{ij} x_j^t \right) \pi_i \leq \sum_{j \in J} c_j x_j^t \qquad \forall x^t \in R' \tag{3.3.34}$$

$$\pi_i \geq 0 \qquad \forall i \in I_0. \tag{3.3.35}$$

Let us define (π^*, ρ^*) as an optimal solution to the problem (3.3.32)–(3.3.35). The pricing problems based on π^* and ρ^* are defined as

$$\xi_Q = \min \sum_{j\in J}\left(c_j - \sum_{i\in I_0} a_{ij}\pi_i^*\right) x_j - \rho^* \tag{3.3.36}$$

$$\text{s.t. } x \in Q, \tag{3.3.37}$$

and

$$\xi_R = \min \sum_{j\in J}\left(c_j - \sum_{i\in I_0} a_{ij}\pi_i^*\right) x_j \tag{3.3.38}$$

$$\text{s.t. } x \in R. \tag{3.3.39}$$

Using the block structure defined in (3.3.2)–(3.3.5), the resulting pricing problems are composed of l independent LP problems, i.e.,

$$\min \sum_{j\in J_k}\left(c_j - \sum_{i\in I_0} a_{ij}\pi_i^*\right) x_j \tag{3.3.40}$$

$$\sum_{j\in J_k} a_{ij}x_j \geq b_i \qquad \forall i \in I_k,\ k \in K \tag{3.3.41}$$

$$x_j \geq 0 \qquad \forall j \in J_k \tag{3.3.42}$$

for $k \in \{1, 2, \ldots, l\}$. Due to the strong duality theorem [18], we then generate a new column based on the following cases. If one of the problems in (3.3.40)–(3.3.42) is unbounded, there exists an extreme ray $x^{t'} (\in R)$ with

$$\sum_{j\in J}\left(c_j - \sum_{i\in I_0} a_{ij}\pi_i^*\right) x_j^{t'} < 0 \tag{3.3.43}$$

(i.e., $\xi_R < 0$). In this case, we add $x^{t'}$ to R' and solve the updated restricted master problem. If all the problems (3.3.40)–(3.3.42) are bounded and

$$\xi_Q < 0, \tag{3.3.44}$$

we add to Q' an extreme point $x^{t'}$, an optimal solution to the problems in (3.3.36)–(3.3.37), and solved the updated restricted master problem. Otherwise, no extreme rays or extreme points can be a variable entering the basis, i.e., Dantzig-Wolfe decomposition is proved to solve the problem to optimality.

By reviewing Chapter 2, Dantzig-Wolfe decomposition can be viewed as a dual counterpart of the classical Benders decomposition [62]. Indeed, corresponding to the block structure depicted in Figure 3.1 for Dantzig-Wolfe Decomposition, Figure 3.2 shows the constraint matrix in the Benders decomposition for linear programming. However, this observation does not hold for logic-based Benders decomposition in Section 2.3.

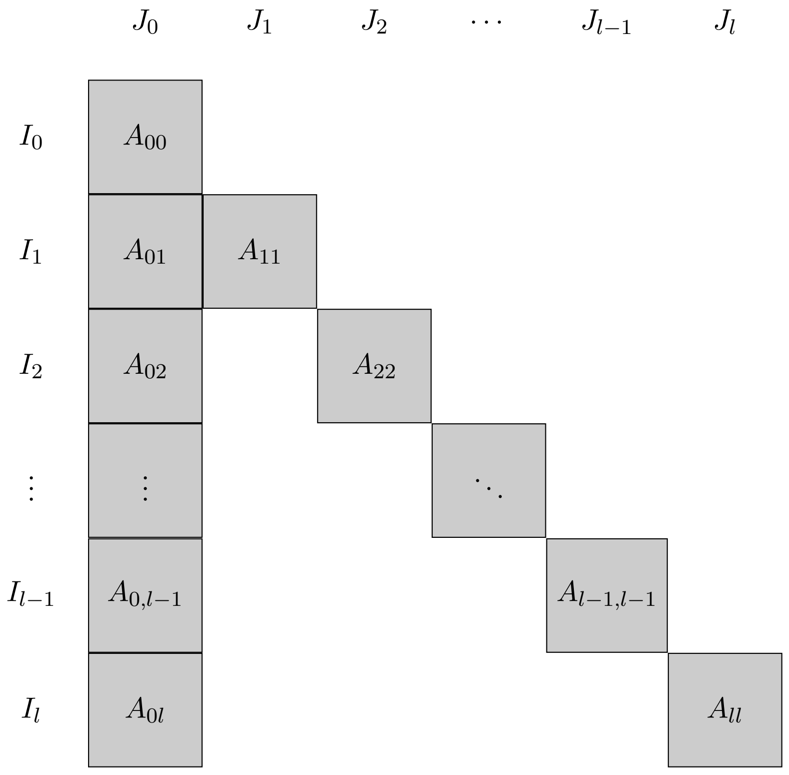


Figure 3.2: Block structure in Benders decomposition for LP

Chapter 4

Min-Max Regret Generalized Assignment Problem

Row generation using in robust optimization has attracted significant research efforts. In this chapter, we study a variant of the generalized assignment problem with uncertainty in input data. We propose a logic-based Benders decomposition approach, as well as several heuristic and exact algorithms.

Many optimization problems arising in real-world applications do not have accurate estimates of the problem parameters when the optimization decision is taken. Stochastic programming and robust optimization are two common approaches for the solution of optimization problems under uncertainty. The min-max regret criterion is one of the typical approaches for robust optimization. The *regret* is defined as the difference between the actual cost and the optimal cost that would have been obtained if a different solution had been chosen. The min-max regret approach is to minimize the worst-case regret. This criterion is not as pessimistic as the ordinary min-max approach, which looks for a solution with the best worst-case value across all scenarios.

The min-max regret version of a number of important combinatorial optimization problems has been recently studied, such as the traveling salesman problem [73], the assignment problem [83], the minimum spanning tree problem [24, 108, 110], the scheduling problems [8, 56], the shortest path problem [53], the facility location problem [6], the resource allocation problem [7, 110], the set covering problem [84], and the 0-1 knapsack problem [40]. Most of these problems are known to be NP-hard in the strong sense (see, e.g., [1] and [9]). In the literature, the solution of this kind of problems has been tackled with a number of exact and heuristic techniques. Concerning exact algorithms, logic-based Benders decomposition methods iteratively solve a relaxed master problem in which only a subset of scenarios is considered. Cuts corresponding to violated scenarios are computed by the solution of a slave problem and possibly added to the master problem, until a solution satisfying all scenarios is found (see, e.g., Montemanni [72]). Branch-and-cut methods extend the Logic-based Benders decomposition by including the solution of the slave problem at all nodes of the enumeration tree, and have been applied to several min-max regret problems (see Pereira and Averbakh [84]). Other authors used instead the structure of the min-max regret problem at hand to devise tailored combinatorial branch-and-bound algorithms, such as the one by Montemanni and Gambardella [74]. Concerning heuristic methods, several attempts have been made, including constructive heuristics based on the solution of a fixed scenario (Kasperski and Zielinski [56]) or on the inclusion of a dual relaxation component (Furini et al. [40]), as well as more elaborated metaheuristics such as genetic algorithms and filter-and-fan methods (Pereira and Averbakh [84]). For a deeper analysis of the existing literature, we refer the interested reader to Kouvelis and Yu [60], Kasperski [55], Aissi, Bazgan and Vanderpooten [2], and Candia-Véjar, Álvarez-Miranda, and Maculan [19].

In this chapter we consider the *generalized assignment problem* (GAP) with min-max regret criterion under interval costs. The classical GAP is an NP-hard combinatorial optimization problem [91] having many applications (see [38], [70], and [89]). The *interval min-max regret generalized assignment problem* (MMR-GAP) is a generalization of the GAP to the case where the cost coefficients are uncertain. We assume that every cost coefficient can take any value in a corresponding given interval, regardless of the values taken by the other cost coefficients. The problem requires to find a robust solution that minimizes the maximum regret. We prove that the decision version of the problem is Σ_2^p-complete. We computationally evaluate a heuristic algorithm for the MMR-GAP that solves the underlying GAP to optimality under a fixed scenario. We consider three scenarios: lowest cost, highest cost, and median cost. The median cost scenario leads to a solution of the MMR-GAP whose objective function value is within twice the optimal value. We also computationally evaluate a dual substitution heuristic based on a *mixed integer programming* (MIP) model obtained by replacing some constraints with the dual of their continuous relaxation.

We also examine three exact algorithmic approaches that iteratively solve the problem by only including a subset of scenarios. The first approach is based on logic-based Benders decomposition: it solves a MIP with incomplete scenarios and iteratively supplements the scenarios corresponding to violated constraints. The second approach is a basic branch-and-cut algorithm, in which the violated constraints are added at the nodes of a MIP enumeration tree. We then introduce a new Lagrangean-based branch-and-cut algorithm and enhance it through: (i) effective Lagrangian relaxations, to provide tighter lower bounds than those produced by the linear programming relaxation; (ii) an efficient variable fixing technique; (iii) a two-direction dynamic programming approach to effectively solve the Lagrangian subproblems. We evaluate the algorithms through computational experiments on different benchmarks.

4.1 Problem Description

4.1.1 Generalized Assignment Problem

The *generalized assignment problem* (GAP) is defined as follows. Given a set of n jobs $J = \{1, \ldots, n\}$ and a set of m agents $I = \{1, \ldots, m\}$, we look for a minimum cost assignment, subject to assigning each job to exactly one agent and satisfying a resource constraint for each agent. Assigning job j to agent i incurs a cost c_{ij} and consumes an amount a_{ij} of a resource, whereas the total amount of the resource available at agent i (agent capacity) is b_i.

A natural formulation of the GAP is defined over a two-dimensional binary variable x_{ij} indicating that job j is assigned to agent i if and only if $x_{ij} = 1$:

$$\min \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{4.1.1}$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij}x_{ij} \le b_i \qquad \forall i \in I \tag{4.1.2}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad \forall j \in J \tag{4.1.3}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in I,\ \forall j \in J. \tag{4.1.4}$$

For convenience, we define X_0 to be the set of all feasible solutions of the GAP, i.e.,

$$X_0 = \{x \mid x \text{ satisfies constraints (4.1.2)–(4.1.4)}\}. \tag{4.1.5}$$

4.1.2 Interval Min-Max Regret Generalized Assignment Problem

In this chapter we assume that the cost c_{ij} can take any value within a given range $[c^-_{ij}, c^+_{ij}]$. An array of costs c^s_{ij} satisfying $c^s_{ij} \in [c^-_{ij}, c^+_{ij}]$ $\forall i \in I,\ j \in J$, is called a *scenario* and is denoted by s. We define $z^s(x)$ to be the objective function value of solution $x \in X_0$ under scenario s:

$$z^s(x) = \sum_{i=1}^{m}\sum_{j=1}^{n} c^s_{ij}x_{ij}. \tag{4.1.6}$$

We denote by z^s_* the optimal solution value under scenario s, i.e., $z^s_* = \min_{y \in X_0} z^s(y)$. The *regret* $r^s(x)$ associated with solution x under scenario s is then the difference between these two values:

$$r^s(x) = z^s(x) - z^s_*. \tag{4.1.7}$$

Let S denote the set of all possible scenarios, i.e.,

$$S = \{s \mid c^s_{ij} \in [c^-_{ij}, c^+_{ij}] \ \forall i \in I, j \in J\}.$$

The *maximum regret* of a solution x is then the maximum $r^s(x)$ value over all scenarios:

$$r_{\max}(x) = \max_{s \in S} r^s(x). \tag{4.1.8}$$

The MMR-GAP is to find a feasible solution x such that the maximum regret is minimized:

$$\min_{x \in X_0} r_{\max}(x) = \min_{x \in X_0} \max_{s \in S} r^s(x) = \min_{x \in X_0} \max_{\substack{y \in X_0 \\ s \in S}} \left\{ \sum_{i=1}^{m} \sum_{j=1}^{n} c^s_{ij} x_{ij} - \sum_{i=1}^{m} \sum_{j=1}^{n} c^s_{ij} y_{ij} \right\}. \tag{4.1.9}$$

This formulation can be rewritten using the following classical general result that was proposed by Aissi et al. [2] (whose roots are in Yaman et al. [108]), which also applies to many other interval min-max problems.

Lemma 4.1.1. *The regret of a solution $x \in X_0$ is maximized under the following scenario $\sigma(x)$:*

$$c^{\sigma(x)}_{ij} = \begin{cases} c^+_{ij} & \textit{if } \ x_{ij} = 1 \\ c^-_{ij} & \textit{otherwise} \end{cases} \qquad \forall i \in I, \ \forall j \in J. \tag{4.1.10}$$

In other words, the value $r_{\max}(x)$ is achieved by the scenario that gives the worst costs to the job-agent pairs selected by the assignment x, and the best costs to the non-selected job-agent pairs. Since x_{ij} is a binary variable, $c^{\sigma(x)}_{ij}$ can also be written as

$$c^{\sigma(x)}_{ij} = c^-_{ij} + (c^+_{ij} - c^-_{ij}) x_{ij} \qquad \forall i \in I, \ \forall j \in J. \tag{4.1.11}$$

From Lemma 4.1.1, the maximum regret is achieved if we apply the worst scenario $\sigma(x)$ to s, and, from (4.1.9), the MMR-GAP can be rewritten as

$$\begin{aligned}
\min_{x \in X_0} r_{\max}(x) &= \min_{x \in X_0} \max_{\substack{y \in X_0 \\ s \in S}} \left\{ \sum_{i=1}^{m} \sum_{j=1}^{n} c^s_{ij} x_{ij} - \sum_{i=1}^{m} \sum_{j=1}^{n} c^s_{ij} y_{ij} \right\} \\
&= \min_{x \in X_0} \left\{ \sum_{i=1}^{m} \sum_{j=1}^{n} c^{\sigma(x)}_{ij} x_{ij} - \min_{y \in X_0} \sum_{i=1}^{m} \sum_{j=1}^{n} c^{\sigma(x)}_{ij} y_{ij} \right\} \\
&= \min_{x \in X_0} \left\{ \sum_{i=1}^{m} \sum_{j=1}^{n} c^+_{ij} x_{ij} - \min_{y \in X_0} \sum_{i=1}^{m} \sum_{j=1}^{n} c^{\sigma(x)}_{ij} y_{ij} \right\}.
\end{aligned} \tag{4.1.12}$$

We assume in the following that all input data are integers.

4.1.3 Complexity of the MMR-GAP

The decision version of the MMR-GAP is the problem of deciding whether there exists a feasible solution x such that the maximum regret of x is less than a given constant. In this section we consider complexity issues of the MMR-GAP and show that the decision version of the MMR-GAP is Σ_2^p-complete, where the class Σ_2^p is defined as follows. We have $\Sigma_0^p = \mathrm{P}$; for $k \geq 0$, $\Sigma_{k+1}^p = \mathrm{NP}^{\Sigma_k^p}$, where NP^X is the set of decision problems that are solvable in polynomial time by a non-deterministic Turing machine with an oracle for a problem of class X. Hence $\Sigma_1^p = \mathrm{NP}$, and Σ_2^p is the class of decision problems that are solvable in non-deterministic polynomial time with an oracle for a problem of class NP.

It is not hard to see that the MMR-GAP(0), a special case of the MMR-GAP that requires to find a solution whose maximum regret equals 0, is at least as hard as the classical GAP. Indeed a solution to a special case of the MMR-GAP(0) with $c_{ij}^+ = c_{ij}^- = c_{ij}$ for all $i \in I$ and $j \in J$ gives an optimal solution to the GAP, implying that GAP $\leq_{\mathrm{P}}$ MMR-GAP(0), where "A $\leq_{\mathrm{P}}$ B" signifies that problem A is polynomial time reducible to B. It is easy to show that the problem of finding a feasible solution to the GAP is NP-hard in the strong sense, as it can model (as mentioned in [104]), without introducing large numbers, the *bin packing problem*, which is known to be NP-hard in the strong sense [41]. The same holds for the MMR-GAP, because it has exactly the same feasible region X_0 as the GAP. Moreover, it is not hard to see by a similar reduction from bin packing that even if we restrict our attention to feasible instances, the GAP is still NP-hard in the strong sense, and so is the MMR-GAP(0). Accordingly, we have the following lemma.

Lemma 4.1.2. *The problem of finding a feasible solution to the MMR-GAP is NP-hard in the strong sense, and even if it is assumed that only feasible instances are given as input, the MMR-GAP(0) is still NP-hard in the strong sense.*

Since it is NP-hard to find a feasible solution, it is not possible (unless P = NP) for the GAP or the MMR-GAP to design a polynomial time approximation algorithm (for which it is required to find a feasible solution in polynomial time), and hence approximation algorithms with performance guarantees have only been proposed for variants of the GAP (see a survey in [104]). The fact that the MMR-GAP(0) is NP-hard for feasible instances rules out (under the assumption of P $\neq$ NP) the possibility of designing for the MMR-GAP, a polynomial time k-factor approximation algorithm for any finite k (≥ 1) even if feasible instances are assumed, where a k-factor approximation algorithm for the MMR-GAP guarantees to find a solution $\hat{x}$ that satisfies $r_{\max}(\hat{x}) \leq k r_{\max}(x)$ for all $x \in X_0$. This follows from the fact that if an optimal solution x^* to the MMR-GAP has the maximum regret of value 0, the solution $\hat{x}$ obtained by a k-factor approximation algorithm is optimal, because $r_{\max}(\hat{x}) \leq k r_{\max}(x^*) = 0$ holds.

In order to clarify the complexity of the MMR-GAP, consider an arbitrary instance of the interval min-max regret knapsack problem (MMR-KP): given a set $\bar{J}$ of $\bar{n}$ items with weights $w_1, \dots, w_{\bar{n}}$, profit ranges $[p_1^-, p_1^+], \dots, [p_{\bar{n}}^-, p_{\bar{n}}^+]$ and capacity d, find a solution that minimizes the maximum regret. Let $\bar{S}$ denote the set of all possible scenarios, i.e.,

$$\bar{S} = \{s \mid p_j^s \in [p_j^-, p_j^+] \ \forall j \in \bar{J}\},$$

and $\bar{x}$ denote binary variables

$$\bar{x}_j = \begin{cases} 1 & \text{if item } j \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \qquad \forall j \in \bar{J}. \tag{4.1.13}$$

The MMR-KP can be formulated as

$$\max_{\bar{x} \in X_1} \min_{s \in \bar{S}} \left\{ \max_{\bar{y} \in X_1} \sum_{j=1}^{\bar{n}} p_j^s \bar{y}_j - \sum_{j=1}^{\bar{n}} p_j^s \bar{x}_j \right\},$$

where

$$X_1 = \{\bar{x} \mid \sum_{j=1}^{\bar{n}} w_j \bar{x}_j \leq d \text{ and } \bar{x} \in \{0,1\}\}. \tag{4.1.14}$$

By using the worst-case lemma, MMR-KP can be written as

$$\min_{\bar{x} \in X_1} \left\{ \max_{\bar{y} \in X_1} \sum_{j=1}^{\bar{n}} (p_j^+ + (p_j^- - p_j^+)\bar{x}_j)\bar{y}_j - \sum_{j=1}^{\bar{n}} p_j^- \bar{x}_j \right\}. \tag{4.1.15}$$

Let $p_{\max}^+$ denote $\max_{j \in \bar{J}} p_j^+$. We construct an instance of the MMR-GAP as follows:

$$m = 2, \quad n = \bar{n}, \tag{4.1.16}$$

$$a_{1j} = w_j, \quad a_{2j} = 1, \qquad \forall j \in J, \tag{4.1.17}$$

$$b_1 = d, \quad b_2 = \bar{n},$$

$$c_{1j}^+ = p_{\max}^+ - p_j^-, \quad c_{1j}^- = p_{\max}^+ - p_j^+, \qquad \forall j \in J, \tag{4.1.18}$$

$$c_{2j}^+ = c_{2j}^- = p_{\max}^+, \qquad \forall j \in J. \tag{4.1.19}$$

Any feasible solution $\bar{x}$ of this MMR-KP instance can be transformed into a feasible solution x of the corresponding the MMR-GAP instance by

$$x_{1j} = \bar{x}_j, \ x_{2j} = 1 - \bar{x}_j, \ \forall j \in J, \tag{4.1.20}$$

and vice versa. From (4.1.16)–(4.1.19), for any solutions x and $\bar{x}$, and y and $\bar{y}$ that respectively satisfy (4.1.20) and for each j, we have

$$\begin{aligned}
&\sum_{i=1}^{2}(c_{ij}^- + (c_{ij}^+ - c_{ij}^-)x_{ij})y_{ij} \\
&= (p_{\max}^+ - p_j^+ - (p_j^- - p_j^+)x_{1j})y_{1j} + (p_{\max}^+ + 0 \cdot x_{2j})y_{2j} \\
&= (p_{\max}^+ - p_j^+ - (p_j^- - p_j^+)\bar{x}_j)\bar{y}_j + p_{\max}^+(1 - \bar{y}_j) \\
&= p_{\max}^+ - (p_j^+ + (p_j^- - p_j^+)\bar{x}_j)\bar{y}_j, \qquad (4.1.21)
\end{aligned}$$

$$\begin{aligned}
\sum_{i=1}^{2} c_{ij}^+ x_{ij} &= (p_{\max}^+ - p_j^-)x_{1j} + p_{\max}^+ x_{2j} \\
&= (p_{\max}^+ - p_j^-)\bar{x}_j + p_{\max}^+(1 - \bar{x}_j) \\
&= p_{\max}^+ - p_j^- \bar{x}_j. \qquad (4.1.22)
\end{aligned}$$

Then, by applying (4.1.21) and (4.1.22) to (4.1.12), together with the fact that for any two solutions $\bar{x}$ and x that satisfy (4.1.20), $\bar{x}$ is feasible to MMR-KP if and only if x is feasible to the corresponding instance of MMR-GAP, the maximum regret of every solution x to the above MMR-GAP instance is

$$\begin{aligned}
&\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}^+ x_{ij} - \min_{y \in X_0} \sum_{i=1}^{m}\sum_{j=1}^{n}(c_{ij}^- + (c_{ij}^+ - c_{ij}^-)x_{ij})y_{ij} \\
&= \bar{n}p_{\max}^+ - \sum_{j=1}^{\bar{n}} p_j^- \bar{x}_j - \min_{\bar{y} \in X_1} \sum_{j=1}^{\bar{n}}(p_{\max}^+ - (p_j^+ + (p_j^- - p_j^+)\bar{x}_j)\bar{y}_j) \\
&= \max_{\bar{y} \in X_1} \sum_{j=1}^{\bar{n}}(p_j^+ + (p_j^- - p_j^+)\bar{x}_i)\bar{y}_j - \sum_{j=1}^{\bar{n}} p_j^- \bar{x}_j, \qquad (4.1.23)
\end{aligned}$$

and the optimal value of this instance is

$$\begin{aligned}
&\min_{x \in X_0}\left\{\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}^+ x_{ij} - \min_{y \in X_0} \sum_{i=1}^{m}\sum_{j=1}^{n}(c_{ij}^- + (c_{ij}^+ - c_{ij}^-)x_{ij})y_{ij}\right\} \\
&= \min_{\bar{x} \in X_1}\left\{\max_{\bar{y} \in X_1} \sum_{j=1}^{\bar{n}}(p_j^+ + (p_j^- - p_j^+)\bar{x}_i)\bar{y}_j - \sum_{j=1}^{\bar{n}} p_j^- \bar{x}_j\right\}. \qquad (4.1.24)
\end{aligned}$$

The formulation in the last line of (4.1.24) is exactly the form of MMR-KP (4.1.15). This ensures that a solution of the MMR-GAP is optimal if and only if its corresponding solution transformed by (4.1.20) is an optimal solution to MMR-KP, and hence the MMR-GAP has the interval MMR-KP as special case. Using this reduction from MMR-KP, which is known to be Σ_2^p-hard [28], we then obtain the following.

Lemma 4.1.3. *The MMR-GAP is Σ_2^p-hard.*

The decision version of the MMR-GAP satisfies the condition of Theorem 7.4 in [41] for $k = 2$, which implies the following.

Lemma 4.1.4. *The decision version of the MMR-GAP lies in Σ_2^p.*

By combining Lemma 4.1.3 and 4.1.4, we can conclude with the following property.

Property 4.1.1. *The decision version of the MMR-GAP is Σ_2^p-complete.*

4.2 Heuristic Algorithms

In this section we present heuristic algorithms that, in view of the considerations of the previous section, are not guaranteed to run in polynomial time.

4.2.1 Fixed-Scenario Algorithm

We introduce a heuristic approach based on the observation that a feasible solution to an MMR-GAP instance can be obtained by fixing a scenario, solving the resulting GAP instance to optimality, and evaluating the maximum regret of the obtained solution using (4.1.12). This approach was used for other interval min-max regret problems (see, e.g., [56] and [84]).

We consider three scenarios: the lowest cost $c_{ij}^s = c_{ij}^-$, the highest cost $c_{ij}^s = c_{ij}^+$, and the median cost $c_{ij}^s = (c_{ij}^- + c_{ij}^+)/2$.

For the median-cost scenario, the following result is a special case of a more general result proved in [56].

Lemma 4.2.1. *Let $\widetilde{s}$ be the median-cost scenario, i.e., $c_{ij}^{\widetilde{s}} = (c_{ij}^- + c_{ij}^+)/2 \ \ \forall i \in I, \ \forall j \in J$, and let $\widetilde{x}$ be an optimal solution to the GAP under $\widetilde{s}$. Then, $r_{\max}(\widetilde{x}) \leq 2r_{\max}(x)$ holds for all $x \in X_0$.*

Proof. For any two solutions x, $y \in X_0$ and scenario $s \in S$, we have

$$z^s(x) - z^s(y) = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}^s (x_{ij} - y_{ij}) \tag{4.2.25}$$

from definition (4.1.6). Using (4.2.25) and definitions (4.1.7) and (4.1.8), the following inequality holds:

$$\begin{aligned} r_{\max}(y) &= \max_{s\in S}\{z^s(y) - z_*^s\} \\ &\geq \max_{s\in S}\{z^s(y) - z^s(x)\} \\ &= \max_{s\in S}\sum_{i=1}^m\sum_{j=1}^n c_{ij}^s(y_{ij} - x_{ij}) \\ &= \sum_{(i,j):\, y_{ij}>x_{ij}} c_{ij}^+(y_{ij} - x_{ij}) + \sum_{(i,j):\, y_{ij}<x_{ij}} c_{ij}^-(y_{ij} - x_{ij}). \end{aligned} \tag{4.2.26}$$

From (4.2.25), we can also obtain

$$z^{\sigma(x)}(x) = z^{\sigma(x)}(y) + \sum_{i=1}^m\sum_{j=1}^n c_{ij}^{\sigma(x)}(x_{ij} - y_{ij}) \tag{4.2.27}$$

$$= z^{\sigma(x)}(y) + \sum_{(i,j):\, x_{ij}>y_{ij}} c_{ij}^+(x_{ij} - y_{ij}) + \sum_{(i,j):\, x_{ij}<y_{ij}} c_{ij}^-(x_{ij} - y_{ij}), \tag{4.2.28}$$

where $\sigma(x)$ is the worst-case scenario defined by Lemma 4.1.1. Then, by subtracting $z_*^{\sigma(x)}$ from both sides of (4.2.27), we get

$$r_{\max}(x) = z^{\sigma(x)}(x) - z_*^{\sigma(x)} \tag{4.2.29}$$

$$= z^{\sigma(x)}(y) - z_*^{\sigma(x)} + \sum_{(i,j):\, x_{ij}>y_{ij}} c_{ij}^+(x_{ij} - y_{ij}) + \sum_{(i,j):\, x_{ij}<y_{ij}} c_{ij}^-(x_{ij} - y_{ij}). \tag{4.2.30}$$

By combining this with $r_{\max}(y) \geq z^{\sigma(x)}(y) - z_*^{\sigma(x)}$, we get

$$r_{\max}(x) \leq r_{\max}(y) + \sum_{(i,j):\, x_{ij}>y_{ij}} c_{ij}^+(x_{ij} - y_{ij}) + \sum_{(i,j):\, x_{ij}<y_{ij}} c_{ij}^-(x_{ij} - y_{ij}). \tag{4.2.31}$$

Now we consider the median-cost scenario $\widetilde{s}$. Since $\widetilde{x}$ is an optimal solution under $\widetilde{s}$, we have $z^{\widetilde{s}}(\widetilde{x}) \leq z^{\widetilde{s}}(y)$, and according to (4.2.25),

$$\sum_{i=1}^m\sum_{j=1}^n \frac{1}{2}(c_{ij}^+ + c_{ij}^-)(\widetilde{x}_{ij} - y_{ij}) = z^{\widetilde{s}}(\widetilde{x}) - z^{\widetilde{s}}(y) \leq 0 \tag{4.2.32}$$

holds, which is is equivalent to

$$\sum_{(i,j):\, y_{ij}>\widetilde{x}_{ij}} c_{ij}^+(y_{ij} - \widetilde{x}_{ij}) + \sum_{(i,j):\, y_{ij}<\widetilde{x}_{ij}} c_{ij}^-(y_{ij} - \widetilde{x}_{ij}) \tag{4.2.33}$$

$$\geq \sum_{(i,j):\, \widetilde{x}_{ij}>y_{ij}} c_{ij}^+(\widetilde{x}_{ij} - y_{ij}) + \sum_{(i,j):\, \widetilde{x}_{ij}<y_{ij}} c_{ij}^-(\widetilde{x}_{ij} - y_{ij}). \tag{4.2.34}$$

Then, by applying this to (4.2.26), we obtain

$$r_{\max}(y) \geq \sum_{(i,j):\, \widetilde{x}_{ij}>y_{ij}} c_{ij}^+(\widetilde{x}_{ij} - y_{ij}) + \sum_{(i,j):\, \widetilde{x}_{ij}<y_{ij}} c_{ij}^-(\widetilde{x}_{ij} - y_{ij}). \tag{4.2.35}$$

By combining (4.2.31) and (4.2.35), we finally obtain

$$r_{\max}(\widetilde{x}) \leq 2r_{\max}(y) \qquad \forall y \in X_0. \tag{4.2.36}$$

□

We next provide a tight example for the approximation ratio of Lemma 4.2.1.

Lemma 4.2.2. *There is an instance of the MMR-GAP for which an optimal solution to GAP under the median-cost scenario $\widetilde{s}$ has a regret twice as large as the optimal regret.*

Proof. Let $m = 3$, $n = 1$, $a_{11} = a_{21} = a_{31}$, $b_1 = 1$ with interval costs $c_{11} \in [1, 1]$, $c_{21} \in [0, 2]$, $c_{31} \in [0, 2]$. Assigning the unique job to any agent gives an optimal solution to GAP under the median-cost scenario, because all median costs have value 1. The optimal MMR-GAP solution assigns the job to agent 1, attaining the maximum regret of value 1, while assigning the job to agent 2 (or 3) gives the maximum regret of value 2. □

Recall that in Section 4.1.3 we derived Lemma 4.1.2 from the observation that GAP $\leq_{\mathrm{P}}$ MMR-GAP(0). It might be interesting to note that we can also show the opposite direction MMR-GAP(0) $\leq_{\mathrm{P}}$ GAP, because Lemma 4.2.1 implies that by solving the GAP under the median-cost scenario, we can obtain an optimal solution to the MMR-GAP(0) (for the reason discussed in the paragraph after Lemma 4.1.2).

4.2.2 Dual Substitution Heuristic

The dual substitution heuristic introduced in this section is based on a *mixed integer programming* (MIP) formulation in which some of the constraints are replaced by their dual counterpart in the linear relaxation of the problem. Kasperski provided a general technique in [55], and similar ideas of using MIP models have been used a number of times to produce exact algorithms for other min-max regret problems having zero duality gap, such as the min-max regret shortest problem [53], the min-max regret spanning tree problem [108]. To our knowledge, it is relatively new to use such ideas to design heuristic algorithms for problems with possibly non-zero duality gaps [40], and not much has been done to apply such ideas to various problems and computationally evaluate the resulting heuristics. In our research, we use it as a heuristic for the MMR-GAP.

The minimization problem over y in (4.1.12),

$$\min_{y \in X_0} \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}^{\sigma(x)} y_{ij}, \tag{4.2.37}$$

for every fixed x is an instance of the GAP. We consider the linear program obtained by replacing the integrality constraint $y_{ij} \in \{0, 1\}$ with the weaker requirement $y_{ij} \geq 0$ for all $i \in I$ and $j \in J$, i.e.,

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}^{\sigma(x)} y_{ij} \tag{4.2.38}$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} y_{ij} \leq b_i \qquad \forall i \in I \tag{4.2.39}$$

$$\sum_{i=1}^{m} y_{ij} = 1 \qquad \forall j \in J \tag{4.2.40}$$

$$y_{ij} \geq 0 \qquad \forall i \in I,\ \forall j \in J. \tag{4.2.41}$$

We introduce two types of dual variables: λ_i $(i \in I)$ for constraints (4.2.39) and μ_j $(j \in J)$ for constraints (4.2.40). The dual of (4.2.38)–(4.2.41) is then

$$\max -\sum_{i=1}^{m} b_i \lambda_i + \sum_{j=1}^{n} \mu_j \tag{4.2.42}$$

$$\text{s.t.} \ -a_{ij}\lambda_i + \mu_j \leq c_{ij}^{\sigma(x)} \qquad \forall i \in I,\ \forall j \in J \tag{4.2.43}$$

$$\lambda_i \geq 0 \qquad \forall i \in I. \tag{4.2.44}$$

By using (4.1.11) and embedding (4.2.42)–(4.2.44) into (4.1.12), we obtain the following dual substitution model (D-MMR-GAP):

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}^{+} x_{ij} + \sum_{i=1}^{m} b_i \lambda_i - \sum_{j=1}^{n} \mu_j \tag{4.2.45}$$

$$\text{s.t.} \ -a_{ij}\lambda_i + \mu_j \leq c_{ij}^{-} + (c_{ij}^{+} - c_{ij}^{-}) x_{ij} \qquad \forall i \in I, \forall j \in J \tag{4.2.46}$$

$$\lambda_i \geq 0 \qquad \forall i \in I \tag{4.2.47}$$

$$x \in X_0. \tag{4.2.48}$$

Intuitively, the D-MMR-GAP is not easier than the GAP, since it contains all the GAP constraints. In fact, the D-MMR-GAP has the following complexity:

Property 4.2.1. *Problem D-MMR-GAP is NP-hard in the strong sense.*

Proof. Given an instance of the classical GAP, we transform it to an instance of the D-MMR-GAP by considering intervals with no gap $c_{ij}^{+} = c_{ij}^{-} = c_{ij}$. Then, because the right-hand side of (4.2.46) becomes a constant value c_{ij}, the resulting D-MMR-GAP decomposes into two problems:

$$\text{(i)}\quad \min \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{4.2.49}$$

$$\text{s.t.}\quad x \in X_0; \tag{4.2.50}$$

$$\text{(ii)}\quad \min \sum_{i=1}^{m} b_i\lambda_i - \sum_{j=1}^{n} \mu_j \tag{4.2.51}$$

$$\text{s.t.}\quad -a_{ij}\lambda_i + \mu_j \leq c_{ij} \qquad \forall i \in I,\ \forall j \in J \tag{4.2.52}$$

$$\lambda_i \geq 0 \qquad \forall i \in I. \tag{4.2.53}$$

Note that these two problems are totally separated and the former problem is exactly the same as the given GAP instance. This implies that the GAP is polynomial-time reducible to the D-MMR-GAP, and in this reduction it is not necessary to introduce large numbers (i.e., the maximum among the absolute values of the coefficients c^+_{ij}, c^-_{ij}, a_{ij}, and b_i of the D-MMR-GAP instance is the same as that of the GAP instance). Since the GAP is known to be NP-hard in the strong sense, the same hods for the D-MMR-GAP. □

Property 4.2.2. *The optimal solution value of the D-MMR-GAP is an upper bound for the MMR-GAP.*

Proof. Let $\hat{X}$ denote the set of feasible solutions for the continuous relaxation of the GAP:

$$\hat{X} = \{y \mid y \text{ satisfies constraints (4.2.39)–(4.2.41)}\}.$$

From (4.1.5), we have $X_0 \subseteq \hat{X}$, and hence

$$\min_{x\in X_0}\left\{\sum_{i=1}^{m}\sum_{j=1}^{n} c^+_{ij}x_{ij} - \min_{y\in X_0}\sum_{i=1}^{m}\sum_{j=1}^{n} c^{\sigma(x)}_{ij}y_{ij}\right\} \tag{4.2.54}$$

$$\leq \min_{x\in X_0}\left\{\sum_{i=1}^{m}\sum_{j=1}^{n} c^+_{ij}x_{ij} - \min_{y\in \hat{X}}\sum_{i=1}^{m}\sum_{j=1}^{n} c^{\sigma(x)}_{ij}y_{ij}\right\}. \tag{4.2.55}$$

Note that, according to (4.1.12), the left side of the above inequality is the optimal value of the MMR-GAP.

Let $D(x)$ denote the set of feasible solutions of the dual problem in (4.2.42)–(4.2.44):

$$D(x) = \{(u, v) \mid (u, v) \text{ satisfies constraints (4.2.43)–(4.2.44)}\}.$$

Due to the strong duality theorem, the optimal values of the two problems (4.2.38)–(4.2.41) and (4.2.42)–(4.2.44) are the same, and hence

$$\min_{x\in X_0}\left\{\sum_{i=1}^{m}\sum_{j=1}^{n}c_{ij}^{+}x_{ij}-\min_{y\in\hat{X}}\sum_{i=1}^{m}\sum_{j=1}^{n}c_{ij}^{\sigma(x)}y_{ij}\right\} \tag{4.2.56}$$

$$=\min_{x\in X_0}\left\{\sum_{i=1}^{m}\sum_{j=1}^{n}c_{ij}^{+}x_{ij}-\max_{(u,v)\in D(x)}\left\{-\sum_{i=1}^{m}b_i\lambda_i+\sum_{j=1}^{n}\mu_j\right\}\right\}. \tag{4.2.57}$$

It is not hard to see that the D-MMR-GAP is equivalent to the right-hand side of equality (4.2.56). The thesis follows from (4.2.54) and (4.2.56). □

In addition, it easily follows that a tighter upper bound can be obtained as follows.

Property 4.2.3. *The upper bound obtained by evaluating the maximum regret of any optimal solution of the D-MMR-GAP is at least as good as the optimal value of the D-MMR-GAP.*

We observed in our experiments that the dual substitution heuristic tends to obtain better solutions compared to the fixed-scenario heuristic. However, unlike the fixed-scenario algorithm with the median-cost scenario, this algorithm cannot have a guarantee on its solution quality.

Property 4.2.4. *For any positive K, there exists an instance of the MMR-GAP for which the dual substitution heuristic obtains a solution whose maximum regret is at least K times the optimal value.*

Proof. Let $m = 2$, $n = 3$, $a_{11} = a_{21} = 1$, $a_{12} = a_{22} = 4$, $a_{13} = a_{23} = 2$, $b_1 = 5$, $b_2 = 3$ with interval costs $c_{11} \in [1, K_1]$, $c_{21} \in [1, 2]$, $c_{12} \in [K_2, K_2]$, $c_{22} = c_{13} = c_{23} \in [1, 1]$, where K_1 and K_2 satisfy $K_2 \gg K_1 \gg 1$ and $K_1 \geq K + 1$. Obviously, jobs 2 and 3 have to be assigned to agents 1 and 2, respectively, while job 1 can be assigned to either agent. There are only two feasible solutions: solution x assigning job 1 to agent 1 with maximum regret $K_1 - 1$, and the optimal solution y assigning job 1 to agent 2 with maximum regret 1. On the other hand, since we applied LP relaxation in the D-MMR-GAP, solution x with maximum regret $(3/4)(K_2 - 1)$ is better than y with maximum regret $(3/4)(K_2 - 1) + 1$ under the formulation (4.2.45)–(4.2.48), and hence the dual substitution heuristic outputs solution x. Therefore, the ratio of the obtained objective value to the optimal value is $K_1 - 1 \geq K$, which can be arbitrarily large. □

4.3 Exact Algorithms

The two exact algorithms examined in this section are both rooted from a MIP model of the MMR-GAP. By using Lemma 4.1.1, and introducing a new continuous variable φ, along with a constraint that forces φ to satisfy $\varphi \leq z_*^s \; \forall s \in S$, the MMR-GAP can be expressed by the following MIP model (MIP-MMR-GAP):

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}^{+} x_{ij} - \varphi \tag{4.3.58}$$

$$\text{s.t.} \quad \varphi \leq \sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij}^{-} + (c_{ij}^{+} - c_{ij}^{-}) x_{ij}) y_{ij} \qquad \forall y \in X_0 \tag{4.3.59}$$

$$x \in X_0. \tag{4.3.60}$$

4.3.1 Logic-Based Benders Decomposition

Benders' decomposition was originally proposed in [17]. Logic-based Benders decompositions are standard techniques that have been frequently used for the exact solution of min-max regret problems [40, 37, 72, 75, 73, 84, 97]. We discussed this technique in Section 2.3

Model (4.3.58)–(4.3.60) is hard to handle due to the exponential number of constraints (4.3.59). Let us define a *master problem* $P(X)$ as the relaxation of the MIP-MMR-GAP in which set X_0 in constraints (4.3.59) is replaced by a subset X of X_0:

$$\varphi \leq \sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij}^{-} + (c_{ij}^{+} - c_{ij}^{-}) x_{ij}) y_{ij} \qquad \forall y \in X. \tag{4.3.61}$$

We name the constraints (4.3.61) Benders' cuts. For an optimal solution (x^*, φ^*) to the current master problem $P(X)$, we define a *slave problem* $Q(x^*)$ as:

$$\min_{y \in X_0} \sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij}^{-} + (c_{ij}^{+} - c_{ij}^{-}) x_{ij}^{*}) y_{ij}. \tag{4.3.62}$$

Let $q(y)$ be the objective value of a solution y and let y^* be an optimal solution to $Q(x^*)$. If $q(y^*) < \varphi^*$ holds, then the specific constraint (4.3.59) induced by y^* is violated by the current optimal solution (x^*, φ^*) of $P(X)$, and it is called a Benders' cut. Whenever such a cut is found, the proposed algorithm adds the solution y^* to X, and solves the updated $P(X)$. The process is iterated until the algorithm finds a solution (x^*, φ^*) for which no violated constraint exists.

Since $P(X)$ is a relaxation of the MIP-MMR-GAP, the optimal solution value at each iteration is a valid lower bound on the optimal solution value of the original MMR-GAP, and hence the final solution, which does not violate any constraint (4.3.59), is an optimal solution to the MMR-GAP.

The choice of the Benders' cuts added to X at each iteration can have a strong influence on the overall performance. We start with set $X = \{\widetilde{x}\}$, where $\widetilde{x}$ is the optimal solution obtained by the fixed-scenario heuristic under the median-cost scenario. When set X contains exactly one Benders' cut, the optimal solution of $P(X)$ has the following properties.

Property 4.3.1. *If $X = \{y\}$ for any $y \in X_0$, then the optimal value of $P(X)$ cannot be positive.*

Proof. In $P(X)$, φ only appears in constraint (4.3.59) and in the objective function, where its value has to be maximized. Hence we have

$$\varphi = \sum_{i=1}^{m}\sum_{j=1}^{n}(c_{ij}^{-} + (c_{ij}^{+} - c_{ij}^{-})x_{ij})y_{ij}.$$

Accordingly, $P(X)$ can be written as

$$\min_{x \in X_0} \left\{ \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}^{+}x_{ij} - \sum_{i=1}^{m}\sum_{j=1}^{n}(c_{ij}^{-} + (c_{ij}^{+} - c_{ij}^{-})x_{ij})y_{ij} \right\} \tag{4.3.63}$$

$$= \min_{x \in X_0} \left\{ \sum_{(i,j):\, y_{ij}=0} c_{ij}^{+}x_{ij} + \sum_{(i,j):\, y_{ij}=1} c_{ij}^{-}(x_{ij} - 1) \right\}. \tag{4.3.64}$$

Then, the optimal value cannot be positive, because the objective value becomes zero when $x = y$. □

On the other hand, if set X consists of an optimal solution y to the GAP instance with a fixed scenario, we have the following property.

Property 4.3.2. *If $X = \{y\}$ for an optimal solution y to the GAP instance obtained by fixing the scenario to any $s \in S$, then the optimal value of $P(X)$ cannot be negative.*

Proof. We prove the thesis by contradiction. Assume that the optimal value is negative, and let x^* be an optimal solution of $P(X)$. Then, from (4.3.63), we get

$$\sum_{\substack{(i,j):\, y_{ij}=0,\\ x_{ij}^*=1}} c_{ij}^{+} < \sum_{\substack{(i,j):\, y_{ij}=1,\\ x_{ij}^*=0}} c_{ij}^{-}. \tag{4.3.65}$$

Since $c_{ij}^{-} \le c_{ij}^{s} \le c_{ij}^{+}$ for any i, j and s, we have

$$\sum_{\substack{(i,j):\, y_{ij}=0,\\ x_{ij}^*=1}} c_{ij}^{s} < \sum_{\substack{(i,j):\, y_{ij}=1,\\ x_{ij}^*=0}} c_{ij}^{s} \qquad \forall s \in S. \tag{4.3.66}$$

This inequality indicates that the objective value $z^s(x^*)$ of x^* is strictly better than that of y for the GAP instance with scenario s. This contradicts the assumption that y is an optimal solution to this GAP instance, which completes the proof. □

In our logic-based Benders decomposition approach, we start with a set X consisting of the solution $\widetilde{x}$ obtained by the fixed-scenario heuristic with the median-cost scenario. From Properties 4.3.1 and 4.3.2, the optimal value of $P(X)$ is zero when $X = \{\widetilde{x}\}$. The approach is outlined in Algorithm 3.

Algorithm 3 Logic-based Benders Decomposition

```
1: Solve the GAP under the median-cost scenario $\widetilde{s}$, and let $\widetilde{x}$ be the obtained optimal
   solution.
2: $X \leftarrow \{\widetilde{x}\}$.
3: repeat
4:     Solve $P(X)$ and let $(x^*, \varphi^*)$ be its optimal solution.
5:     Solve $Q(x^*)$ and let $y^*$ be its optimal solution.
6:     $X \leftarrow X \cup \{y^*\}$.
7: until $q(y^*) \geq \varphi^*$
8: return $x^*$.
```

4.3.2 A Branch-and-Cut Algorithm

Branch-and-cut is another basic approach widely applied as an exact algorithm for interval min-max regret problems [40, 73, 84]. Our second exact algorithm uses Benders' cuts in the context of a basic branch-and-cut framework. We define $\bar{P}(X)$ as the linear relaxation of $P(X)$:

$$\bar{P}(X) \quad \min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}^{+} x_{ij} - \phi \tag{4.3.67}$$

$$\text{s.t.} \quad \phi \leq \sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij}^{-} + (c_{ij}^{+} - c_{ij}^{-}) x_{ij}) y_{ij}, \qquad \forall y \in X \tag{4.3.68}$$

$$\sum_{j=1}^{n} a_{ij} x_{ij} \leq b_i, \qquad \forall i \in I \tag{4.3.69}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad \forall j \in J \tag{4.3.70}$$

$$x_{ij} \geq 0, \qquad \forall i \in I,\ \forall j \in J. \tag{4.3.71}$$

We solve it (with respect to the free variables) at each node of the search tree. Its optimal value is a lower bound for the corresponding partial problem. Since the boundaries of the cost intervals c^- and c^+ for all the instances are integral, we strengthen this bound by rounding it up. If it is not smaller than the incumbent solution value, then we prune the current node. Otherwise, we look for a violated constraint (4.3.59) by solving the slave problem $Q(x^*)$ for an optimal solution x^* of the current $\bar{P}(X)$. If such a violation is found, we add an optimal solution y^* for $Q(x^*)$ to the current set X, and solve the updated $\bar{P}(X)$. The process continues until no violated constraint (4.3.59) exists. When this occurs, if the current optimal solution x^* to $\bar{P}(X)$ is integral, we update the incumbent solution and terminate the current node. If instead it is fractional, a branching follows.

The general framework of branch-and-cut can be implemented in various ways. We maintain the set X for constraints (4.3.61) as follows. The search starts with the set X that only contains an optimal solution $\widetilde{x}$ to the GAP instance with the median-cost scenario $\widetilde{s}$. The cuts added to X at each active node are used throughout the entire computation of the branch-and-cut algorithm, i.e., they contribute to all other active nodes.

In the branching step, we branch on the most fractional variable, i.e., the one closest to 0.5, and we adopt a *depth-first search* strategy that chooses, as the next node to search, an active node at the maximum depth in the search tree. This tends to allow the upper bound to be improved quickly in the early phases of the algorithm.

This method is outlined in Algorithm 4, where U denotes the incumbent solution value.

The logic-based Benders decomposition approach of the previous section cannot provide a feasible solution until it terminates, while the branch-and-cut algorithm usually obtains feasible solutions before reaching optimality. Hence, branch-and-cut algorithms can also serve as heuristics by prematurely halting them.

4.4 Lagrangian-Based Branch-and-Cut Algorithm

To improve the performance of the basic branch-and-cut algorithm of the previous section, we introduce a Lagrangian-based branch-and-cut algorithm. A similar approach has been used in [40], but we make use of many new techniques tailored for the MMR-GAP. In particular, we propose a stronger lower bound, an efficient variable fixing method, and an effective solution of the Lagrangian subproblems through dynamic programming.

4.4.1 Best-First Search

The algorithm adopts a best-first search strategy. It is known that such strategy, which chooses for the next search an active node with the smallest lower bound, tends to minimize the number of partial problems generated during the search, and to quickly improve the lower bound. However, this often comes at the price of sacrificing a quick improvement of the upper bound. The lack of a good upper bound can also influence the overall efficiency as it may reduce the chance of terminating nodes due to a large optimality gap. In order to overcome this we can compute a good initial upper bound, e.g., by applying the heuristic

Algorithm 4 Basic Branch-and-Cut Algorithm

```
1: Initialize the global upper bound $U \leftarrow +\infty$.
2: Solve the GAP under scenario $\widetilde{s}$ and obtain an optimal solution $\widetilde{x}$.
3: $X \leftarrow \{\widetilde{x}\}$.
4: Initialize the set $A$ of active nodes (i.e., partial problems) with the root node that
   corresponds to the original problem.
5: **while** the set $A$ of active nodes is not empty **do**
6:    Select as the current node the active node that was most recently added to $A$, and
      remove it from $A$.
7:    **repeat**
8:       Solve the LP relaxation $\bar{P}(X)$ for the partial problem corresponding to the current
         node, and obtain an optimal solution $(x^*, \varphi^*)$.
9:       **if** the optimal value of $\bar{P}(X) \geq U$ **then**
10:         **go to** 5 (the current node is terminated).
11:      **end if**
12:      Solve $Q(x^*)$ to obtain its optimal solution $y^*$.
13:      **if** $q(y^*) < \varphi^*$ **then** $X \leftarrow X \cup \{y^*\}$.
14:   **until** $q(y^*) \geq \varphi^*$
15:   **if** $x^*$ is integral **then**
16:      $U \leftarrow$ optimal value of $\bar{P}(X)$.
17:      **go to** 5 (the current node is terminated).
18:   **end if**
19:   From the partial problem corresponding to the current node, generate two nodes
      corresponding to the partial problems obtained by fixing the most fractional free
      variable $x_{ij}$ to 0 or to 1, respectively, and add them to $A$.
20: **end while**
21: **return** $U$
```

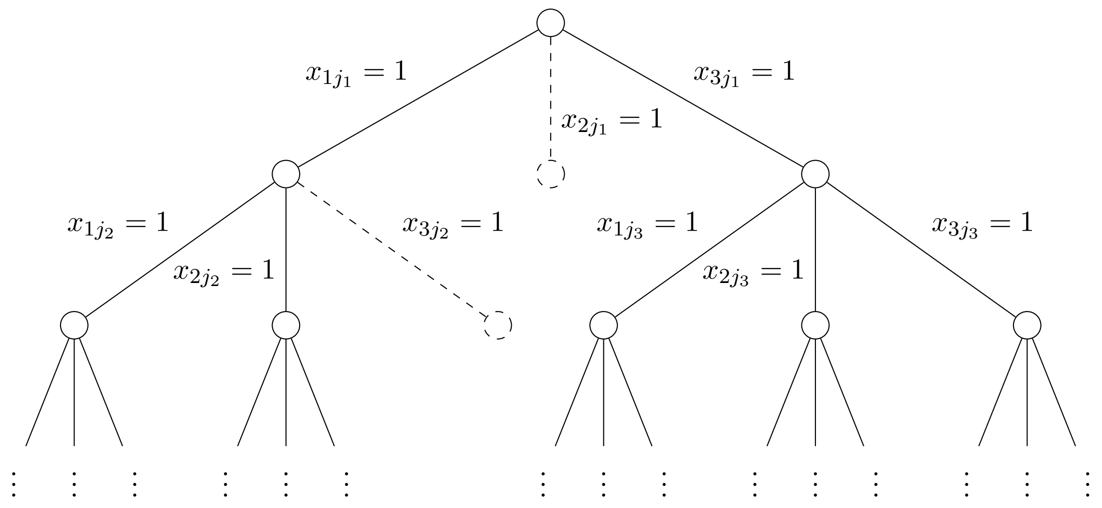


Figure 4.1: A branching tree for the MMR-GAP with $m = 3$

algorithms of Sections 4.2.1 and 4.2.2, which can provide good upper bounds for most of the instances, as it will be seen from the computational results in Section 4.5. Our branch-and-cut algorithm incorporates both the dual substitution heuristic and the fixed-scenario algorithm under the median-cost scenario. Each heuristic had a time limit of 300 seconds, and the best solution value was selected as the initial upper bound at the root node.

4.4.2 Branching Strategy Based on Semi-Assignment Constraints

In this section, we consider a branching strategy that exploits the structure of semi-assignment constraints (4.1.3). In the MMR-GAP, fixing a variable x_{ij} to one requires job j to be assigned to agent i and forbids x_{kj} to take the value one for all agents $k \neq i$. Based on this, we adopted the following branching rule: a partial problem branches to at most m partial problems by fixing the assignment of an unfixed job j to each possible agent $i \in I$, except for those agents i whose remaining capacity is less than a_{ij} or those for which x_{ij} has already been fixed by the variable fixing rules described in Section 4.4.4. A branching tree for the case with $m = 3$ is illustrated in Figure 4.1, in which the branches in dashed lines represent the nodes that are not generated due to earlier variable fixing or violation of capacity constraint.

We choose a job j for branching by utilizing the information from the lower bound computation. Let μ_j^* denote the optimal value of the dual variable associated with the semi-assignment constraints (4.1.3), which is obtained by solving $\bar{P}(X)$, the linear relaxation of the master problem $P(X)$, with respect to the free variables. Indeed, solving

$\bar{P}(X)$ also provides an optimal solution to its dual $\bar{D}(X)$:

$$\max \sum_{j=1}^{n} \mu_j - \sum_{i=1}^{m} b_i \lambda_i - \sum_{y^s \in X} \left(\nu^s \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}^{-} y_{ij}^{s} \right) \tag{4.4.72}$$

$$\text{s.t.} \quad \sum_{y^s \in X} \left(c_{ij}^{+} - c_{ij}^{-} \right) y_{ij}^{s} \nu^s - a_{ij} \lambda_i + \mu_j \le c_{ij}^{+} \qquad \forall i \in I,\ \forall j \in J \tag{4.4.73}$$

$$\sum_{y^s \in X} \nu^s = 1 \tag{4.4.74}$$

$$\lambda_i \ge 0 \qquad \forall i \in I \tag{4.4.75}$$

$$\nu^s \ge 0 \qquad \forall y^s \in X, \tag{4.4.76}$$

where λ, μ, and ν are the dual variables associated with the capacity constraints (4.1.2), the semi-assignment constraints (4.1.3), and the Benders' constraints (4.3.59), respectively. For branching, we select the job $j \in J$ with the highest $|\mu_j^*|$ among those whose assignment has not been fixed yet. For this job j, a branch is considered for each agent that has sufficient remaining capacity to receive it, thus creating up to m child nodes. The highest $|\mu_j^*|$ value intuitively indicates that the corresponding semi-assignment constraint is critical, and hence it is expected that the LP lower bound of the current node can be strengthened after such fixing.

It is not hard to see that the resulting branching scheme reduces the number of nodes in the entire enumeration tree from $O(2^{mn})$ of the basic 0-1 branching to $O(m^n)$. (Note however that, in our case, the number of nodes that can actually be generated will not be that different, because the rule to choose the most fractional variable prevents the 0-1 branching from generating those nodes in which the semi-assignment constraints are violated.)

4.4.3 Lagrangian-Based Lower Bound

In this section we propose an improved lower bound computation based on Lagrangian relaxation. Other studies on min-max regret problems also applied Lagrangian methods (see [34] and [40]). By embedding constraints (4.1.3) and (4.3.61) in the objective function (4.3.58) through Lagrangian multipliers μ_j and ν^s, respectively, we get the following

Lagrangian relaxation $L(X, \mu, \nu)$:

$$L(X,\mu,\nu) = \min \left\{ \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}^{+}x_{ij} - \varphi + \sum_{j=1}^{n}\mu_j \left(1 - \sum_{i=1}^{m} x_{ij}\right) \right.$$

$$\left. + \sum_{y^s\in X} \nu^s \left(\varphi - \sum_{i=1}^{m}\sum_{j=1}^{n}(c_{ij}^{-} + (c_{ij}^{+} - c_{ij}^{-})x_{ij})y_{ij}^{s}\right)\right\} \tag{4.4.77}$$

s.t. (4.1.2) and (4.1.4).

We use the values of μ and ν in an optimal solution to $\bar{D}(X)$ as the Lagrangian multipliers for (4.4.77). Then we have $\sum_{y^s\in X}\nu^s = 1$ according to (4.4.74), and hence the objective function (4.4.77) of $L(X, \mu, \nu)$ can be rewritten as follows:

$$\min \sum_{i=1}^{m}\sum_{j=1}^{n} \hat{c}_{ij}x_{ij} + \sum_{j=1}^{n}\mu_j - \sum_{y^s\in X}\left(\nu^s \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}^{-}y_{ij}^{s}\right) \tag{4.4.78}$$

where

$$\hat{c}_{ij} = c_{ij}^{+} - \mu_j - \sum_{y^s\in X}\left(c_{ij}^{+} - c_{ij}^{-}\right) y_{ij}^{s}\nu^s. \tag{4.4.79}$$

Note that $L(X, \mu, \nu)$ is independent from φ, and its optimal solution can be obtained by solving m 0-1 knapsack problems in the x variables. To solve such knapsack problems, we use a dynamic programming algorithm, introduced in Section 4.4.5.

It is not hard to show that $L(X, \mu, \nu)$ provides a lower bound at least as good as the LP lower bound of Section 4.3.2 when we use the values of μ and ν in an optimal solution to the dual $\bar{D}(X)$ of problem $\bar{P}(X)$.

At each active node, we first obtain a lower bound by solving $\bar{P}(X)$. If it is lower than the incumbent solution value, then a round of Benders' cut additions is performed. When no violated constraint (4.3.59) exists, we solve $L(X, \mu, \nu)$ by setting μ and ν to the values they have in an optimal solution to $\bar{D}(X)$. If the resulting lower bound is not smaller than the incumbent solution value, the node is fathomed.

4.4.4 Variable Fixing

An advantage of Lagrangian relaxation is that the obtained information can also be used for variable fixing in an efficient way. Let U be the incumbent solution value and $l^*(X, \mu, \nu)$ the optimal solution value of $L(X, \mu, \nu)$. We denote by Δ, the optimality gap at the current node:

$$\Delta = U - l^*(X, \mu, \nu). \tag{4.4.80}$$

For obtaining $l^*(X, \mu, \nu)$, recall that the Lagrangian relaxation decomposes into m independent 0-1 knapsack problems, one for each agent $i \in I$:

$$L_i(X, \mu, \nu) \quad \min \sum_{j=1}^{n} \hat{c}_{ij} x_{ij} \tag{4.4.81}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_{ij} \leq b_i \tag{4.4.82}$$

$$x_{ij} \in \{0, 1\} \qquad \forall j \in J. \tag{4.4.83}$$

For a given set of multipliers μ, ν and Benders' cut set X, let $l_i^*(X, \mu, \nu)$ denote the optimal solution value of $L_i(X, \mu, \nu)$. Using (4.4.81)–(4.4.83), the Lagrangian function (4.4.78) can be rewritten as

$$l^*(X, \mu, \nu) = \sum_{j=1}^{n} \mu_j - \sum_{y^s \in X} \left(\nu^s \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}^- y_{ij}^s \right) + \sum_{i=1}^{m} l_i^*(X, \mu, \nu). \tag{4.4.84}$$

Now, let $\breve{x}$ be an optimal solution of $L(X, \mu, \nu)$ and $l_i^*(X, \mu, \nu)_{x_{ij}=1-\breve{x}_{ij}}$ be the optimal value of the knapsack problem for agent i in which we force x_{ij} to take value $1 - \breve{x}_{ij}$. The increase Ξ_{ij} on lower bound $l^*(X, \mu, \nu)$ can be written as follows when we force x_{ij} to take value $1 - \breve{x}_{ij}$:

$$\Xi_{ij} = \begin{cases} l_i^*(X, \mu, \nu)_{x_{ij}=0} - l_i^*(X, \mu, \nu) & \text{if } \breve{x}_{ij} = 1 \\ l_i^*(X, \mu, \nu)_{x_{ij}=1} - l_i^*(X, \mu, \nu) & \text{if } \breve{x}_{ij} = 0 \end{cases} \qquad \forall i \in I, \ \forall j \in J. \tag{4.4.85}$$

With the definitions above, we can consider many variable fixing rules:

- Rule 1: If $\breve{x}_{ij} = 0$ and $\Xi_{ij} \geq \Delta$, then x_{ij} can be fixed to 0.
- Rule 2: If $\breve{x}_{ij} = 1$ and $\Xi_{ij} \geq \Delta$, then x_{ij} can be fixed to 1 and x_{kj} to 0 for all $k \neq i$.
- Rule 3: This is a rule strengthened from Rule 1. Suppose that $\breve{x}_{ij} = 0$. If x_{ij} took the value 1 in a feasible solution, then all other x_{kj} would have to take the value 0; therefore if

  $$\Xi_{ij} + \sum_{k:\, \breve{x}_{kj}=1} \Xi_{kj} \geq \Delta,$$

  then x_{ij} can be fixed to 0.
- Rule 4: Suppose that $\breve{x}_{ij} = 1$ and $\breve{x}_{kj} = 0$ for all $k \neq i$. If x_{ij} took the value 0 in a feasible solution, then some x_{kj} with $k \neq i$ would have to take value 1; therefore if

  $$\Xi_{ij} + \min \{\Xi_{kj} \mid k \neq i\} \geq \Delta,$$

  then x_{ij} can be fixed to 1 and x_{kj} can be fixed to 0 for all $k \neq i$.

- Rule 5: If x_{kj} is fixed to 0 for all $k \neq i$ then x_{ij} must be fixed to 1. Moreover, if $\breve{x}_{ij} = 0$ and $\Xi_{ij} \geq \Delta$, then the current node can be pruned.
- Rule 6: Suppose that $\sum_{i=1}^{m} \breve{x}_{ij} = 0$. At least one unfixed variable x_{ij} for some $i \in I$ must take the value 1; therefore if
$$\min \{\Xi_{ij} \mid i \in I\} \geq \Delta,$$
then the current node can be pruned.
- Rule 7: If x_{ij} is fixed to 0 for all $i \in I$ then the current node can be pruned.
- Rule 8: Suppose that $\breve{x}_{ij} = 1$ for more than one $i \in I$. As only one of them can take the value 1, if
$$\sum_{k:\, \breve{x}_{kj}=1} \Xi_{kj} - \max \{\Xi_{kj} \mid \breve{x}_{kj} = 1, k \in I\} \geq \Delta,$$
then the current node can be pruned.

Rules 1–7 have been proposed by Posta et al. [85] for the classical GAP. Preliminary computational experiments confirmed that they are also effective for the MMR-GAP, and hence they were incorporated in our variable fixing step.

4.4.5 Dynamic Programming Approach

As all input data are integers, the knapsack problems (4.4.81)–(4.4.83), which are needed to compute $l^*(X, \mu, \nu)$ as in (4.4.84), can be solved through the following two-direction dynamic programming approach, which was first proposed for the classical GAP by Posta et al. [85].

For each agent i, we introduce a quantity $f_i(j, k)$ for $j = 0, 1, \ldots, n$ and $k = 0, 1, \ldots, b_i$, where j is the number of jobs and k is an amount of resource. The value $f_i(j, k)$ gives the minimum cost when only jobs from 1 to j are available, and the resource limit is k instead of b_i in (4.1.2). We can compute $f_i(j, k)$ through the classical dynamic programming recursion

$$f_i(j,k) = \begin{cases} 0 & \text{if } \; j = 0; \\ f_i(j-1,k) & \text{if } \; j \geq 1 \text{ and } k < a_{ij}; \\ \min \{f_i(j-1,k), f_i(j-1,k-a_{ij}) + \hat{c}_{ij}\} & \text{otherwise.} \end{cases}$$

The computation can be implemented using an $(n+1) \times (b_i+1)$ array whose (j,k)-element contains the value of $f_i(j,k)$, and computing their values by increasing j. The value of

$f_i(n, b_i)$ gives the optimal value $l_i^*(X, \mu, \nu)$. We call this dynamic programing a *head-to-tail* approach.

In order to efficiently determine on variable fixing, we additionally use a *tail-to-head* approach for each knapsack problem. For each agent i, we define $g_i(j, k)$ as the minimum cost when we are only allowed to use jobs from j to n, and we have a resource restriction of $b_i - k$. Values $g_i(j, k)$ are computed in a symmetric way, by decreasing j:

$$g_i(j,k) = \begin{cases} 0 & \text{if } \ j = n+1; \\ g_i(j+1,k) & \text{if} j \leq n \text{ and } k > b_i - a_{ij}; \\ \min\{g_i(j+1,k), g_i(j+1,k+a_{ij}) + \hat{c}_{ij}\} & \text{otherwise.} \end{cases}$$

Next, we show how this DP approach also provides $l_i^*(X, \mu, \nu)_{x_{ij}=1-\breve{x}_{ij}}$ from the two DP tables, i.e., the table of $f_i(j, k)$ and that of $g_i(j, k)$. We compute $l_i^*(X, \mu, \nu)_{x_{ij}=1-\breve{x}_{ij}}$ by considering the combination of two partial knapsack problems for each k: a knapsack problem on jobs from 1 to $j-1$ with capacity k and another one on jobs from $j+1$ to n with capacity $b_i - a_{ij}(1 - \breve{x}_{ij}) - k$. Formally, when $n \geq 2$, $l_i^*(X, \mu, \nu)_{x_{ij}=1-\breve{x}_{ij}}$ can be obtained by

$$l_i^*(X,\mu,\nu)_{x_{ij}=0} = \min_{0 \leq k \leq b_i} \begin{cases} g_i(j+1,k) & \text{if } \ j = 1; \\ f_i(j-1,k) & \text{if } \ j = n; \\ f_i(j-1,k) + g_i(j+1,k) & \text{otherwise} \end{cases} \tag{4.4.86}$$

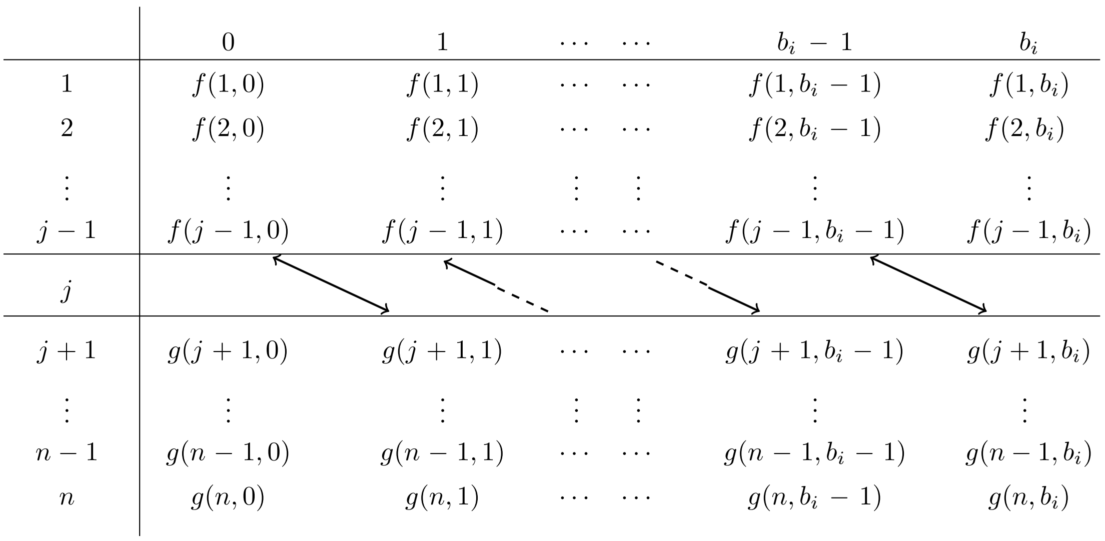
or

$$l_i^*(X,\mu,\nu)_{x_{ij}=1} = \min_{0 \leq k \leq b_i - a_{ij}} \begin{cases} \hat{c}_{ij} + g_i(j+1,k+a_{ij}) & \text{if } \ j = 1; \\ \hat{c}_{ij} + f_i(j-1,k) & \text{if } \ j = n; \\ \hat{c}_{ij} + f_i(j-1,k) + g_i(j+1,k+a_{ij}) & \text{otherwise.} \end{cases} \tag{4.4.87}$$

The computation of (4.4.86) and (4.4.87) can be done in $O(b_i)$ time for each pair (i, j), and hence, this two-direction dynamic programming approach has time complexity $O(nb_i)$ for each i, i.e., $O(n \sum_{i=1}^m b_i)$ in total to compute Ξ_{ij} for all i and j. Figure 4.2 illustrates the case in which we fix x_{ij} to 1 and the value of a_{ij} is 1. The upper and lower parts represent the tables for $f_i(j, k)$ and $g_i(j, k)$, and the lines in the jth column connect the two elements in the table corresponding to the two terms in the last line of (4.4.87). Then, the value of $l_i^*(X, \mu, \nu)_{x_{ij}=1}$ can be obtained by searching for the minimum among the sums of the values at the two ends.

4.4.6 Benders' Cuts Management

In our Lagrangian-based branch-and-cut approach, it takes only polynomial or pseudo-polynomial time to solve the LP relaxation $\bar{P}(X)$ or the Lagrangian relaxation problems, while the



	0	1	$\cdots$	$\cdots$	$b_i - 1$	b_i
1	$f(1,0)$	$f(1,1)$	$\cdots$	$\cdots$	$f(1,b_i - 1)$	$f(1,b_i)$
2	$f(2,0)$	$f(2,1)$	$\cdots$	$\cdots$	$f(2,b_i - 1)$	$f(2,b_i)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$j-1$	$f(j-1,0)$	$f(j-1,1)$	$\cdots$	$\cdots$	$f(j-1,b_i - 1)$	$f(j-1,b_i)$
j						
$j+1$	$g(j+1,0)$	$g(j+1,1)$	$\cdots$	$\cdots$	$g(j+1,b_i - 1)$	$g(j+1,b_i)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n-1$	$g(n-1,0)$	$g(n-1,1)$	$\cdots$	$\cdots$	$g(n-1,b_i - 1)$	$g(n-1,b_i)$
n	$g(n,0)$	$g(n,1)$	$\cdots$	$\cdots$	$g(n,b_i - 1)$	$g(n,b_i)$

Figure 4.2: Two-direction dynamic programing for fixing $x_{ij} = 1$, where $a_{ij} = 1$

slave problem $Q(\cdot)$ is NP-hard in the strong sense. We handle such difficulty by reducing the computation time needed to solve each $Q(\cdot)$, and by limiting the number of times we solve it. We consider the following methods.

Full-cut

This approach generates, at every node, as many Benders' cuts as possible, and adds them to X.

Cut-and-branch

At the root node we repeat the process of adding Benders' cuts to X until no such cut can be found. For the other nodes, we solve the LP relaxation $\bar{P}(X)$ first, and continue our Benders' cut generation only if in the optimal solution (x^*, φ^*) to $\bar{P}(X)$, x^* is integral. Whenever a new Benders' cut is found, we add it to X and we solve the updated $\bar{P}(X)$ again. This process is repeated until the obtained x^* becomes fractional or we find a solution (x^*, φ^*) for which no constraint is violated. Note that we can terminate an active node if we find an optimal solution (x^*, φ^*) to $\bar{P}(X)$ such that x^* is integral and no violated constraint exists, because such a solution is also optimal to problem $P(X_0)$ (with respect to the free variables at the node). The rule for non-root nodes is adopted for this reason. In this approach, the number of solved $Q(\cdot)$ is drastically reduced with respect to the *full-cut* approach.

Adaptive generation

We now introduce two techniques to improve the performance of the cut-and-branch method. The cut-and-branch strategy concentrates on lowering the number of times cuts are generated, which is shown to be more effective than the full-cut approach by the computational experiments of Section 4.5. However, this comes at the expense of sacrificing the quality of lower bounds due to the lack of effective Benders' cuts. That is, there is a trade-off between the quality of lower bounds and the time to generate Benders' cuts. A simple idea to reduce the time for generating Benders' cuts without sacrificing the quality of the lower bound (i.e., without reducing the number of Benders' cuts) is to shorten the time for solving $Q(\cdot)$ by using heuristic approaches instead of exact ones. Note that even if we use heuristics to solve $Q(\cdot)$, the lower bounds are valid and hence the resulting branch-and-cut algorithm remains exact. It should also be noted, however, that when integral solutions of $\bar{P}(X)$ are obtained, they are not necessarily feasible with respect to the MIP-MMR-GAP (4.3.58)–(4.3.60) unless $Q(\cdot)$ is solved to optimality. Thus, we solve $Q(\cdot)$ exactly only if integral solutions of $\bar{P}(X)$ are obtained, and use heuristics otherwise. We tested two heuristics, originally developed for the classical GAP: an ejection chain approach [105] and a path relinking approach with ejection chains [106], with a limit T (a parameter) on the number of calls to EC probe, which is the basic local search component of these heuristic algorithms. If no violated Benders' constraint is found within T iterations, a branching follows. On the basis of preliminary computational experiments, we adopted the ejection chain approach.

The second improvement is the method we use for adding Benders' cuts at non-root nodes. For this approach, we impose an upper limit on the total number of Benders' cuts generated at non-root nodes, and we skip the cut generation once this number was been reached (except for the case where x^* is integral). The upper limit was set to W times the number of Benders' cuts generated at the root (W a parameter). Moreover, for each non-root node, we stop the Benders' cut generation process when the ratio

$$\alpha = \frac{\varphi^* - q(y^*)}{q(y^*)} \times 100$$

becomes smaller than or equal to a parameter Υ (except for the case where x^* is integral), where (x^*, φ^*) is an optimal solution to $\bar{P}(X)$ and y^* is a (not necessarily optimal) solution to the slave problem $Q(x^*)$ defined by (4.3.62). This rule is motivated by the phenomena observed in Figure 4.3, in which the y-axis represents the percentage increase Δ_{LB} in the LP lower bound after adding every Benders' cut, and the x-axis represents the percentage gap between the value of ϕ^* and the optimal value $q(y^*)$ of the corresponding slave problem. Figure 4.3 shows how Δ_{LB} changes as Benders' cuts are added at an active node of some representative instances, where points corresponding to two consecutive iterations are connected. We can observe that high increments in the lower bound mostly occur at early iterations, when $\alpha > 2\%$, while most cuts with $\alpha \leq 1\%$ only produce a tiny increase of the node lower bound. This suggests that it is possible to save computational effort by halting the generation of Benders' cuts when α becomes small. It turned out that the adaptive generation of Benders' cuts performs well with parameter setting $T = 100$, $W = 8$, and $\Upsilon = 1\%$.

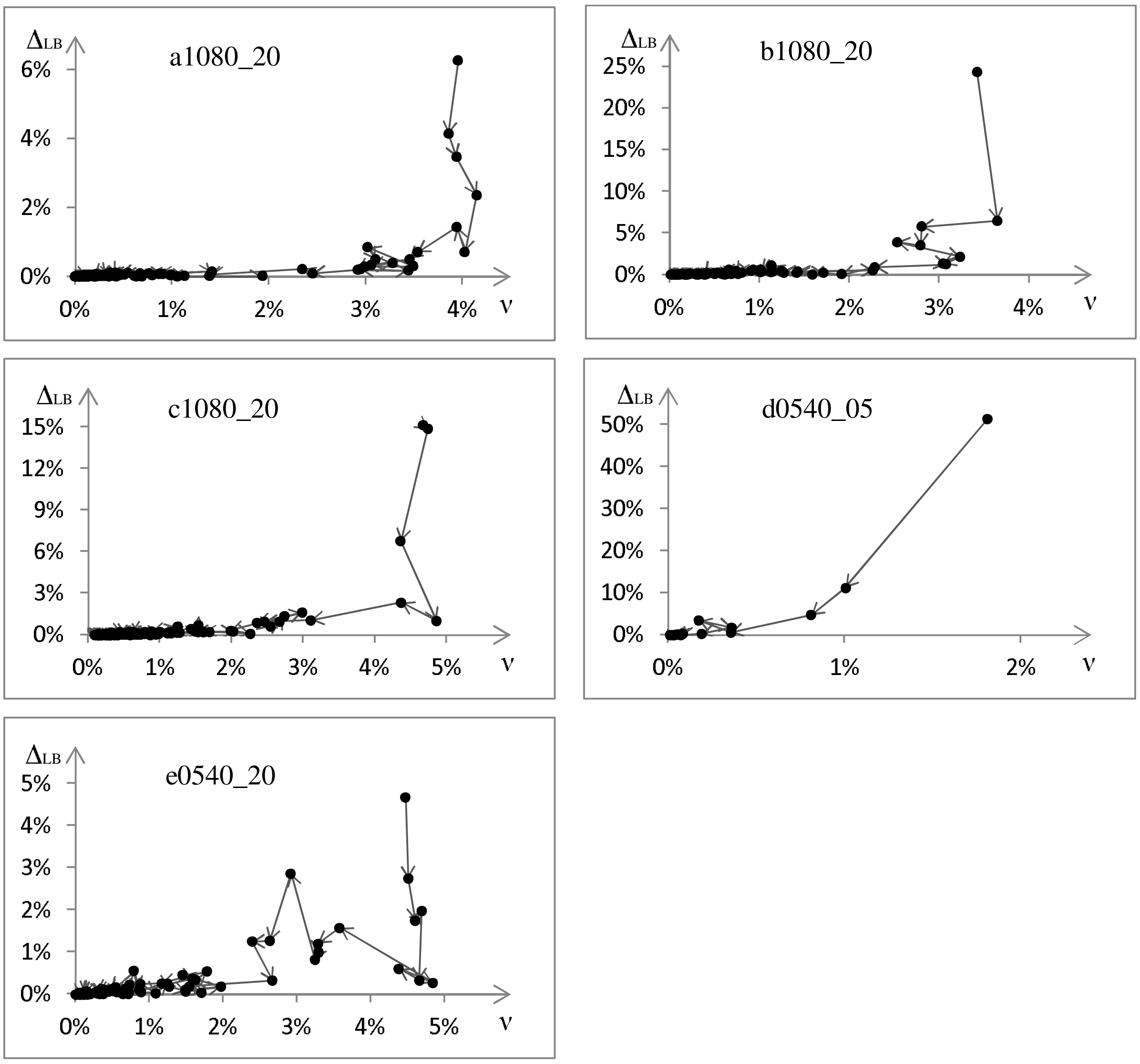


Figure 4.3: Increase in lower bound against α

4.4.7 Overall Algorithmic Framework

In this section we provide the resulting framework of the Lagrangian-based branch-and-cut algorithm under the adaptive cut generation strategy.

Let U denote the global upper bound, and X the current set of global Benders' cuts.

We first present the processing of a non-root active node. The process starts by solving the LP relaxation $\bar{P}(X)$: we obtain an optimal solution (x^*, φ^*), and an optimal solution (λ, μ, ν) to the dual. If this LP lower bound is not less than U, we terminate the node processing. Otherwise, Benders' cut generation follows if x^* is integral or if the condition for adaptive cut addition is satisfied. Whenever a new Benders' cut is found, we add it to X and solve the updated $\bar{P}(X)$ again. This process is repeated until one of the following conditions holds:

(i) the LP lower bound becomes not less than U; or

(ii) no violated constraint is found for (x^*, φ^*); or

(iii) x^* is fractional and the condition for adaptive cut addition is not satisfied.

After the above Benders' cut generation, if x^* is integral, we terminate the node processing because the obtained solution x^* is optimal for the partial problem corresponding to the current node.

The second part of the node processing is based on Lagrangian relaxation. For each agent i, we solve the 0-1 knapsack problem $L_i(X, \mu, \nu)$, obtaining an optimal solution $\breve{x}_i$ through the algorithm of Section 4.4.5. Then, the Lagrangian lower bound $l^*(X, \mu, \nu)$ is computed according to (4.4.84). If it is not less than U, we terminate the node processing. Otherwise, we compute the values of Ξ_{ij} through the two-direction DP tables, and apply the variable fixing rules of Section 4.4.4.

The above node processing procedure is summarized in Algorithm 5.

The overall framework of the Lagrangian-based branch-and-cut algorithm is as follows. We initialize the global upper bound U with the lowest maximum regret between the solution given by the fixed-scenario heuristic under the median-cost scenario and the one given by the dual substitution heuristic. Let A denote the set of active nodes (i.e., partial problems). We initialize A with the root node that corresponds to the original problem. The processing for the root node is the same as that for the other nodes except for the Benders' cut generation, which is applied until condition (ii) above is satisfied. Let l_{Root} denote the Lagrangian lower bound at the root node. Let L denote the global lower bound, and initialize L with l_{Root}. Then we start the branch-and-cut process. We select as the current node an active node η whose corresponding partial problem has the lowest lower bound, i.e., $l_\eta \leq l_\zeta$ for all nodes $\zeta \in A$, and we remove η from A. We apply a branching operation to the current node η, i.e., we generate the children of η by using the rule given in Section 4.4.2. For each child node $\mathcal{N}$, we invoke Algorithm 5. If $\mathcal{N}$ was not terminated by Algorithm 5, we add it to the set A of active nodes. The process is iterated until the

Algorithm 5 Processing at a non-root active node $\mathcal{N}$

```
1: comment $U$ denotes the global upper bound, and $X$ denotes the set of global Benders' cuts.
2: repeat
3:     Solve the LP relaxation $\bar{P}(X)$ with respect to free variables at $\mathcal{N}$, obtaining an optimal solution $(x^*, \varphi^*)$ and an optimal solution $(\lambda, \mu, \nu)$ to the dual.
4:     if the optimal value of $\bar{P}(X) \geq U$ then
5:         terminate node $\mathcal{N}$ and return.
6:     if $x^*$ is integral then
7:         Solve $Q(x^*)$ exactly and obtain an optimal solution $y^*$.
8:     else
9:         if the condition for adaptive cut addition of Section 4.4.6 is not satisfied then goto 19
10:        Solve $Q(x^*)$ by heuristics and obtain a solution $y^*$.
11:    end if
12:    if $q(y^*) < \varphi^*$ then $X \leftarrow X \cup \{y^*\}$.
13: until $q(y^*) \geq \varphi^*$
14: if $x^*$ is integral then
15:     $U \leftarrow$ optimal value of $\bar{P}(X)$.
16:     Terminate node $\mathcal{N}$.
17:     return
18: end if
19: for each agent $i$ do
20:     solve the 0-1 knapsack problem $L_i(X, \mu, \nu)$, obtaining an optimal solution $\breve{x}_i$.
21: end for
22: Compute $l^*(X, \mu, \nu)$ according to (4.4.84).
23: if $l^*(X, \mu, \nu) \geq U$ then terminate node $\mathcal{N}$ and return.
24: for each agent $i$ and each job $j$ do
25:     compute $l_i^*(X, \mu, \nu)_{x_{ij}=1-\breve{x}_{ij}}$ using the algorithm of Section 4.4.5, and then compute $\Xi_{ij}$ through (4.4.85).
26: end for
27: Apply the variable fixing rules of Section 4.4.4 using the values of $\Xi_{ij}$.
28: return
```

set A of active nodes becomes empty or the global lower bound L meets the global upper bound U.

The overall framework of our Lagrangian-based branch-and-cut is provided in Algorithm 6.

Algorithm 6 Lagrangian-Based Branch-and-Cut Approach

```
1: Solve: (i) the classical GAP under the median-cost scenario, and (ii) the
   D-MMR-GAP. Initialize the global upper bound $U$ to the lower maximum regret
   obtained.
2: Initialize the set $A$ of active nodes to the root node.
3: For the root node, apply Benders' cut generation as much as possible, compute the
   Lagrangian lower bound $l_{\text{Root}}$ and apply the variable fixing.
4: Initialize the global lower bound as $L \leftarrow l_{\text{Root}}$.
5: while set $A$ is not empty and $L < U$ do
6:    select as the current node an active node $\eta$ whose corresponding partial problem
      has the lowest lower bound among the nodes of $A$, and remove $\eta$ from $A$.
7:    Generate the children of $\eta$ using the rule of Section 4.4.2.
8:    for each child node $\mathcal{N}$ created from node $\eta$ do
9:       invoke Algorithm 5 to process node $\mathcal{N}$.
10:      if $\mathcal{N}$ was not terminated then add $\mathcal{N}$ to $A$.
11:   end for
12:   $L \leftarrow \min\{l_\zeta \mid \zeta \in A\}$.
13: end while
14: return $U$
```

4.5 Computational Experiments

4.5.1 Instance Generation

To the best of our knowledge, this is the first research on the MMR-GAP. In order to test our approaches, we generated MMR-GAP instances from the following well-known (see [23] and [61]) GAP benchmark instances known as Types A, B, C, D, and E (u.d.

stands for "uniformly distributed"):

A: $\forall i, j$, a_{ij} is a random integer u.d. in $[5, 25]$, c_{ij} is a random integer u.d. in $[10, 50]$; $b_i = 0.6(n/m)15 + 0.4\gamma$, where $\gamma = \max_{i \in I} \sum_{j \in J, \theta_j = i} a_{ij}$ and $\theta_j = \min\{i \mid c_{ij} \le c_{kj}, \forall k \in I\}$.

B: a_{ij} and c_{ij} as for Type A; b_i is set to 70% of the value in Type A.

C: a_{ij} and c_{ij} as for Type A; $b_i = 0.8 \sum_{j=1}^{n} a_{ij}/m$.

D: $\forall i, j$, a_{ij} is a random integer u.d. in $[1, 100]$ and $c_{ij} = 111 - a_{ij} + e_1$ (e_1 a random integer u.d. in $[-10, 10]$); $b_i = 0.8 \sum_{j=1}^{n} a_{ij}/m$.

E: $\forall i, j$, $a_{ij} = 1 - 10 \ln e_2$ (e_2 a random number u.d. in $(0, 1]$), $c_{ij} = 1000/a_{ij} - 10e_3$ (e_3 a random number u.d. in $[0, 1]$); $b_i = 0.8 \sum_{j=1}^{n} a_{ij}/m$.

For each type, we generated GAP instances with $m \in \{5, 10\}$ and $n \in \{40, 80\}$. We then obtained MMR-GAP instances by setting the extremes of the cost intervals, c_{ij}^- and c_{ij}^+, to random integers u.d. in $[(1 - \delta)c_{ij}, c_{ij}]$ and $[c_{ij}, (1 + \delta)c_{ij}]$, respectively, with $\delta \in \{0.10, 0.25, 0.50\}$. We randomly generated 5 instances for each δ, thus obtaining 15 MMR-GAP instances by using each original GAP instance as a seed. As a result, we generated 300 MMR-GAP instances in total, which are available at `http://www.co.cm.is.nagoya-u.ac.jp/~goi/mmr-gap/`.

4.5.2 Implementation Details

The heuristic algorithms proposed for the GAP in [105] and [106] were coded in C, while the algorithms proposed in this chapter were all coded in C++. All codes were built and compiled under Microsoft Visual Studio 2010. We used IBM ILOG CPLEX, version 12.4, for solving linear programming and mixed integer linear programming problems. The CPLEX solver was also used to exactly solve the classical GAP, the Benders' master problems $P(X)$ and $\bar{P}(X)$, and problem D-MMR-GAP for the dual substitution heuristic. All experiments were carried out on a PC with two cores i7-5820K at 3.30 GHz and 32 GB RAM memory under Windows 10 operating system, where the computation was always conducted on a single core.

4.5.3 Heuristic Algorithms

We compared two heuristic algorithms, the fixed-scenario algorithm of Section 4.2.1 and the dual substitution heuristic of Section 4.2.2.

Table 4.1 shows the upper bounds computed by these algorithms for all instances. The GAP instances are denoted as $Xyyzz$, where X = type, $yy = m$ and $zz = n$. For each GAP instance, we give the results for the three values of δ. For every 5 random instances under the same δ, the entries give the average CPU time in seconds spent by the algorithm ("time"), the average optimality gap ("gap"), and the total number of failures ("#f"), i.e., the total number of instances not solved exactly within the time limit. Both the average CPU times and the average optimality gaps were taken for those instances whose optimal solutions were found within the time limit. The optimality gap of solution x is the quantity $(r_{\max}(x) - \mathrm{OPT})/\mathrm{OPT}$, where OPT is the optimal value obtained by the exact algorithms.

We used CPLEX to exactly solve the classical GAP for evaluating the maximum regret of the obtained solutions through Lemma 4.1.1. A time limit of 3600 seconds was assigned to each algorithm. Notation "t.l." indicates that the algorithm was not able to exactly solve even one of the five instances within the time and memory limit. Columns "DS" refer to the dual substitution algorithm. An empty entry indicates that no optimal value was available for calculating the optimality gap.

Among the fixed-scenario approaches, the median-cost scenario $(c^+ + c^-)/2$ obtains the best solutions for most of the instances. The dual substitution algorithm provides better solutions with objective values very close to the optimal values. However, for Types D and E, the computation of the dual substitution algorithm becomes very expensive.

Table 4.1: Results of heuristic algorithms

		Fixed Scenario											
		c^-			$(c^+ + c^-)/2$			c^+			DS		
instance	δ	time	#f	gap	time	#f	gap	time	#f	gap	time	#f	gap
A0540	0.10	0.03	0	4.3%	0.08	0	1.5%	0.01	0	7.6%	0.06	0	0.0%
	0.25	0.02	0	15.2%	0.22	0	1.4%	0.01	0	3.6%	0.50	0	0.7%
	0.50	0.02	0		0.05	0		0.01	0		0.60	0	
A0580	0.10	0.03	0	16.3%	0.02	0	1.7%	0.03	0	8.0%	0.19	0	1.0%
	0.25	0.01	0		0.01	0		0.01	0		0.67	0	
	0.50	0.02	0		0.01	0		0.03	0		6.45	0	
A1040	0.10	0.02	0	17.5%	0.02	0	4.9%	0.02	0	19.4%	0.21	0	0.0%
	0.25	0.03	0	9.6%	0.03	0	1.7%	0.04	0	9.9%	0.42	0	0.5%
	0.50	0.05	0		0.03	0		0.01	0		1.17	0	
A1080	0.10	0.04	0	15.6%	0.03	0	1.1%	0.02	0	6.7%	0.68	0	0.0%
	0.25	0.02	0		0.03	0		0.01	0		1.43	0	
	0.50	0.05	0		0.02	0		0.03	0		15.83	0	
B0540	0.10	0.31	0	4.6%	0.15	0	9.7%	0.22	0	12.7%	0.42	0	8.0%
	0.25	0.18	0	11.6%	0.34	0	3.1%	0.16	0	9.9%	1.05	0	0.6%
	0.50	0.37	0	16.9%	0.20	0	1.2%	0.16	0	3.8%	1.89	0	0.0%
B0580	0.10	0.55	0	15.2%	0.41	0	2.1%	0.41	0	1.1%	2.70	0	0.0%
	0.25	0.57	0		0.90	0		0.28	0		19.35	0	
	0.50	0.57	0		0.35	0		0.73	0		80.03	0	

Table 4.1: continued from previous page

		Fixed Scenario											
		c^-			$(c^+ + c^-)/2$			c^+			DS		
instance	δ	time	#f	gap	time	#f	gap	time	#f	gap	time	#f	gap
B1040	0.10	0.34	0	11.6%	0.16	0	0.9%	0.12	0	2.5%	0.52	0	6.7%
	0.25	0.22	0	5.6%	0.24	0	3.0%	0.17	0	11.3%	4.09	0	3.5%
	0.50	0.23	0	24.4%	0.19	0	4.5%	0.14	0	8.7%	14.07	0	0.0%
B1080	0.10	0.90	0	18.0%	0.98	0	5.8%	0.58	0	13.2%	6.31	0	2.2%
	0.25	0.63	0		0.56	0		0.45	0		168.19	0	
	0.50	1.16	0		0.78	0		0.87	0		2052.81	1	
C0540	0.10	0.40	0	4.0%	0.26	0	0.0%	0.27	0	6.7%	0.55	0	0.0%
	0.25	0.48	0	11.0%	0.22	0	1.7%	0.38	0	6.4%	1.26	0	0.0%
	0.50	0.38	0	12.8%	0.45	0	3.2%	0.24	0	4.2%	5.82	0	0.0%
C0580	0.10	0.42	0	11.2%	0.69	0	0.0%	0.58	0	7.1%	3.11	0	0.0%
	0.25	0.74	0		0.70	0		0.38	0		13.24	0	
	0.50	0.51	0		0.44	0		0.53	0		677.74	0	
C1040	0.10	0.44	0	8.3%	0.47	0	7.1%	0.28	0	7.4%	1.08	0	3.8%
	0.25	0.40	0	16.6%	0.31	0	5.7%	0.24	0	7.0%	1.68	0	2.1%
	0.50	0.85	0	21.4%	0.28	0	0.0%	0.40	0	0.0%	14.00	0	0.0%
C1080	0.10	2.09	0	20.0%	0.88	0	3.5%	0.63	0	7.5%	12.38	0	3.9%
	0.25	0.93	0		1.48	0		1.00	0		75.95	0	
	0.50	1.66	0		1.24	0		0.70	0		1033.34	1	
D0540	0.10	10.83	0		11.89	0		21.71	0		121.95	0	
	0.25	7.77	0		4.89	0		7.53	0		931.42	0	
	0.50	2.46	0		6.74	0		9.30	0		574.27	0	
D0580	0.10	169.32	0		166.96	0		89.29	0		t.l.	5	
	0.25	98.41	0		178.03	0		214.99	0		t.l.	5	
	0.50	26.71	0		136.86	0		60.81	0		t.l.	5	
D1040	0.10	26.89	0		37.98	0		37.59	0		518.40	0	
	0.25	6.94	0		58.14	0		19.59	0		1301.18	0	
	0.50	3.14	0		16.02	0		4.09	0		387.35	0	
D1080	0.10	1652.03	4		739.86	3		t.l.	5		t.l.	5	
	0.25	1422.67	0		t.l.	5		1927.13	4		t.l.	5	
	0.50	260.87	0		849.46	3		1668.17	3		t.l.	5	
E0540	0.10	0.64	0	9.5%	0.51	0	2.5%	0.48	0	5.5%	3.05	0	0.4%
	0.25	0.26	0	10.8%	0.68	0	2.5%	0.61	0	4.6%	16.31	0	0.0%
	0.50	0.63	0		0.68	0		0.44	0		47.27	0	
E0580	0.10	1.06	0	9.9%	1.73	0	2.6%	0.99	0	4.3%	94.13	0	0.0%
	0.25	1.84	0		1.02	0		1.18	0		246.46	0	
	0.50	0.95	0		1.59	0		1.09	0		502.27	1	
E1040	0.10	4.03	0	10.6%	2.98	0	5.6%	3.31	0	11.6%	64.45	0	9.1%
	0.25	2.46	0		2.25	0		2.62	0		436.48	0	
	0.50	2.65	0		2.44	0		1.65	0		1185.87	0	
E1080	0.10	8.72	0		7.11	0		5.43	0		1477.02	4	
	0.25	8.76	0		5.14	0		7.70	0			5	

Table 4.1: continued from previous page

		Fixed Scenario											
		c^-			$(c^+ + c^-)/2$			c^+			DS		
instance	δ	time	#f	gap	time	#f	gap	time	#f	gap	time	#f	gap
	0.50	3.69	0		9.87	0		8.57	0			5	

The average gaps from the known optimal values for Types A, B, C, and E are reported in Table 4.2. The dual substitution heuristic obtained better upper bounds than the fixed-scenario heuristic for the instances of all types.

Table 4.2: Average optimality gap of heuristic algorithms

	Fixed-Scenario			
Type	c^-	$(c^+ + c^-)/2$	c^+	DS
A	13.0%	2.1%	9.3%	0.4%
B	12.5%	3.7%	7.7%	2.9%
C	12.5%	2.9%	6.3%	1.4%
E	10.2%	3.2%	6.4%	2.1%

4.5.4 Exact Algorithms

We tested the exact algorithms of Section 4.3 on all instances. Recall that, for the logic-based Benders decomposition, when it is halted with a time limit, the objective value of the obtained solution is a valid lower bound, although it is usually an infeasible solution (see Section 4.3.1). On the other hand, the solution value obtained by the branch-and-cut framework provides a valid upper bound on the optimal regret when it terminates with a time limit. Each pair of entries in Table 4.3 shows the average CPU time in seconds spent by the algorithm (computed for those instances whose optimal solutions were found within the time limit), the average optimality gap from the optimal value or from a lower bound (taken for those instances whose optimal solutions or valid lower bounds were found within the time limit), and the number of instances where an optimal solution was found (in column "#opt") within the time limit. We calculated the optimality gap of solution x in the same way as described in Section 4.5.3, where for those instances whose optimal values are not known, the valid lower bounds obtained by logic-based Benders decomposition are used instead. An instance was not included into the average calculation if no feasible solution was obtained by the corresponding algorithm. We use B&C and C&B to denote branch-and-cut and cut-and-branch, respectively. "Benders" is the logic-based Benders decomposition approach of Section 4.3.1, while "basic B&C" is the basic branch-and-cut algorithm discussed in Section 4.3.2. Columns “Lagrangian-based B&C” refer to the approach proposed in Section 4.4: “full-cut,” “C&B” and “adaptive” represent the three cut management approaches discussed in Section 4.4.6. A time limit of 3600 seconds was assigned to each algorithm. For the Lagrangian-based branch-and-cut algorithms, the time limit for solving the D-MMR-GAP to obtain an initial upper bound was set to 300 seconds. Notation “t.l.,” has the same meaning as in Table 4.1.

The adaptive cut generation introduced in Section 4.4.6 is the most efficient among Lagrangian-based branch-and-cut algorithms.

With respect to logic-based Benders decomposition, the Lagrangian-based branch-and-cut algorithm with adaptive cut generation has on average shorter CPU times for Types A and E, while for Types B and C there is no clear winner. For Type D (the most difficult one), we omit the results since none of the algorithms was able to obtain optimal solutions, and there is a large gap between the lower bound obtained by the logic-based Benders decomposition approach and the upper bound obtained by the branch-and-cut algorithms.

Table 4.3: Results of exact algorithms

							Lagrangian-based B&C								
		Benders		basic B&C			full-cut			C&B			adaptive		
instance	δ	time	#opt	time	#opt	gap	time	#opt	gap	time	#opt	gap	time	#opt	gap
A0540	0.10	1.2	5	1.1	5	0.0%	2.3	5	0.0%	0.8	5	0.0%	0.3	5	0.0%
	0.25	484.0	5	59.4	5	0.0%	420.1	4	0.5%	86.7	5	0.0%	50.8	5	0.0%
	0.50	t.l.	0	t.l.	0	11.3%	t.l.	0	9.3%	t.l.	0	9.3%	t.l.	0	9.3%
A0580	0.10	205.5	5	10.8	5	0.0%	114.6	5	0.0%	35.9	5	0.0%	16.7	5	0.0%
	0.25	t.l.	0	t.l.	0	20.1%	t.l.	0	11.6%	t.l.	0	11.6%	t.l.	0	11.6%
	0.50	t.l.	0	t.l.	0	58.4%	t.l.	0	35.8%	t.l.	0	35.8%	t.l.	0	35.8%
A1040	0.10	2.8	5	3.3	5	0.0%	12.3	5	0.0%	5.0	5	0.0%	1.0	5	0.0%
	0.25	1668.2	3	1192.3	4	0.3%	196.8	2	0.5%	1122.5	4	0.0%	926.6	5	0.0%
	0.50	t.l.	0	t.l.	0	18.8%	t.l.	0	11.9%	t.l.	0	11.9%	t.l.	0	11.9%
A1080	0.10	1523.2	3	1379.7	4	1.7%	184.7	3	1.7%	1531.7	3	1.7%	1470.3	3	1.7%
	0.25	t.l.	0	t.l.	0	31.3%	t.l.	0	17.0%	t.l.	0	17.0%	t.l.	0	17.0%
	0.50	t.l.	0	t.l.	0	71.2%	t.l.	0	41.5%	t.l.	0	41.5%	t.l.	0	41.5%
B0540	0.10	1.8	5	29.6	5	0.0%	213.8	5	0.0%	5.5	5	0.0%	6.5	5	0.0%
	0.25	49.0	5	167.1	5	0.0%	1580.9	4	0.0%	47.5	5	0.0%	14.0	5	0.0%
	0.50	2169.1	4	2152.7	3	4.3%	757.2	1	0.0%	1598.0	4	0.0%	804.8	5	0.0%
B0580	0.10	131.5	5	946.7	4	0.0%	1286.1	2	0.0%	212.0	5	0.0%	80.6	5	0.0%
	0.25	t.l.	0	t.l.	0	40.9%	t.l.	0	14.8%	t.l.	0	14.8%	t.l.	0	14.8%
	0.50	t.l.	0	t.l.	0	81.0%	t.l.	0	40.5%	t.l.	0	40.5%	t.l.	0	40.5%
B1040	0.10	7.0	5	428.2	5	0.0%	228.7	4	0.0%	15.4	5	0.0%	9.7	5	0.0%

Table 4.3: continued from previous page

							Lagrangian-based B&C								
		Benders		basic B&C			full-cut			C&B			adaptive		
instance	δ	time	#opt	time	#opt	gap	time	#opt	gap	time	#opt	gap	time	#opt	gap
	0.25	825.5	4	1653.4	3	4.0%	t.l.	0	0.9%	868.6	4	0.6%	822.1	4	0.6%
	0.50	3562.3	1	t.l.	0	14.4%	t.l.	0	4.0%	t.l.	0	4.0%	t.l.	0	4.0%
B1080	0.10	464.0	5	2688.6	2	242.4%	696.5	1	1.8%	1706.4	3	1.0%	1597.2	3	1.0%
	0.25	t.l.	0	t.l.	0	93.3%	t.l.	0	19.8%	t.l.	0	19.8%	t.l.	0	19.8%
	0.50	t.l.	0	t.l.	0		0.0	0	45.6%	t.l.	0	45.6%	t.l.	0	45.6%
C0540	0.10	2.6	5	20.0	5	0.0%	153.9	5	0.0%	6.4	5	0.0%	5.4	5	0.0%
	0.25	63.3	5	217.8	5	0.0%	1010.4	3	0.0%	64.9	5	0.0%	22.6	5	0.0%
	0.50	2538.4	2	2412.0	2	4.2%	t.l.	0	2.0%	2173.8	3	2.0%	1913.2	4	2.0%
C0580	0.10	95.6	5	429.7	5	0.0%	928.2	1	0.0%	205.2	5	0.0%	75.6	5	0.0%
	0.25	t.l.	0	t.l.	0	33.5%	t.l.	0	12.7%	t.l.	0	12.7%	t.l.	0	12.7%
	0.50	t.l.	0	t.l.	0	101.5%	t.l.	0	45.0%	t.l.	0	45.0%	t.l.	0	45.0%
C1040	0.10	9.6	5	1072.7	5	0.0%	1111.2	5	0.0%	13.8	5	0.0%	14.7	5	0.0%
	0.25	87.0	5	2992.3	1	11.6%	t.l.	0	1.4%	174.8	5	0.0%	136.7	5	0.0%
	0.50	2883.7	2	t.l.	0	44.1%	t.l.	0	4.5%	t.l.	0	4.5%	t.l.	0	4.5%
C1080	0.10	711.6	5	t.l.	0	451.2%	t.l.	0	2.1%	1721.2	3	1.5%	1664.5	3	1.5%
	0.25	t.l.	0	t.l.	0	203.5%	t.l.	0	16.7%	t.l.	0	16.7%	t.l.	0	16.7%
	0.50	t.l.	0	t.l.	0		t.l.	0	44.4%	t.l.	0	44.4%	t.l.	0	44.4%
E0540	0.10	54.9	5	386.9	5	0.0%	1712.0	3	0.4%	49.5	5	0.0%	19.5	5	0.0%
	0.25	t.l.	0	t.l.	0	10.9%	t.l.	0	0.0%	2507.1	5	0.0%	2706.2	3	0.0%
	0.50	t.l.	0	t.l.	0	38.7%	t.l.	0	22.6%	t.l.	0	22.6%	t.l.	0	22.6%
E0580	0.10	t.l.	0	t.l.	0	74.2%	t.l.	0	22.6%	3282.0	1	22.6%	3121.6	1	22.6%

Table 4.3: continued from previous page

							Lagrangian-based B&C								
		Benders		basic B&C			full-cut			C&B			adaptive		
instance	δ	time	#opt	time	#opt	gap	time	#opt	gap	time	#opt	gap	time	#opt	gap
	0.25	t.l.	0	t.l.	0	115.4%	t.l.	0	53.7%	t.l.	0	53.7%	t.l.	0	53.7%
	0.50	t.l.	0	t.l.	0	126.6%	t.l.	0	63.5%	t.l.	0	63.6%	t.l.	0	63.6%
E1040	0.10	2110.7	3	t.l.	0	125.2%	t.l.	0	5.7%	2248.4	2	4.6%	3399.4	1	5.0%
	0.25	t.l.	0	t.l.	0	102.7%	t.l.	0	18.6%	t.l.	0	18.6%	t.l.	0	18.6%
	0.50	t.l.	0	t.l.	0	84.0%	t.l.	0	27.0%	t.l.	0	27.0%	t.l.	0	27.0%
E1080	0.10	t.l.	0	t.l.	0		t.l.	0	47.0%	t.l.	0	47.3%	t.l.	0	47.0%
	0.25	t.l.	0	t.l.	0		t.l.	0	61.5%	t.l.	0	61.7%	t.l.	0	61.9%
	0.50	t.l.	0	t.l.	0		t.l.	0	65.5%	t.l.	0	65.5%	t.l.	0	66.1%

Table 4.4 provides the overall results for the introduced exact algorithms. For each algorithm and instance type, the entries provide the number of instances for which the algorithm determined the optimal solution with the least computation time among the three methods. The value in parentheses gives the number of instances solved to optimality within one hour. The Lagrangian-based branch-and-cut algorithm with adaptive cut generation had the best results for the instances of Types A and E. In total, it solved to optimality 102 out of 240 instances of Types A, B, C, and E, in 64 cases with the lowest CPU time.

Table 4.4: Results of exact algorithms

Type	Benders	basic B&C	Lagrangian-based B&C
A	0 (26)	7 (28)	22 (28)
B	15 (34)	0 (27)	20 (32)
C	21 (34)	0 (23)	13 (32)
E	3 (8)	0(5)	9 (10)

4.6 Conclusion of the MMR-GAP

In this chapter, we studied the min-max regret GAP, a robust version of the GAP. We showed that the decision version of this problem is Σ_2^p-complete. Problems of this class are much more difficult than NP-hard problems; indeed it is considered very unlikely that they even belong to NP.

We presented and compared heuristic and exact methods. We computationally examined a fixed-scenario heuristic, for which three scenarios were considered. We also presented a dual substitution heuristic that uses a mixed integer programming formulation obtained by replacing a subproblem with its dual. We observed that this heuristic method in most cases provides better upper bounds than the fixed-scenario heuristic.

We also examined three exact methods: a logic-based Benders decomposition approach, a basic branch-and-cut algorithm, and a Lagrangian-based branch-and-cut algorithm that incorporates several ideas. We used a Lagrangian lower bound that is stronger than the LP lower bound and is obtained by solving m 0-1 knapsack problems through dynamic programming. This computation was further exploited for variable fixing. Compared with logic-based Benders decomposition, the Lagrangian-based branch-and-cut had better performance for instances of Types A and E, and it exactly solved 102 out of 240 instances of Types A, B, C, and E with m up to 10 and n up to 80.

Chapter 5

Min-Max Regret Multidimensional Knapsack Problem

In this chapter, we continue to study another combinatorial optimization problem with uncertainty. We consider the *multidimensional knapsack problem* (MKP) with min-max regret criterion under interval profits. The classical MKP is a strongly NP-hard combinatorial optimization problem [41] and has been widely studied over many decades due to both theoretical interests and its broad applications in several engineering fields, such as cargo loading, cutting stock, bin-packing, financial and other management issues [99]. The *interval min-max regret multidimensional knapsack problem* (MMR-MKP) is a generalization of the MKP under the interval profit coefficients, in which all profit coefficients can take any value from a corresponding given interval independently. The MMR-MKP aims to find a robust solution that minimizes the maximum regret.

We examine several heuristic and exact algorithms, such as the fixed-scenario algorithm, the logic-based approach and the basic branch-and-cut for the MMR-MKP, which we introduced in Chapter 4. We further propose an iterative heuristic based on a mixed integer programming (MIP) model obtained by replacing a subproblem with the dual of its continuous relaxation, which we call the *iterated dual substitution* (IDS) algorithm.

We evaluate the algorithms through computational experiments on a wide set benchmark instances. The proposed IDS algorithm performs best for all of the tested instances.

5.1 Problem Description

The *multidimensional knapsack problem* (MKP) is defined as follows. Given a set of n items $J = \{1, \ldots, n\}$ and a set of m types of resource $I = \{1, \ldots, m\}$, we want to select a subset of items with maximum total profit, subject to satisfying all resource constraints.

Selecting item j with profit p_j consumes an amount a_{ij} of the resource in dimension i, whereas the total amount of the resource available at dimension i (capacity) is b_i. The MKP can be formulated over binary variables x_j indicating that item j is chosen if and only if $x_j = 1$:

$$\max \sum_{j=1}^{n} p_j x_j \tag{5.1.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \qquad \forall i \in I \tag{5.1.2}$$

$$x_j \in \{0,1\}, \qquad \forall j \in J. \tag{5.1.3}$$

For convenience, we define X_0 to be the set of all feasible solutions of MKP: $X_0 = \{x \mid x \text{ satisfies constraints (5.1.2)–(5.1.3)}\}$.

In many real-world applications the profit p_j is affected by a number of factors and may not be an accurate value. In this chapter we assume that every profit p_j can take any value within a given range $[p_j^-, p_j^+]$. A scenario s is defined as an array of profits p_j^s satisfying $p_j^s \in [p_j^-, p_j^+], \forall j \in J$. We denote by $z^s(x)$ the objective function value of solution x under scenario s:

$$z^s(x) = \sum_{j=1}^{n} p_j^s x_j. \tag{5.1.4}$$

Let z_*^s be the optimal solution value under scenario s, i.e.,

$$z_*^s = \max_{y \in X_0} z^s(y). \tag{5.1.5}$$

The *regret* $r^s(x)$ corresponding to solution x under scenario s is then the difference between these two values:

$$r^s(x) = z_*^s - z^s(x). \tag{5.1.6}$$

We define S to be the set of all possible scenarios, i.e., $S = \{s \mid p_j^s \in [p_j^-, p_j^+]\}$. The *maximum regret* of a solution x is then the maximum $r^s(x)$ value over all scenarios:

$$r_{\max}(x) = \max_{s \in S} r^s(x). \tag{5.1.7}$$

The MMR-MKP requires to find a feasible solution x such that the maximum regret is minimized:

$$\begin{aligned} \min_{x \in X_0} r_{\max}(x) &= \min_{x \in X_0} \max_{s \in S} r^s(x) \\ &= \min_{x \in X_0} \max_{\substack{y \in X_0 \\ s \in S}} \left\{ \sum_{j=1}^{n} p_j^s y_j - \sum_{j=1}^{n} p_j^s x_j \right\}. \end{aligned} \tag{5.1.8}$$

The following lemma is a classical general result that was proposed by Aissi et al. [2] (whose roots are in Yaman et al. [108]).

Lemma 5.1.1. *The regret of a solution $x \in X_0$ is maximized under the following scenario $\sigma(x)$:*

$$p_j^{\sigma(x)} = \begin{cases} p_j^- & \text{if } x_j = 1 \\ p_j^+ & \text{otherwise} \end{cases} \qquad \forall j \in J. \tag{5.1.9}$$

This worst case lemma implies that the value $r_{\max}(x)$ is achieved by the scenario that gives the worst profits to the selected items, and the best profits to the non-selected items.

From Lemma 5.1.1, the MMR-MKP (5.1.8) can be rewritten as

$$\begin{aligned} &\min_{x\in X_0} r_{\max}(x) = \\ &\min_{x\in X_0} \left\{ \max_{y\in X_0} \sum_{j=1}^{n} (p_j^+ + (p_j^- - p_j^+)x_j)y_j - \sum_{j=1}^{n} p_j^- x_j \right\}. \end{aligned} \tag{5.1.10}$$

5.2 Standard Techniques

5.2.1 Fixed-Scenario Algorithm

We described the same idea in Section 4.2.1, and we now applied it to MMR-MKP. This fixed-scenario algorithm is based on the observation that the feasible region of the MMR-MKP is the same as that of the classical MKP. This implies that we can obtain a feasible solution to an MMR-MKP instance by fixing a scenario, solving the resulting MKP instance to optimality, and evaluating the maximum regret of the obtained solution using (5.1.10).

Among the three types of commonly used scenarios, namely, the lowest profit $p_j^s = p_j^-$, the highest profit $p_j^s = p_j^+$, and the median profit $p_j^s = (p_j^- + p_j^+)/2$, in this chapter we consider the median-profit scenario for which the following general result showed in Lemma 4.2.1 can also be directly applied to the case of MMR-MKP.

Lemma 5.2.1. *Let $\widetilde{s}$ be the median-profit scenario, i.e., $p_j^{\widetilde{s}} = (p_j^- + p_j^+)/2$, $\forall j \in J$, and let $\widetilde{x}$ be an optimal solution to the MKP under $\widetilde{s}$. Then, $r_{\max}(\widetilde{x}) \leq 2r_{\max}(x)$ holds for all $x \in X_0$.*

We next provide a tight example for the approximation ratio of Lemma 5.2.1.

Lemma 5.2.2. *There is an instance of the MMR-MKP for which an optimal solution to the MKP under the median-profit scenario $\widetilde{s}$ has a regret twice as large as the optimal regret.*

Proof. Let $m = 1$, $n = 3$, $a_{11} = a_{12} = a_{13} = 1$, $b_1 = 1$ with interval profits $p_1 \in [1, 1]$, $p_2 \in [0, 2]$, $p_3 \in [0, 2]$. Choosing any item gives an optimal solution to MKP under the median-profit scenario, because all median profits have value 1. The optimal MMR-MKP solution chooses item 1, attaining the maximum regret of value 1, while choosing item 2 (or 3) gives the maximum regret of value 2. □

5.2.2 Logic-Based Benders Decomposition

We examine the logic-based Benders decomposition to MMR-MKP (see Section 2.3 for details and 4.3.1). This technique is able to solve the MMR-MKP to optimality if sufficient time and memory are available; otherwise, it will provide a lower bound on the optimal solution value. We use this lower bound for calculating the optimality gap for our heuristic algorithms.

This approach is based on a MIP model of the MMR-MKP. By introducing a new continuous variable λ, along with constraints that force λ to satisfy $\lambda \geq z_*^s$, $\forall s \in S$, the MMR-MKP can be formulated as the following MIP model (MIP-MMR-MKP):

$$\min \lambda - \sum_{j=1}^{n} p_j^- x_j \tag{5.2.11}$$

$$\text{s.t.} \quad \lambda \geq \sum_{j=1}^{n} (p_j^+ + (p_j^- - p_j^+)x_j)y_j, \qquad \forall y \in X_0 \tag{5.2.12}$$

$$x \in X_0. \tag{5.2.13}$$

This model is still hard due to the exponential number of constraints (5.2.12). We define a *master problem* $P(X)$ as the relaxation of the MIP-MMR-MKP in which set X_0 in constraints (5.2.12) is replaced by a subset $X \subseteq X_0$:

$$\lambda \geq \sum_{j=1}^{n} (p_j^+ + (p_j^- - p_j^+)x_j)y_j, \qquad \forall y \in X. \tag{5.2.14}$$

We call the constraints (5.2.14) Benders' cuts. For an optimal solution (x^*, λ^*) to the current master problem $P(X)$, we define a *slave problem* $Q(x^*)$ as

$$\max_{y \in X_0} \sum_{j=1}^{n} (p_j^+ + (p_j^- - p_j^+)x_j^*)y_j. \tag{5.2.15}$$

Let y^* be an optimal solution to $Q(x^*)$ and $q(y^*)$ be the corresponding objective function value. If $q(y^*) > \lambda^*$ holds, then the constraint (5.2.12) induced by y^* is violated by the current optimal solution (x^*, λ^*) of $P(X)$. Whenever such a violated constraint is found, we add the solution y^* to X and solve the updated $P(X)$. The process is iterated until the algorithm finds a solution (x^*, λ^*) for which no violated constraints exist.

When the algorithm terminates with this termination criterion, the obtained solution does not violate any constraint (5.2.12) and is optimal to the original MMR-MKP. Note that $P(X)$ is a relaxation of the MIP-MMR-MKP, and its optimal solution value at each iteration is a lower bound on the optimal value of the original MMR-MKP.

The iteration starts with set $X = \{\widetilde{x}\}$, where $\widetilde{x}$ is the optimal solution obtained by the fixed-scenario heuristic under the median-profit scenario.

5.2.3 A Branch-and-Cut Algorithm

Branch-and-cut is another standard exact approach widely applied to interval min-max regret problems (see [40, 73] and [84]). We implement this algorithm using Benders' cuts based on a basic branch-and-cut framework. We define $\bar{P}(X)$ as the continuous relaxation of $P(X)$. We solve $\bar{P}(X)$ (with respect to the free variables) at each node of the search tree. Its optimal value is a lower bound for the corresponding partial problem. If this lower bound is not smaller than the incumbent solution value, then we prune the current node. When an integer-feasible solution candidate x^* has been identified, we check a violated constraint (5.2.12) by solving the slave problem $Q(x^*)$. If such a Benders' cut exists, we add an optimal solution y^* for $Q(x^*)$ to the current set X, and solve the updated $\bar{P}(X)$. Then, if the updated lower bound exceeds the incumbent solution value, we prune the node. We then check the integrality of the newly obtained x^*. If it becomes fractional, a branching follows; otherwise, we repeat the above process to check a new violated constraint (5.2.12) until no such constraint exists. When it occurs, we update the incumbent solution and prune the node.

The general framework of branch-and-cut can be implemented in many ways. We maintain the set X for constraints (5.2.14) as follows. To prevent the LP relaxation $\bar{P}(X)$ at the root node from being unbounded, we start with set $X = \{\widetilde{x}\}$, where $\widetilde{x}$ is the optimal solution obtained by the fixed-scenario heuristic under the median-profit scenario. All cuts added to X at any active node are used throughout the remaining computation, i.e., they contribute to all other active nodes.

The branch-and-cut algorithm usually obtains feasible solutions during the search process. Hence, the branch-and-cut algorithm can also be used as a heuristic by prematurely terminating it.

5.3 Iterated Dual Substitution Heuristic

5.3.1 Dual Substitution Heuristic

We introduce the dual substitution (DS) heuristic, an algorithm based on a MIP model in which a subproblem is replaced by the dual counterpart of its linear relaxation.

By using (5.1.9), the maximization problem over y in (5.1.10),

$$\max_{y \in X_0} \sum_{j=1}^{n} (p_j^+ + (p_j^- - p_j^+)x_j)y_j,$$

for every fixed x is an MKP that can be expressed as

$$\max_{y \in X_0} \sum_{j=1}^{n} p_j^{\sigma(x)} y_j.$$

We consider the corresponding linear program obtained by replacing the integrality constraint $y_j \in \{0,1\}$ with a weaker requirement: $0 \le y_j \le 1$.

We define two types of dual variables, u_i $(i \in I)$ for constraints (5.1.2) and v_j $(j \in J)$ for constraints $y_j \le 1$. By embedding the dual counterpart into (5.1.10), we obtain the following dual substitution model (D-MMR-MKP):

$$\begin{aligned}
\min \quad & \sum_{i=1}^{m} b_i u_i + \sum_{j=1}^{n} v_j - \sum_{j=1}^{n} p_j^- x_j && && (5.3.16)\\
\text{s.t.} \quad & \sum_{i=1}^{m} a_{ij} u_i + v_j \ge p_j^+ + (p_j^- - p_j^+)x_j, && \forall j \in J && (5.3.17)\\
& u_i \ge 0, && \forall i \in I && (5.3.18)\\
& v_j \ge 0, && \forall j \in J && (5.3.19)\\
& x \in X_0. && && (5.3.20)
\end{aligned}$$

The dual substitution heuristic exactly solves the D-MMR-MKP and outputs the obtained solution. The D-MMR-MKP is not easier than the MKP, since it contains all the MKP constraints.

Property 5.3.1. *The optimal solution value of the D-MMR-MKP is an upper bound on the optimal value the MMR-MKP.*

In addition, for any instance, a tighter upper bound can be obtained as follows.

Property 5.3.2. *The bound obtained by evaluating the maximum regret of any optimal solution of the D-MMR-MKP is at least as good as the optimal value of the D-MMR-MKP.*

We observed through computational experiments that the dual substitution heuristic obtains better solutions compared to the fixed-scenario heuristic for most instances. However, we proved that this algorithm cannot have a guarantee on its solution quality.

Lemma 5.3.1. *For any positive value $\epsilon < 1$, there exists an instance such that the solution value of the dual substitution heuristic is at least $1/\epsilon$ times the optimal value.*

Proof. Let $m = 1$, $n = 2$, $a_{11} = 1$, $a_{12} = K_1$, $b_1 = K_1$ with interval profits $p_1 \in [1, 2 + \epsilon]$, $p_2 \in [2, 2]$, where $0 < \epsilon < 1$ and $K_1 \gg 1$. Obviously, we can select at most one item for any feasible solution due to the capacity constraint (5.1.2). The solution selecting item 2 has the maximum regret ϵ and is optimal to this instance, while selecting item 1 leads to the maximum regret 1. However, under the formulation (5.3.16)–(5.3.20), the solution selecting item 1 with the objective value $2(K_1 - 1)/K_1$, is better than the solution selecting item 2 with the objective value $2(K_1 - 1)/K_1 + \epsilon$. Therefore, the ratio of the obtained objective value to the optimal value is $1/\epsilon$, which can be arbitrarily large. $\square$

5.3.2 Iterated Dual Substitution

The dual substitution heuristic performs well compared to the fixed scenario algorithm. However, it can fail in obtaining good solutions as stated in Lemma 5.3.1. Below we propose an iterative method to improve the performance of the dual substitution heuristic by iteratively applying it, excluding from the search space the solutions already checked in the previous iterations.

Recall that any feasible solution $\hat{x}$ to the D-MMR-MKP is also feasible to the MMR-MKP. For a set $\hat{X}$ $(\subseteq X_0)$ of feasible solutions, we consider the following constraint:

$$\sum_{j:\hat{x}_j=0} x_j + \sum_{j:\hat{x}_j=1} (1 - x_j) \geq d + 1, \qquad \forall \hat{x} \in \hat{X}, \tag{5.3.21}$$

where d (≥ 0) is a parameter. Any solution $x \in X_0$ that satisfies (5.3.21) for a solution $\hat{x}$ has a Hamming distance larger than d from $\hat{x}$. We denote by model D-MMR-MKP$(\hat{X})$ the problem obtained by adding constraints (5.3.21) to (5.3.16)–(5.3.20). This is the problem of finding a best solution to D-MMR-MKP under the constraint that the solution must have a distance greater than d to every solution in $\hat{X}$. That is, solutions within distance d from a solution in $\hat{X}$ are removed from the feasible region.

The *iterated dual substitution* (IDS) algorithm starts with an empty set $\hat{X} = \emptyset$. It solves the D-MMR-MKP$(\hat{X})$, obtaining a solution $\hat{x}$, and evaluates its maximum regret $r_{\max}(\hat{x})$. It then adds $\hat{x}$ to $\hat{X}$, and solves the updated D-MMR-MKP$(\hat{X})$. This process is repeated until a termination condition (time limit in this chapter) is satisfied.

Note that since the IDS algorithm does not use any problem specific knowledge of the MKP, it is a general framework that can also be applicable to other min-max regret problems with interval profits (costs).

5.3.3 Local Exact Subroutine

Note that if we set d to 0, the IDS approach at every iteration only excludes from the search space, those $\hat{x}$ that have been checked during the search by then. On the other hand, when $d \geq 1$, the IDS approach also removes unchecked solutions around $\hat{x}$ from the search space. Such an unchecked space around a solution $\hat{x}$ can be defined by

$$1 \leq \sum_{j:\hat{x}_j=0} x_j + \sum_{j:\hat{x}_j=1} (1 - x_j) \leq d. \tag{5.3.22}$$

When $d \geq 1$, we consider an option that exactly solves the MMR-MKP with an additional constraint of (5.3.22), which we call the MMR-MKP($\hat{x}$), whenever a solution $\hat{x}$ is added to $\hat{X}$. If we take this option for $d \geq 1$ or if d is set to 0, the IDS algorithm is guaranteed to find an exact optimal solution for the MMR-MKP when sufficient computation time and memory space are available so that it runs until the search space of D-MMR-MKP($\hat{X}$) becomes empty. Both Logic-based Benders decomposition and branch-and-cut are applicable for solving MMR-MKP($\hat{x}$).

5.4 Computational Experiments

To the best of our knowledge, this is the first research on the MMR-MKP. We generated MMR-MKP instances from MKP benchmark instances (see [22]), with $m \in \{5, 10\}$, $n = 100$, and tightness ratio $\alpha = b_i / \sum_{j=1}^{n} a_{ij} \in \{0.25, 0.50, 0.75\}$. We then generated MMR-MKP instances by randomly taking the profit intervals, p_j^- and p_j^+, from ranges $[(1-\delta)p_j, p_j]$ and $[p_j, (1+\delta)p_j]$, respectively, with $\delta \in \{0.1, 0.3, 0.5\}$. As a result, we generated 18 MMR-MKP instances in total.

The algorithms proposed in this chapter were all coded in C++. We used IBM ILOG CPLEX, version 12.6, for solving linear programming and mixed integer linear programming problems. The CPLEX solver was also used to exactly solve the classical MKP, the Benders' master problems $P(X)$ and $\bar{P}(X)$, and problem D-MMR-MKP for the dual substitution heuristic. All experiments were carried out on a PC with two Core i7-5820K at 3.30 GHz and 32 GB RAM memory under Windows 10 operating system, by using a single core.

We report in Table 5.1 the results of all the algorithms described in Sections 5.2 and 5.3. Concerning the fixed scenario algorithm, the branch-and-cut, and the dual substitution method, each pair of entries shows the CPU time in seconds spent by the algorithm ("time") and the optimality gap in % ("gap"), i.e., the gap between the solution value from their optimal or lower bound values ("opt (LB)"). The values in the last column inside parentheses mean that, for those instances whose optimal values were not obtained by the proposed exact algorithms, we provide the best known lower bounds that were obtained by running the Logic-based Benders decomposition algorithm for one hour. Concerning Logic-based Benders decomposition, when it is terminated before an optimal solution is found, optimality gap information is not available because a feasible solution cannot be obtained in such cases. Concerning the iterated dual substitution (IDS) approach, we set the value of parameter d to 0. We set a time limit of 1000 seconds for each iteration of IDS to ensure that IDS can iterate at least 3–4 times within an hour; however, each iteration of IDS usually terminates in much shorter time and this time limit is rarely reached. The column "iteration" provides the iteration index during which the algorithm obtained the best solution value along with the total iterations in parentheses. Notation "t.l." indicates that the algorithm was not able to exactly solve the corresponding instances within the time limit (3600 seconds). An asterisk mark signifies that the algorithm(s) obtained the best optimality gap among the tested algorithms.

Table 5.1: Results of exact and heuristic algorithms

			Benders	Branch-and-Cut		Fixed Scenario		DS		IDS		
m	α	δ	time	time	gap	time	gap	time	gap	iteration	gap	opt (LB)
5	0.25	0.1	443.5	158	*0.0%	3.7	3.8%	20.3	*0.0%	1(56)	*0.0%	474.18
		0.3	t.l.	t.l.	0.5%	3.2	0.5%	19.9	*0.0%	1(52)	*0.0%	2436.37
		0.5	t.l.	t.l.	45.2%	1.4	*41.5%	31.5	*41.5%	1(44)	*41.5%	(3423.67)
5	0.50	0.1	192.8	112.4	*0.0%	1.3	*0.0%	2.5	*0.0%	1(63)	*0.0%	450.3
		0.3	t.l.	3544.8	*0.0%	4.2	*0.0%	13.5	4.2%	2(47)	*0.0%	1907.91
		0.5	t.l.	t.l.	56.4%	2.9	*48.4%	37.2	*48.4%	1(44)	*48.4%	(3219.34)
5	0.75	0.1	34.3	40.8	*0.0%	0.9	17.8%	0.5	17.1%	3(100)	*0.0%	246.63
		0.3	t.l.	1716.8	*0.0%	2.5	3.1%	17.2	13.5%	4(73)	*0.0%	1355.55
		0.5	t.l.	t.l.	31.3%	5.1	31.3%	19.5	33.8%	18(54)	*30.6%	(3072.95)
10	0.25	0.1	t.l.	2090.5	*0.0%	48.8	1.9%	106.6	7.2%	5(20)	*0.0%	743.64
		0.3	t.l.	t.l.	3.8%	44.8	3.8%	64.4	*0.0%	1(24)	*0.0%	2737.28
		0.5	t.l.	t.l.	56.4%	27.3	*45.2%	156.2	45.7%	2(14)	*45.2%	(3105.56)
10	0.50	0.1	t.l.	t.l.	30.0%	75.1	13.4%	31.3	*0.0%	1(9)	*0.0%	596.94
		0.3	t.l.	t.l.	70.1%	42.4	64.8%	493.9	*59.0%	1(5)	*59.0%	(1916.15)
		0.5	t.l.	t.l.	84.6%	48.5	79.9%	t.l.	77.4%	3(3)	*76.6%	(3681.01)
10	0.75	0.1	291.8	178.0	*0.0%	3.8	*0.0%	7.6	*0.0%	1(43)	*0.0%	302.04
		0.3	t.l.	t.l.	2.5%	3.1	*0.0%	22	*0.0%	1(32)	*0.0%	1724.77
		0.5	t.l.	t.l.	54.2%	48.7	54.5%	701.4	56.0%	5(8)	*50.5%	(3286.13)

Concerning the exact algorithms, for both Logic-based Benders decomposition and branch-and-cut approach, it becomes hard to obtain optimal solutions or even good lower bounds for the instances with large δ values, which results in large values in "gap" columns in Table 5.1. The branch-and-cut algorithm dominates the Logic-based Benders algorithm for most instances. In total, these two exact algorithms only solved 7 out of 18 instances to optimality.

Comparing the two constructive heuristics, both the dual substitution heuristic (DS in the table) and the fixed scenario approach under the median-profit scenario provide solutions with objective values quite close to the optimal values. However, for the instances with $\delta = 0.5$, the computation of the dual substitution algorithm becomes very expensive.

The proposed IDS algorithm obtained solutions with the best optimality gaps for all of the tested instances. The IDS algorithm provides exact optimal solutions for all the instances whose optimal values are known. We also observe that even when the computation time is insufficient, the IDS algorithm obtains good solutions with a small number of iterations. For the instances that the branch-and-cut algorithm cannot solve to optimality, the IDS algorithm always obtained better upper bounds within the same time limit.

5.5 Conclusion of the MMR-MKP

In this chapter, we studied a robust version of the multidimensional knapsack problem (MKP) called the min-max regret MKP.

We examined two exact methods: a Logic-based Benders decomposition approach and a branch-and-cut algorithm. For the Logic-based Benders decomposition, when it is halted with a time limit, the objective value of the obtained solution is a valid lower bound. On the other hand, the solution value obtained by the branch-and-cut framework provides a valid upper bound on the optimal regret when it terminates with a time limit. These two exact algorithms solved to optimality 7 out of 18 instances that we proposed.

We also examined two constructive heuristics: a fixed-scenario heuristic with the median-profit scenario, as well as a dual substitution (DS) heuristic that uses a mixed integer programming formulation obtained by replacing a subproblem with its dual. To improve the performance of the DS method, we proposed a row generation approach in which we call the iterated dual substitution (IDS) heuristic. We observed that the IDS method provided the best upper bounds over all the tested algorithms we examined, obtaining exact optimal solutions to all of the instances whose optimal values are known. All of the best solutions obtained by the IDS algorithm were found at an early stage of the iterations.

Chapter 6

Airline Crew Pairing Problem

6.1 Introduction

In this chapter, we solve a crew pairing problem by column generation approach.

The crew scheduling problem frequently appears in real-world applications, such as those in bus and rail transit industry. In this chapter, we consider a crew pairing problem in airline scheduling with a series of constraints and conditions particular to this industry. The crew costs constitute a high proportion (up to 20%) of total airline operation costs, and the number of airline flights increases with globalization. For this reason, a small percentage saving amounts to substantial reduction in expenses as mentioned by Anbil et al. [4] and Barutt and Hull [14].

Onodera and Mori raised an example that a Japanese airline company developed a knowledge-based system for crew scheduling in 1990, which cost about $4 million to build [77]. However, it got paid for itself in direct cost savings only in about 18 months.

Several approaches for the airline crew scheduling problem have been proposed in the literature, including exact algorithms such as tree search by Beasley and Cao (1996), branch-and-cut by Hoffman and Padberg [49], column generation by Barnhart et al. [11], as well as heuristic methods such as simulated annealing by Emden-Weinert and Proksch [35] and genetic algorithms by Levine [63].

In this chapter, we model the problem as a set covering problem with costs of columns defined by the number of person-days, and then we present an efficient method to find promising columns through a graph representation that describes connections between flights, where the size of the graphs is kept small by using the cost structure. To solve the problem of finding a promising column, we propose a branch-and-bound method based upon a resource constrained dynamic programming, which enables the algorithm to solve large-scale instances. Moreover, regularity is exploited to quickly generate many columns just by repeating itineraries. Such repeated itineraries are preferable so that crew members will have a low risk of making mistakes in duty time.

The rest of this chapter is organized as follows: In Section 6.2, we describe the crew scheduling problem in general, and we further present some requirements and constraints that are particular to some specific airline companies. In Section 6.3, we discuss a set covering model for this problem. In Section 6.4, we propose a column generation approach, which utilizes the branch-and-bound framework and dynamic programming. Section 6.4 also proposes some ideas for solving this set covering problem (SCP) efficiently. In Section 6.6, we present our computational results, and finally, we give concluding remarks in Section 6.7.

6.2 Problem Description

First we give a few definitions for later description. A *flight leg* (sometimes also called *segment*) is a single nonstop flight. A pairing is a sequence of flight legs that begins and ends at a crew base city, where the arrival airport of every flight leg in the sequence coincides with the departure airport of the next flight leg. A *deadhead* (DH) is a special flight leg such that the crew member assigned to it flies as a passenger to transport to the departure airport of another flight leg or to return to the departure city (crew base) at the end of a pairing.

Gopalakrishnan and Johnson described that airline scheduling usually consists of five planning stages. The last two stages among them, crew pairing and rostering, are usually referred to as crew scheduling problems [47].

As the input data of crew scheduling problem, a schedule consisting of all flight legs is provided before the crew pairing stage. A number of constraints also have to be satisfied for each pairing according to requirements from industrial applications. Each pairing has a cost associated with it. In our problem, we define the value of person-days as the cost of a pairing, where in our formulation, this value is defined to be q for a pairing if it consists of flight legs from q consecutive working days. One unit of person-day represents the amount of work done by one person in one working day. The objective of the crew pairing stage is to find a subset of all feasible pairings with the minimum total cost such that the subset contains every non-DH flight leg at least once (sometimes exactly once depending on the model definition).

In the rostering stage, a monthly (or weekly) schedule that can be operated by the crew is created by using the set of pairings generated at the crew pairing stage. Such a monthly (or weekly) schedule for the crew is called a *roster*. Although the exact number of crew members for the month (or week) becomes clear after crew rostering, this number is roughly determined after the pairing stage, and hence it is important to find a good solution in the pairing stage.

In this chapter, we concentrate on the approach for the crew pairing stage.

6.2.1 Constraints for Pairing

Each airline company may have several basic and specific constraints for defining feasible pairings. The basic constraints are listed as follows:

Basic Constraint 1: The first departure city in a pairing has to be the same as the last arrival city. In our problem, we define such a city as Tokyo. It signifies that only the flight legs departing from NRT or HND airport can be the first flight leg in a pairing. Similarly, only the flight legs arriving at NRT or HND airport are allowed to be the last flight leg.

Basic Constraint 2: A specified time is required for a crew member to transfer from a flight leg to the next one. In our formulation, for any flight leg, its departure time has to be at least 30 minutes later than the arrival time of its previous flight.

Basic Constraint 3: The duration of a pairing must not exceed a specified limit on the value of person-days, which is usually 4–6 days. In our model, we assume that each crew member is unable to work more than $N_{p_{\max}}$ (a given input) days, which means that the maximum value of person-days in a pairing is $N_{p_{\max}}$. Recall that in our formulation, the cost of a pairing is defined to be the value of person-days, and hence each pairing has a cost of at most $N_{p_{\max}}$. We set $N_{p_{\max}} = 5$ unless otherwise stated.

As in many papers in the literature, we also have constraints regarding the aircraft types; we restrict our attention to a single aircraft type in our scheduling.

Before explaining specific constraints, we give several definitions. An *interval time* t is defined as the time between the arrival and departure of two consecutive flight legs in a pairing, and t has to be at least 30 minutes as discussed above. A *break period* is defined as a short interval time satisfying $30 \leq t < 870$. If an interval time t is above or equals to 870 minutes, it is regarded as a *sleep period*. A *duty period* consists of a sequence of flight legs without sleep periods between them, i.e., sleep periods divide a pairing into duty periods. The flying time of a duty period is the sum of actual flying times of the flights in the duty period except for the flying times in deadhead flights. The maximum flying time f^k_{path} of a pairing k is the maximum flying time among all the flying times of the duty periods in pairing k. The working time of a duty period is the total working minutes in the duty period. A break period also contributes to the working time by the duration of the break period when it is less than 330 minutes; otherwise, it is counted as a constant working time of 90 minutes. The time for a crew member to board a deadhead flight is also counted as part of working time. The maximum working time w_{path}^k of a pairing k is the maximum working time among all the working times of the duty periods in pairing k. A landing number of a duty period is the total number of landings in the duty period. The maximum landing number l_{path}^k of a pairing k is the maximum landing number among all the landing numbers of the duty periods in a pairing k. For convenience, we define P_{all} to be the set of all feasible pairings.

We define three additional constraints as follows:

Specific Constraint 1: $f_{\text{path}}^k \leq N_{f_{\max}}$ for all $k \in P_{\text{all}}$;

Specific Constraint 2: $w_{\text{path}}^k \leq N_{w_{\max}}$ for all $k \in P_{\text{all}}$;

Specific Constraint 3: $l_{\text{path}}^k \leq N_{l_{\max}}$ for all $k \in P_{\text{all}}$,

where $N_{f_{\max}}$, $N_{w_{\max}}$ and $N_{l_{\max}}$ are the given upper bounds on the maximum flying time, the maximum working time and the maximum number of landings, respectively, which are set to $N_{f_{\max}} = 720$, $N_{w_{\max}} = 810$ and $N_{l_{\max}} = 5$ in this chapter.

A solution to the crew scheduling problem is considered feasible only when all the pairings selected by the solution satisfy all the above mentioned constraints, and all the non-deadhead flight legs are covered by the selected pairings. Note that even though the model in this section was formulated based on real-world data from a company and is quite complicated, some of the constraints are simplified from real ones and the parameter values are not necessarily the same as those in the real-world applications.

6.2.2 Instances and Experimental Environment

All the computational experiments in this chapter are conducted on four instances, named I, II, III and IV, that were generated based on real-world flight data. For I, II and IV, we aim to solve a weekly airline crew scheduling problem. For convenience, we assume that the time horizon of every instance from I, II and IV consists of 17 days with all the flight legs in the first and the last N_{dummy} days specified as deadhead. These deadheads at the beginning and the end help to cover the target flights in the middle core days. Instance III is a monthly flight data that is also sandwiched by N_{dummy} days with only deadheads before and after the core period. Since every pairing in any optimal solution includes at least one target flight, it is sufficient to set N_{dummy} to $N_{p_{\max}} - 1$, where $N_{p_{\max}}$ is the maximum value of person-days in a pairing. In this chapter, we set N_{dummy} to 5, because the largest value of $N_{p_{\max}}$ we tested in Section 6.6.2 is 6. The information of the instances is shown in Table 6.1. The number of flights including deadhead flights for each instance is shown in column "#Flight."

All experiments in this chapter are carried out on a PC with 3.30 GHz CPU and 32 GB RAM memory, where the computation is executed on a single core. The algorithms

Table 6.1: Instance information

Instance	#Day	#DummyDay	#Flight	#DH
I	17	10	918	540
II	17	10	2201	1295
III	41	10	2214	540
IV	17	10	11514	6777

proposed in this chapter are all coded in C++ unless otherwise stated. All of them were built and compiled under Microsoft Visual Studio 2012.

6.3 Set Covering Model

Our algorithm solves the crew pairing problem in two steps: the first stage generates feasible pairings, and the second stage selects a good subset of these pairings to cover all the flight legs. In most cases, it is impractical to generate all the feasible pairings in the first stage, since the number of such pairings grows exponentially with the number of flight legs. To deal with this, we propose a column generation approach that is discussed in Section 6.4. The second stage can be modeled as a set covering or set partitioning problem. Since deadheads are allowed in our crew scheduling problem, our objective is to find a set of pairings with minimum cost such that each flight leg is covered by at least one pairing. Let a_{ik} be a binary value that equals 1 when pairing k covers flight leg i, otherwise, $a_{ik} = 0$. The cost c_k is the value of person-days of pairing k. This problem can be formulated as a set covering problem SCP(P) as follows:

$$\begin{aligned}
&\text{minimize} && \sum_{k\in P} c_k x_k && \\
&\text{subject to} && \sum_{k\in P} a_{ik} x_k \geq 1, && \forall i \in F_{\text{all}} \\
& && x_k \in \{0,1\}, && \forall k \in P,
\end{aligned}$$

where P is a subset of P_{all} and F_{all} is the set of all non-deadhead flight legs. The binary decision variable x_k is a 0-1 variable associated with the kth pairing. If the pairing k is selected, then $x_k = 1$, and otherwise $x_k = 0$. When $P = P_{\text{all}}$ holds, the problem SCP(P_{all}) becomes the original problem of finding an optimal set of pairings.

6.4 Column Generation Approach

Compared with the straightforward method of enumerating all the feasible pairings, a column generation approach has an advantage that it provides an optimal solution to the LP (linear programming) relaxation SCP$^*(P_{\text{all}})$ of SCP(P_{all}) by iteratively solving SCP$^*(P)$ to optimality for subsets P whose sizes are relatively small. We call an SCP$^*(P)$ a *master problem*. For any subset P, the cost of an optimal solution to SCP$^*(P)$ can be reduced further by adding good feasible pairings to P. Such pairings can be found by solving a problem called the *pricing problem* that is defined based on an optimal solution to the dual of SCP$^*(P)$ for the current subset P. The approach stops when no good pairing can be found to improve the current solution. At this moment, the current solution is proved to be optimal to SCP$^*(P_{\text{all}})$. Furthermore, in view of the experience over the past researches, the solution obtained by solving SCP(P) is known to be relatively good to SCP(P_{all}). In this section, we propose an efficient algorithm for finding good pairings. We also focus on the initial pairing generation and regularity.

6.4.1 Column Generation

We consider the LP relaxation problem SCP$^*(P)$:

$$\text{minimize} \quad \sum_{k\in P} c_k x_k \tag{6.4.1}$$

$$\text{subject to} \quad \sum_{k\in P} a_{ik} x_k \geq 1, \qquad \forall i \in F_{\text{all}} \tag{6.4.2}$$

$$0 \leq x_k \leq 1, \qquad \forall k \in P. \tag{6.4.3}$$

Its dual problem DSCP$^*(P)$ is formulated as follows:

$$\text{maximum} \quad \sum_{i\in F_{\text{all}}} u_i \tag{6.4.4}$$

$$\text{subject to} \quad \sum_{i\in F_{\text{all}}} a_{ik} u_i \leq c_k, \qquad \forall k \in P \tag{6.4.5}$$

$$u_i \geq 0, \qquad \forall i \in F_{\text{all}}. \tag{6.4.6}$$

We iteratively solve this LP problem with its dual problem. Denoting an optimal solution to DSCP$^*(P)$ as u^*, the pricing problem PRICE(u^*) to find a pairing k from P_{all} to be added to P can be defined as follows:

$$\max_{k\in P_{\text{all}}} \frac{1}{c_k} \sum_{i\in F_{\text{all}}} a_{ik} u_i^*. \tag{6.4.7}$$

Let $\sigma(u^*)$ be the optimal value of PRICE(u^*) and let k^* be an optimal solution. If $\sigma(u^*) > 1$ holds, then the optimal value of DSCP$^*(P)$ can be updated by adding pairing k^* to P. The process is iterated until $\sigma(u^*) \leq 1$ holds.

6.4.2 Graph Description

The problem of finding a sequence of flight legs can be formulated as a routing problem in digraphs, where the flight legs are associated to nodes. We link two nodes (i, j) with a directed edge if the following two conditions are satisfied.

Condition 1: The arrival airport of flight leg i coincides with the departure airport of j.

Condition 2: The departure time of flight leg j is at least 30 minutes later than the arrival time of i.

For verifying the maximum person-day and reducing the computation time in column generation, we generate several subgraphs for one instance. We define $G(p, q)$ as the subgraph corresponding to the period from the pth day to the $(p - 1 + q)$th day for all $p \in \{1, 2, \dots, N_d - q + 1\}$ and $q \in \{1, 2, \dots, N_{p_{\max}}\}$, where N_d is the total days of the instance. Therefore, each instance has $(2N_d - N_{p_{\max}} + 1)N_{p_{\max}}/2$ subgraphs. For example, for instance I whose number of days is 17, we first create 17 subgraphs for each day involving only those flight legs whose departure and arrival times are both in this day. Then, for every pair of consecutive two days, we apply a similar rule to generate 16 graphs. Similar rules are applied to the cases of $3, 4, \dots, N_{p_{\max}}$ consecutive days. As a result, we obtain 75 subgraphs in total if $N_{p_{\max}} = 5$.

For each subgraph, we connect a source node s to the flight legs whose departure is on the first day in Tokyo. A sink node t is linked from the flight legs whose arrival is on the last day in Tokyo. Finally, we remove the nodes that are not reachable from s and those not reachable to t, because such nodes are not necessary.

Note that all the subgraphs are directed acyclic graphs (DAG) and any path from s to t represents a pairing with q person-days. Although such an s–t path generated from these subgraphs is not necessarily feasible, it must satisfy all the basic constraints.

With these subgraphs, the pricing problem PRICE(u^*) can be decomposed into subproblems PRICE(u^*, p, q) for all the subgraphs $G(p, q)$, where PRICE(u^*, p, q) is the pricing problem to find a pairing that maximizes $(1/c_k) \sum_{i \in F_{\text{all}}} a_{ik} u_i^*$ among those that correspond to s–t paths in $G(p, q)$. Because c_k is the same for all such pairings k (to be more precise, $c_k = q$ holds for every pairing k that corresponds to an s–t path in $G(p, q)$), this problem becomes a constrained longest path problem in directed acyclic digraphs $G(p, q)$. Let $\sigma(u^*, p, q)$ be the optimal solution value of PRICE(u^*, p, q) and $k^*(u^*, p, q)$ be an optimal solution. If there is a graph $G(p,q)$ that satisfies $\sigma(u^*,p,q) > 1$, then the optimal value of DSCP*(P) can be updated by adding pairing $k^*(u^*,p,q)$ to P. Otherwise, i.e.,

$$\sigma(u^*,p,q) \leq 1 \tag{6.4.8}$$

holds for all the subgraphs $G(p,q)$, we terminate the column generation.

In the column generation phase, we adopt the following strategy. We examine subgraphs $G(p,q)$ one by one for possible pairs of p and q, solving the corresponding pricing problem, and whenever a pairing that violates (6.4.8) is found by solving the pricing problem PRICE(u^*) for a subgraph $G(p,q)$, we add such a pairing to P and immediately start a new iteration (i.e, we stop examining the remaining subgraphs and immediately move to the phase of solving SCP*(P) for the updated P).

6.4.3 Initial Pairing Set Generation

The set covering problem SCP(P) is feasible only when the initial pairing set P can cover all the non-DH flight legs. Our initial pairing set generation method to generate such a P starts from $P = \emptyset$ and consists of two steps.

In the first step, we consider an iterative process based on depth first search (DFS). For each subgraph, we define Q as the set of all the nodes directly connected from source node s and execute a DFS from each node in Q. In choosing the next candidate node in DFS, we divide the unvisited nodes that are connected from the current node into two sets: currently uncovered nodes and covered nodes, where a node is covered if there is a pairing in P containing the flight leg corresponding to the node and uncovered otherwise. If the set of uncovered nodes is not empty, we choose from this set the node whose departure time is closest to the arrival time of the current node; otherwise, we choose such a node from the set of covered nodes. Whenever the DFS reaches a node that is connected to the sink t, it checks if the current path from s to t satisfies the pairing constraints explained in Section 6.2.1, and if it does, we add the obtained s–t path into P, terminate the current DFS, and start a new DFS from another node in Q that has not been used as the starting node of DFS. Whenever we start a DFS from a node in Q, all the nodes are labeled unvisited, i.e., the new DFS can visit those nodes that have been visited by a former DFS. A set of such calls to DFS for a subgraph, which we call a DFS probe, comes to an end when DFS has been executed from every node in Q. We repeat the process of invoking DFS probes to all subgraphs, where one round consists of DFS probes with a DFS probe to every subgraph, and such rounds are repeated until no more uncovered nodes have become covered in a round. In each round, the time complexity for one graph is $O(|Q|E_{G(p,q)})$, where $E_{G(p,q)}$ denotes the number of edges in graph $G(p,q)$.

Even this simple approach works effectively for the tested instances as shown in Table 6.2. The first and the second columns express the instance name ("Instance") and the number of non-DH nodes ("#Non-DH") for each instance. Each value in the third column ("#Pairing") shows the number of pairings in the obtained set P, and the fourth column ("#Round") shows the number of rounds executed in the first step. The last two columns represent the number of uncovered nodes ("#Uncovered") after the first step and the coverage (i.e., the ratio of nodes that are covered by the obtained set P) for each instance.

Table 6.2: Node coverage by the path generation with DFS

Instance	#Non-DH	#Pairing	#Round	#Uncovered	Coverage
I	378	66	3	0	100.00%
II	906	251	3	0	100.00%
III	1674	258	4	1	99.94%
IV	4737	1315	4	0	100.00%

For those non-DH nodes that remain uncovered, the second step follows. Assume that node i is uncovered after the first step. Let V_i^- and V_i^+ denote the set of nodes reachable to i and from i, respectively, in the entire graph. For every graph $G(p,q)$ that contains node i, we generate a subgraph $G_i(p,q)$ induced by the node set $V_i^- \cup \{i\} \cup V_i^+$. Then, we modify $G_i(p,q)$ by removing all the edges connecting from a node in V_i^- to a node in V_i^+. Recall that all nodes are reachable from s and to t in $G(p,q)$, and the same also holds for $G_i(p,q)$. This ensures that any s–t path in $G_i(p,q)$ must contain node i. Furthermore, it is also clear that any s–t path in $G_i(p,q)$ satisfies the basic constraints in Section 6.2.1. Hence, our problem reduces to the problem of finding an s–t path that satisfies all the specific pairing constraints.

We consider a dynamic programming (DP) for solving this problem. For each node h in graph $G_i(p,q)$, we maintain a matrix (w_{lf}) where the value w_{lf}^h of the (l,f)-element represents the minimum working time in a path among all feasible paths from source node s to h such that (after the latest sleep period if such a period exists) the number of landings is at most l and the total flying time is at most f. Note that the matrix size is $N_{l_{\max}} \times N_{f_{\max}}$ because of Specific Constraint 1 and 3. Denote by $w_{(jh)}$ the working time during the period between flight leg j and h, and if $j = s$ or $h = t$, let $w_{(jh)} = 0$. Similarly, we define w_h as the working time of flight leg h, and if $h = s$ or $h = t$, let $w_h = 0$. Let all the values w_{lf}^s for node s be 0. Then the cells for all other nodes can be computed by a forward programming:

$$w_{lf}^{h} = \begin{cases} \min\limits_{j \in V_h^-} w_{(l-l_j),(f-f_j)}^{j} + w_{(jh)} + w_h & \text{if } \min\limits_{j \in V_h^-} w_{(l-l_j),(f-f_j)}^{j} \le N_{w_{\max}} \text{ and} \\ & (j,h) \text{ is not a sleep period} \\ w_h & \text{if } \min\limits_{j \in V_h^-} w_{(l-l_j),(f-f_j)}^{j} \le N_{w_{\max}} \text{ and} \\ & (j,h) \text{ is a sleep period} \\ N_{w_{\max}} + 1 & \text{otherwise.} \end{cases}$$

The value of the $(N_{l_{\max}}, N_{f_{\max}})$-element at the sink node t indicates whether a feasible path exists in the current subgraph. If its value is lower than or equals to the maximum working time $N_{w_{\max}}$, a feasible path exists, and the target uncovered node i will be covered by such a path. This feasible path can be obtained by tracking back the DP cells through the path starting from the $(N_{l_{\max}}, N_{f_{\max}})$-element at the sink node. The path generated with this procedure must be feasible, since this DP approach respects all the constraints.

The time complexity of this DP is $O(N_{l_{\max}} N_{f_{\max}} N_{G_i(p,q)} E_{G_i(p,q)})$, where $N_{G_i(p,q)}$ and $E_{G_i(p,q)}$ denote the number of nodes and edges in graph $G_i(p,q)$. In practice, the computation time can further be reduced if the input values are multiples of an integer. For example, for the tested instances I to IV, all the time-related input, including flying time, is divisible by 5, and $N_{f_{\max}}$ can be reduced from the original value 720 to 144, which reduces the computational cost to one fifth.

We apply this scheme to every subgraph $G(p,q)$ until a valid path is found. The instance is proved to be infeasible, if no feasible path is found for a target node i.

6.4.4 Lower Bound

For large-scale instances, it becomes hard for the column generation approach to continue the search until all the necessary columns are generated, i.e., until the termination condition (6.4.8) is satisfied for all subgraphs $G(p,q)$. In this section, we consider a lower bound on the optimal value of SCP$^*(P_{\text{all}})$ that can be obtained by solving the pricing problems PRICE(u^*, p, q) for all p and q even if the termination condition (6.4.8) is not satisfied in the current iteration.

Recall that P is a subset of P_{all}, and u^* is an optimal solution to DSCP$^*(P)$, and moreover, $\sigma(u^*, p, q)$ denotes the optimal value of PRICE(u^*, p, q). If $\sigma(u^*, p, q) \le 1$ holds for all the subgraphs $G(p,q)$, SCP$^*(P_{\text{all}})$ has been exactly solved and the optimal value is $\sum_{i \in F_{\text{all}}} u_i^*$ due to the duality theorem. Next we consider the situation where the termination condition (6.4.8) is not satisfied, i.e.,

$$\sigma(u^*, p, q) > 1 \tag{6.4.9}$$

holds for some $G(p,q)$. Recall that the flight legs are associated to nodes, and let i also denote the node corresponding to flight i. Denote by $V(p,q)$ the node set of $G(p,q)$, and define $R(i)$ to be the set of subgraphs that contain flight leg i, i.e.,

$$R(i) = \{G(p,q) \mid i \in V(p,q)\}.$$

Let Δ_i be defined as follows:

$$\Delta_i = \min_{G(p,q)\in R(i)} \frac{1}{\sigma(u^*,p,q)} \qquad\qquad \forall i \in F_{\text{all}}.$$

We denote by $P_{\text{all}}^{(p,q)}$ the set of all feasible pairings (s–t paths) in subgraph $G(p,q)$ and define $\hat{u}_i = \Delta_i u_i^*$ for each flight leg i. Then $\hat{u}$ is a feasible solution of DSCP$^*(P_{\text{all}})$ because the following inequality holds for all the subgraphs $G(p,q)$:

$$\begin{aligned}
\sigma(\hat{u},p,q) &= \max_{k\in P_{\text{all}}^{(p,q)}} \frac{1}{c_k} \sum_{i\in F_{\text{all}}} a_{ik}\Delta_i u_i^* && (6.4.10)\\
&= \max_{k\in P_{\text{all}}^{(p,q)}} \frac{1}{c_k} \sum_{i\in F_{\text{all}}} \min_{G(p',q')\in R(i)} \frac{1}{\sigma(u^*,p',q')} a_{ik}u_i^* && (6.4.11)\\
&\le \max_{k\in P_{\text{all}}^{(p,q)}} \frac{1}{c_k} \sum_{i\in F_{\text{all}}} \frac{1}{\sigma(u^*,p,q)} a_{ik}u_i^* && (6.4.12)\\
&= \frac{1}{\sigma(u^*,p,q)} \max_{k\in P_{\text{all}}^{(p,q)}} \frac{1}{c_k} \sum_{i\in F_{\text{all}}} a_{ik}u_i^* = 1. && (6.4.13)
\end{aligned}$$

The inequality from (6.4.11) to (6.4.12) holds for the following reason. For every i and k, if a_{ik} equals zero, the term corresponding to i and k equals zero both in (6.4.11) and (6.4.12). Otherwise (i.e, if a_{ik} equals one), it means that path k contains node i. Accordingly, $G(p,q)$ is in set $R(i)$, which indicates that the term in (6.4.12) is a candidate among the minimize function range in (6.4.11), and the corresponding term in (6.4.11) is less than or equals to the one in (6.4.12).

As a result, the objective function value $\sum_{i\in F_{\text{all}}} \hat{u}_i$ gives a lower bound on the optimal value of SCP$^*(P_{\text{all}})$ as well as that of SCP(P_{all}).

6.4.5 DP-based Algorithm for Pricing Problem

We solve the pricing problem using a DP-based branch-and-bound algorithm. Before devising this method, we considered a DP approach similar to the one discussed in Section 6.4.3 by extending the matrix at each node to a 3-dimensional table. However, such an additional dimension causes a great increase in both computation time and memory. For this reason, and from preliminary experimental results, we decided to adopt a branch-and-bound framework using a relaxation of the above DP for bounding operations. Our method consists of two phases and can be outlined as follows.

In the first phase, it creates three independent DP lists for each node corresponding to the three constraints regarding working time, flying time and landings, respectively. In the DP list of node j with respect to working time, the value $u_j^w(r)$ of the rth cell represents the maximum obtainable price along a path among all paths from node j to the sink node such that the total working time (before the first sleep period if such a period exists) is at most r, where the price of a path is the sum of u_i^* for all nodes i on the path. Similarly, we prepare the DP lists $u_j^f(\cdot)$ and $u_j^l(\cdot)$ for both flying time and landings, respectively. Note that by reducing the 3-dimensional table into three independent lists, each value in the DP list only indicates an upper bound, since only the constraint associated with the list is guaranteed, and the path realizing the value in the list may violate one or both of the other two constraints. These kind of DP list cannot provide us with a feasible pairing, but with an upper bound on the price value, which is important for bounding operations.

In the second phase, we use a depth-first branch-and-bound search to generate an optimal path. The algorithm generates partial paths from s by expanding the current path along an edge from the last node of the path or by backtracking whenever the current path becomes infeasible or it is concluded that it does not lead to a desirable path (i.e., a bounding operation). Suppose that the current path $\hat{k}$ is a partial path from source node s to a node i in a graph $G(p,q)$. Denote by $F(\hat{k})$ the node set of path $\hat{k}$, and we define $f_{\hat{k}}^*$, $w_{\hat{k}}^*$ and $l_{\hat{k}}^*$ to be the flying time, working time and landing number of path $\hat{k}$ from the last sleep period to node i, or from s to i if $\hat{k}$ contains no sleep period. In expanding the current path $\hat{k}$ by appending a node j at the end of $\hat{k}$, we choose a node whose departure time is earliest among those in V_i^+ that have not been examined yet as a candidate to append to the current path $\hat{k}$. If the path expanded by appending such a node j is feasible, we examine the following three conditions:

$$\begin{cases} \sum_{i\in F(\hat{k})} u_i^* + u_j^f(N_{f_{\max}}) \le q & \text{if } (i,j) \text{ is a sleep period} \\ \sum_{i\in F(\hat{k})} u_i^* + u_j^f(N_{f_{\max}} - f_{\hat{k}}^*) \le q & \text{otherwise;} \end{cases}$$

$$\begin{cases} \sum_{i\in F(\hat{k})} u_i^* + u_j^w(N_{w_{\max}}) \le q & \text{if } (i,j) \text{ is a sleep period} \\ \sum_{i\in F(\hat{k})} u_i^* + u_j^w(N_{w_{\max}} - w_{\hat{k}}^* - w_{(ij)}) \le q & \text{otherwise;} \end{cases}$$

$$\begin{cases} \sum_{i\in F(\hat{k})} u_i^* + u_j^l(N_{l_{\max}}) \le q & \text{if } (i,j) \text{ is a sleep period} \\ \sum_{i\in F(\hat{k})} u_i^* + u_j^l(N_{l_{\max}} - l_{\hat{k}}^*) \le q & \text{otherwise.} \end{cases}$$

Recall that our objective is to find a path in $G(p,q)$ that satisfies the inequality (6.4.9), which means that the total of prices u_i^* on the nodes in the path has to be greater than $c_k = q$. If one of the above three conditions is satisfied, we can conclude that no path with total price greater than q can be generated by further extending from node j the current partial path ($\hat{k} \to j$), i.e., a bounding operation can be applied to node j and a backtracking follows.

The computational results of a simple tree search (that backtracks only if the current path becomes infeasible) and the proposed DP-based branch-and-bound method are shown in Table 6.3. We set the time limit to 7200 seconds for both algorithms. The computation time in seconds for solving the LP relaxation SCP$^*(P_{\text{all}})$ by column generation ("Time") and the objective values ("ObjectValue") to SCP$^*(P)$ for the set P when the column generation terminated are shown in Table 6.3, where "t.l." signifies that the time limit was reached before the column generation stopped with condition (6.4.8) satisfied for all graphs. The column "#Iteration" expresses the number of LP relaxation problems SCP$^*(P)$ that have been solved during the execution.

The branch-and-bound with DP has better performance for all instances.

Table 6.3: The results of simple tree search and DP-based branch-and-bound method

	Simple tree search			B&B with DP		
Instance	#Iteration	ObjectValue	Time	#Iteration	ObjectValue	Time
I	151	228.0	t.l.	3135	92.8	56
II	342	749.0	t.l.	23309	247.4	t.l.
III	273	1115.0	t.l.	14688	382.4	t.l.
IV	53	5769.0	t.l.	3679	2433.3	t.l.

6.4.6 Regularity

Most flight legs are regularly scheduled, e.g., a flight from an airport to another is scheduled with the same departure and arrival times for every weekday. This section considers a method to exploit such regularity. We call two pairings twins if for every corresponding pair of flight legs, the departure and arrival airports are the same, and the departure and arrival times are the same but not on the same day, where the intervals between the two corresponding flight legs are the same for all pairs. When the column generation obtains a valid good pairing to be added to P, all of its twin pairings are also added to P.

We compared the computation time of the cases where the method of adding twins is adopted or not. The results for the case without this idea are shown in Table 6.4, and Table 6.5 shows the results when the idea is incorporated.

We can observe from the computational results that this idea improves the speed of column generation for all the instances. This might be because the bottleneck of our

Table 6.4: The results of the algorithm without twins

Instance	#Iteration	#Pairing	ObjectValue	Time
I	3135	3201	92.8	56
II	23309	23560	247.4	t.l.
III	14688	14947	382.4	t.l.
IV	3679	4994	2433.3	t.l.

Table 6.5: The results of the algorithm with twins

Instance	#Iteration	#Pairing	ObjectValue	Time
I	878	8228	92.8	37
II	10944	97582	247.2	5897
III	653	22064	380.8	1140
IV	821	9042	2238.1	t.l.

approach for these instances is the time to find a good pairing, and it is advantageous to add more than one pairing at each iteration. Note however that more than 20,000 twin pairings were generated only in 653 iterations for instance III, and a different conclusion might be drawn for larger instances if the time horizon becomes much longer.

6.5 SCP Heuristic Approach

The column generation approach stops the iteration if no good pairing can be generated to improve the SCP$^*(P)$, and if this stopping criterion is satisfied, the LP relaxation of the original SCP, SCP$^*(P_{\text{all}})$, has been solved to optimality. However, the obtained solution to SCP$^*(P)$ can have fractional elements.

As the last step, we solve the SCP in its integer programming (IP) formulation described in Section 6.3. The SCP is known to be NP-hard, and many good exact and heuristic algorithms have been proposed. In this chapter, we consider two approaches: One is to solve this integer programming problem by using an IP solver, IBM ILOG CPLEX, and the other is to solve it with a heuristic approach based on a 3-flip neighborhood local search algorithm (3FNLS) proposed by Yagiura et al. [107]. If sufficient time and memory were available, CPLEX would be able to solve SCP(P) to optimality. However, the 3FNLS obtained better results for all instances under a time limit of one hour. Table 6.6 shows the results of solving SCP(P) by CPLEX and 3FNLS, where P is obtained by the proposed column generation approach.

Table 6.6: The objective values of SCP(P) obtained by CPLEX and 3FNLS in one hour

	I	II	III	IV
CPLEX	97	288	454	2533
3FNLS	96	255	406	2456

6.6 Computational Results

We now present computational results. The heuristic algorithms proposed for SCP by Yagiura et al. [107] were coded in C, and we set a time limit of 3600 seconds.

6.6.1 Results of the Proposed Approach

Table 6.7 shows the results of the proposed column generation approach, where the columns show the number of pairings generated by the algorithm ("#Pairing"), the computation time in seconds for solving the LP relaxation by column generation ("Time (LP)") and the objective values ("Value (LP)") to SCP*(P). The objective values obtained by solving the resulting SCP(P) by the 3FNLS are listed in the last column ("Value (SCP)"). As discussed in Section 6.4, the rounded up value of the objective value of LP relaxation, when the column generation stops normally with the termination condition (6.4.8), gives a lower bound on the optimal value of SCP(P_{all}), which are shown in column "LB" in the table. Note that for instance IV, the column generation was stopped with a time limit of 7200 seconds (denoted "t.l." in the table), and the value in column "LB" shows a lower bound obtained by the method proposed in Section 6.4.4. We can observe that the gap between the objective value of SCP (as IP) and its lower bound is within 10% for all the instances except IV.

Table 6.7: Computation times in seconds of our set covering approach

Instance	#Pairing	Time (LP)	Value (LP)	LB	Value (SCP)
I	8228	37	92.8	93	96
II	97582	5897	247.2	248	255
III	22064	1140	380.8	381	406
IV	8899	t.l.	2267.5	837	2456

We also tested an existing software module developed by a company and designed for solving (almost) the same problem. We applied it to the same instances and analyzed the two approaches from various aspects, including some criteria that are not explicitly considered in our formulation. It was observed that solutions obtained by our approach have less number of total person-days and lower hotel expenses for crew members. On the other hand, the software obtained solutions with smaller number of deadheads and with more regularity.

6.6.2 Maximum Person-Days

Recall that we defined the maximum value of person-days $N_{p_{\max}}$ in Section 6.2.1. This input parameter restricts the search space P_{all} of pricing problem PRICE(u^*). Let P^{α}_{all} be the set of all feasible pairings under the condition $N_{p_{\max}} = \alpha$. We can easily prove the following lemma from the fact that $\alpha > \beta$ implies $P^{\beta}_{\text{all}} \subseteq P^{\alpha}_{\text{all}}$.

Lemma 6.6.1. *If $\alpha > \beta$ holds, the optimal value of* SCP$^*(P^{\alpha}_{\text{all}})$ *is not more than that of* SCP$^*(P^{\beta}_{\text{all}})$.

The same result also holds for SCP$(P^{\alpha}_{\text{all}})$ and SCP(P^{β}_{all}).

The parameter $N_{p_{\max}}$ is usually decided by practical reasons and is given as an input, but from computational point of view, it gives a trade-off between the quality of obtained solutions and computational cost in solving SCP$^*(P_{\text{all}})$ by our column generation approach. Intuitively, we might consider that if we use greater $N_{p_{\max}}$, a better objective value of SCP(P) can be obtained by a set P generated by the proposed column generation provided that it stops normally with the termination condition (6.4.8) (no good pairing can be added to improve the SCP$^*(P)$). However, this is not always the case as observed through the experiments.

Table 6.8: Results of Instance I

$N_{p_{\max}}$	#Pairing	Time (LP)	Value (LP)	LB	Value (SCP)
1	-	-	-	-	-
2	1860	1	98.1	99	100
3	5567	11	93.2	94	95
4	8063	34	92.9	93	95
5	8228	37	92.8	93	96
6	11155	80	92.6	93	96

Table 6.9: Results of Instance II

$N_{p_{\max}}$	#Pairing	Time (LP)	Value (LP)	LB	Value (SCP)
1	-	-	-	-	-
2	-	-	-	-	-
3	33466	1390	251.1	252	260
4	69679	3670	247.7	248	256
5	97582	5897	247.2	248	255
6	98736	t.l.	247.2	248	254

Table 6.10: Results of Instance III

$N_{p_{\max}}$	#Pairing	Time (LP)	Value (LP)	LB	Value (SCP)
1	-	-	-	-	-
2	7945	43	407.6	408	427
3	20892	535	382.6	383	395
4	29303	1161	381.9	382	398
5	22064	1140	380.8	381	406
6	27655	1712	380.6	381	408

Table 6.11: Results of Instance IV

$N_{p_{\max}}$	#Pairing	Time (LP)	Value (LP)	LB	Value (SCP)
1	-	-	-	-	-
2	-	-	-	-	-
3	-	-	-	-	-
4	-	-	-	-	-
5	8899	t.l.	2267.5	837	2456
6	10134	t.l.	2350.9	809	2555

Table 6.8–6.11 show the computational results for each instance under different $N_{p_{\max}}$ values. The methods and parameter settings are the same as the experiments shown in Table 6.7, as well as the meaning of each column. The mark "-" signifies that no feasible solution exists under the corresponding value of $N_{p_{\max}}$. For all instances, when we only consider crew pairings with up to 2 person-days (i.e., when $N_{p_{\max}} \leq 2$), the

problem becomes infeasible or the results are much worse than other cases. The values in columns “Time (LP)” and “Value (LP)” in Table 6.8–6.10 show that there is a trade-off between the quality of obtained solutions and computational costs during the column generation process. However, the final results in “Value (SCP)” do not always decrease as the values of $N_{p_{\max}}$ increase for some instances such as Instance I and III, even though its LP relaxation problem $\mathrm{SCP}^*(P_{\mathrm{all}})$ is solved to optimality by column generation. The results in Table 6.8–6.11 suggest that our column generation heuristic can lead to a better objective value of $\mathrm{SCP}(P)$ by applying it with a smaller value of $N_{p_{\max}}$ than its given value. Moreover, although the number of all feasible pairings can exponentially increase with $N_{p_{\max}}$, the total computation time does not grow with $N_{p_{\max}}$ so drastically. Hence, it might be worth trying to invoke the proposed algorithm iteratively with different values of $N_{p_{\max}}$, e.g., from 2 to the given maximum value of person-days.

6.7 Conclusion of the Airline Crew Pairing Problem

We studied a crew pairing problem in which the objective is to minimize the total person-days subject to some basic and specific constraints. We modeled it as a set covering problem.

We presented a column generation approach that incorporates several ideas. We modeled the problem of finding a desirable pairing, in both initial pairing generation and pricing problem, as a problem of finding a longest path in a graph under some resource constraints. We proposed a two-step method for generating initial pairings, which consists of a depth first search, and a dynamic programming (DP) algorithm for covering target flight legs. We then proposed a DP-based branch-and-bound method in solving the pricing problem. This method outperformed a branch-and-bound algorithm without bounding operations by DP. We also considered regularity based on the fact that most flight legs are regularly scheduled. For large-scale instances, we presented a lower bound that can be obtained even if the column generation approach is stopped before the termination condition is satisfied.

Through computational experiments, we confirmed that the proposed approach successfully obtained good solutions for most of the tested instances with up to 10,000 flights.

Chapter 7

Conclusion

Generation methods including row generation and column generation have been two of the hottest topics for combinatorial optimization problems over the past sixty or more years.

In Chapter 2 and 3, we introduced row generation and column generation approaches, respectively. We listed representative historical contributions, provided a standard understanding, and presented a survey with an emphasis on their use in combinatorial optimization problems. For representative milestones of row generation, we described the classical Benders decomposition algorithm for solving the mixed integer programming (MIP) problem, as well as a further extension, known as logic-based Benders decomposition. For column generation, we described Dantzig-Wolfe decomposition. It focuses on utilizing block structure, which can be naturally identified in many problems in real-world applications. The correctness and the convergence of both decomposition algorithms for linear programming (LP) problems are guaranteed by the fact that there is a finite number of extreme points in a convex set. The effectiveness is achieved by considering a restricted problem with a relatively small subset of the rows (or columns), significantly easier to solve. We also discussed the relationship between row and column generation. It should be noted that both generation methods are constructed as iterative procedures to solve combinatorial optimization problems with a huge number of rows or columns, and their combinations with other well-known methods, such as branch-and-bound, often enhance the overall performance of the resulting algorithms for many combinatorial optimization problems.

Throughout this thesis, we discussed how to use the generation algorithms to develop efficient exact and heuristic algorithms for NP-hard problems, for which the best algorithmic strategy may not be obvious. Three problems were studied in Chapter 4, 5 and 6, the min-max regret generalized assignment problem (MMR-GAP), the min-max multidimensional knapsack problem (MMR-GAP), and the crew pairing problem.

For the MMR-GAP, we proved that the decision version of the MMR-GAP is Σ_2^p-complete. We first examined two heuristics, including a fixed-scenario algorithm and a dual substitute methods. We observed that the dual substitution method obtained better solutions than the fixed-scenario heuristic for the instances in types A, B, and C. However, for the instances in types D and E with a large number of jobs, the dual substitution becomes computationally expensive compared to the fixed-scenario heuristic. Regarding the proposed exact algorithms, we designed a logic-based Benders decomposition, and a branch-and-cut approach with Benders cuts. We further proposed a Lagrangian-based branch-and-cut algorithm, into which we incorporated several ideas. At each node, we used a Lagrangian lower bound that is proven to be stronger than the LP bound. For computing this bound, we considered a technique solving m 0-1 knapsack problems by dynamic programming. This computation is further exploited for variable fixing, for which we showed that by using a two-direction DP table, the time complexity was reduced from $O(n^2b)$ to $O(nb)$, where n is the number of jobs and b is the sum of capacities b_i for all agents i. Moreover, we discussed and computationally analysed an efficient cut selection criterion for Benders' cuts in the Lagrangian-based branch-and-cut approach. The resulting Lagrangian-based branch-and-cut algorithm performed satisfactorily on benchmark instances.

For the MMR-MKP, we considered a row generation heuristic by using the dual substitution, which we call the iterated dual substitution (IDS) algorithm. This method showed strength in solving the MMR-MKP, and it seems that this strategy is suitable to exploit the structure of the problem: solutions with good objective values are distributed in a diverse area in the feasible region (i.e., they are not close together), which was observed through computational experiments, and intensification approaches, often effective for classical combinatorial optimization problems, will not work for this type of problems. Another contribution is that we proposed a method to generate a linear constraint at every iteration without solving any NP-hard subproblems. We also evaluated several classical techniques that have been proposed for the MMR-GAP, such as a fixed-scenario heuristic, a branch-and-cut approach, and a logic-based Benders decomposition. We observed that the IDS performed best on all of the tested instances, and it exactly solved 7 out of 18 instances to optimality. Furthermore, IDS always found its best solutions at an early iteration.

Finally, we considered a crew pairing problem to minimize the total person-days under some some basic and specific constraints. We modeled it as a set covering problem and applied a standard column generation framework. In the warm start process, we designed a two-step method to generate the initial column set. The first step is a depth first search for solving the problem of finding a longest path in a graph under some resource constraints. It then invokes a dynamic programming (DP) algorithm to cover all the uncovered flight legs. For solving the subproblem, we proposed a DP-based branch-and-bound method, which outperformed a branch-and-bound algorithm without bounding operations by DP. Moreover, when solving pricing problems, we accelerate this algorithm by considering regularity, exploiting the fact that most flight legs are regularly scheduled. The computational experiments showed that the proposed algorithm had a good overall performance for most of the tested instances and obtained good solutions even for the instances with up to 10,000 flights.

In summary, we have demonstrated that row generation and column generation approaches are two important practical techniques in solving large-scale optimization problems. We hope that the techniques discussed in this thesis can be helpful for solving large-scale combinatorial optimization problems.

Bibliography

[1] H. Aissi, C. Bazgan, and D. Vanderpooten. Complexity of the min-max and min-max regret assignment problem. *Operations Research Letters*, 33:634–640, 2005.

[2] H. Aissi, C. Bazgan, and D. Vanderpooten. Minmax and minmax regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, 197:427–438, 2009.

[3] C. Alves and J. Carvalho. A stabilized branch-and-price-and-cut algorithm for the multiple length cutting stock problem. *Computers & Operations Research*, 35:1315–1328, 2008.

[4] R. Anbil, R. Tanga, and E. Johnson. A global approach to crew pairing optimization. *IBM Systems Journal*, 31:71–78, 1992.

[5] C. Archetti, N. Bianchessi, and M. Speranza. A column generation approach for the split delivery vehicle routing problem. *Networks*, 58:241–254, 2011.

[6] T. Assavapokee, J. Realff, and J. Ammons. Min-max regret robust optimization approach on interval data uncertainty. *Journal of Optimization Theory and Applications*, 137:297–316, 2008.

[7] I. Averbakh. Minmax regret linear resource allocation problems. *Operations Research Letters*, 32:174–180, 2004.

[8] I. Averbakh. The minmax regret permutation flow-shop problem with two jobs. *European Journal of Operational Research*, 169:761–766, 2006.

[9] I. Averbakh and V. Lebedev. Interval data minmax regret network optimization problems. *Discrete Applied Mathematics*, 138:289–301, 2004.

[10] E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1996.

[11] C. Barnhart, A. Cohn, E. Johnson, D. Klabjan, G. Nemhauser, and P. Vance. Airline crew scheduling. In R. Hall, editor, *Handbook of Transportation Science*, chapter 14, pages 517–560. Springer, 2 edition, 2003.

[12] C. Barnhart, C. Jame, and P. Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity network flow problems. *Operations Research*, 48:318–326, 2000.

[13] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1998.

[14] J. Barutt and T. Hull. Airline crew scheduling: Supercomputers and algorithms. *SIAM News*, 23:20–22, 1990.

[15] J. Beli'́en and E. Demeulemeester. A branch-and-price approach for integrating nurse and surgery scheduling. *European Journal of Operational Research*, 189:652–668, 2008.

[16] W. Ben-Ameur and J. Neto. A constraint generation algorithm for large scale linear programs using multiple-points separation. *Mathematical Programming*, 107:517–357, 2006.

[17] J. Benders. Partitioning procedures for solving mixed variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[18] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, Cambridge, 2004.

[19] A. Candia-Véjar, E. Álvarez-Miranda, and N. Maculan. Min-max regret combinatorial optimization problems: An algorithmic perspective. *RAIRO - Operations Research*, 45:101–129, 2011.

[20] S. Canto. Application of benders' decomposition to power plant preventive maintenance scheduling. *European Journal of Operational Research*, 184:759–777, 2008.

[21] S. Chen and Y. Shen. An improved column generation algorithm for crew scheduling problems. *Journal of Information & Computational Science*, 10:175–183, 2013.

[22] P. Chu and J. Beasle. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristic*, 4:63–86, 1998.

[23] P. Chu and J. Beasley. A genetic algorithm for the generalized assignment problem. *Computers & Operations Research*, 24:17–23, 1997.

[24] E. Conde and A. Candia. Minimax regret spanning arborescences under uncertain costs. *European Journal of Operational Research*, 182:561–577, 2007.

[25] J. Cordeau, G. Stojkovic, F. Soumis, and J. Desrosiers. Benders decomposition for simultaneous aircraft routing and crew scheduling. *Transportation Science*, 35:375–388, 2001.

[26] F. Croce, C. Koulamas, and V. T'kindt. A constraint generation approach for the two-machine flow shop problem with jobs selection. In P. Fouilhoux, E. Gouveia, A. Mahjoub, and V. Paschos, editors, *Combinatorial Optimization: Third International Symposium, ISCO 2014, Lisbon, Portugal, March 5-7, 2014, Revised Selected Papers*, pages 198–207, Berlin, 2014. Springer.

[27] G. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[28] V. Deineko and G. Woeginger. Pinpointing the complexity of the interval min-max regret knapsack problem. *Discrete Optimization*, 7:191–196, 2010.

[29] M. Dell'Amico, G. Righini, and M. Salani. A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection. *Transportation Science*, 40:235–247, 2006.

[30] G. Desaulniers, J. Desrosiers, and M. Solomon. *Column Generation.* Springer Science & Business Media, New York, 2006.

[31] M. Desrochers and F. Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23:1–13, 1989.

[32] J. Desrosiers, F. Soumis, and M. Desrochers. Routing with time windows by column generation. *Networks*, 14:545–565, 1984.

[33] G. Easwaran and H. Üster. Tabu search and benders decomposition approaches for a capacitated closed-loop supply chain network design problem. *Transportation Science*, 43:301–320, 2009.

[34] Y. Eldar, A. Ben-Tal, and A. Nemirovski. Linear minimax regret estimation of deterministic parameters with bounded data uncertainties. *IEEE Transactions on Signal Processing*, 52:2177–2187, 2004.

[35] T. Emden-Weinert and M. Proksch. Best practice simulated annealing for the airline crew scheduling problem. *Journal of Heuristics*, 5:419–436, 1999.

[36] H. Fei, C. Chu, and N. Meskens. Solving a tactical operating room planning problem by a column-generation-based heuristic procedure with four criteria. *Annals of Operations Research*, 166:91–108, 2009.

[37] M. Feizollahi and I. Averbakh. The robust (minmax regret) quadratic assignment problem with interval flows. *INFORMS Journal on Computing*, 26:321–335, 2014.

[38] M. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11:109–124, 1981.

[39] L. Ford and D. Fulkerson. suggested computation for maximal multicommodity network flows. *Management Science*, 5:97–101, 1958.

[40] F. Furini, M. Iori, S. Martello, and M. Yagiura. Heuristic and exact algorithms for the interval min-max regret knapsack problem. *INFORMS Journal on Computing*, 27:392–405, 2015.

[41] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman, San Francisco, 1979.

[42] A. Geoffrion. Generalized benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.

[43] P. Gilmore and R. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.

[44] P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem–part ii. *Operations Research*, 11:863–888, 1963.

[45] R. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[46] R. Gomory. Early integer programming. In J. Lenstra, A. Rinnooy Kan, and A. Schrijver, editors, *History of Mathematical Programming: A Collection of Personal Reminiscences*, pages 55–61. North-Holland, 1991.

[47] B. Gopalakrishnan and E. Johnson. Airline crew scheduling: State-of-the-art. *Annals of Operations Research*, 140:305–337, 2005.

[48] G. Gutiérrez-Jarpa, G. Desaulniers, and G. Laporte. A branch-and-price algorithm for the vehicle routing problem with deliveries, selective pickups and time windows. *European Journal of Operational Research*, 206:341–349, 2010.

[49] K. Hoffman and M. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39:657–682, 1993.

[50] J. Hooker. Planning and scheduling by logic-based benders decomposition. *Operations Research*, 55:588–602, 2007.

[51] J. Hooker and G. Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 1:33–60, 1996.

[52] V. Jain and I. Grossmann. Algorithms for hybrid milp/cp models for a class of optimization problems. *INFORMS Journal on Computing*, 13:258–276, 2001.

[53] O. Karasan, C. Pinar, M, and H. Yaman. The robust shortest path problem with interval data. Technical report, Bilkent University, Ankara, Turkey, 2001. available from the web site of Optimization Online (http://www.optimization-online.org/, accesed on May 3, 2016).

[54] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[55] A. Kasperski. *Discrete Optimization with Interval Data.* Springer, Berlin, 2008.

[56] A. Kasperski and P. Zielinski. An approximation algorithm for interval data minmax regret combinatorial optimization problems. *Information Processing Letters*, 97:177–180, 2006.

[57] J. Kelley. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8:703–712, 1960.

[58] A. Klose and S. Görtz. A branch-and-price algorithm for the capacitated facility location problem. *European Journal of Operational Research*, 179:1109–1125, 2007.

[59] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms.* Springer, Berlin, 2012.

[60] P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications.* Kluwer Academic Publishers, Dordrecht, 1997.

[61] M. Laguna, J. Kelly, J. Gonzalez-Velarde, and F. Glover. Tabu search for the multilevel generalized assignment problem. *European Journal of Operational Research*, 82:176–189, 1995.

[62] L. Lasdon. *Optimization theory for large systems.* Dover Publications, New York, 1970.

[63] D. Levine. Application of a hybrid genetic algorithm to airline crew scheduling. *Computers & Operations Research*, 23:547–558, 1996.

[64] D. Li, H. Huang, A. Morton, and E. Chew. Simultaneous fleet assignment and cargo routing using benders decomposition. *OR Spectrum*, 28:319–335, 2006.

[65] H. Li and K. Womer. Scheduling projects with multi-skilled personnel by a hybrid milp/cp benders decomposition algorithm. *Journal of Scheduling*, 12:281–298, 2009.

[66] F. Liberatore, G. Righini, and M. Salani. A column generation algorithm for the vehicle routing problem with soft time windows. *4OR*, 9:49–82, 2011.

[67] M. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53:1007–1023, 2005.

[68] B. Maenhout and M. Vanhoucke. Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. *Journal of Scheduling*, 13:77–79, 2010.

[69] O. Marcotte. An instance of the cutting stock problem for which the rounding property does not hold. *Operations Research Letters*, 4:239–243., 1986.

[70] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley & Sons, Chichester, New York, 1990.

[71] A. Merciera, J. Cordeaub, and F. Soumisa. A computational study of benders decomposition for the integrated aircraft routing and crew scheduling problem. *Computers & Operations Research*, 32:1451–1476, 2005.

[72] R. Montemanni. A benders decomposition approach for the robust spanning tree problem with interval data. *European Journal of Operational Research*, 174:1479–1490, 2006.

[73] R. Montemanni, J. Barta, M. Mastrolilli, and L. Gambardella. The robust traveling salesman problem with interval data. *Transportation Science*, 41:366–381, 2011.

[74] R. Montemanni and L. Gambardella. A branch and bound algorithm for the robust spanning tree with interval data. *European Journal of Operational Research*, 161:771–779, 2005.

[75] R. Montemanni and L. Gambardella. The robust shortest path problem with interval data via benders decomposition. *4OR*, 3:315–328, 2005.

[76] G. Nemhauser. Column generation for linear and integer programming. *Documenta Mathematica*, Extra Volume: Optimization Stories:65–73, 2012.

[77] K. Onodera and A. Mori. Cockpit crew scheduling and supporting system. *Proceedings of the World Congress on Expert Systems*, 1:1–10, 1991.

[78] H. Osman and K. Demirli. A bilinear goal programming model and a modified benders decomposition algorithm for supply chain reconfiguration and supplier selection. *International Journal of Production Economics*, 1:97–105, 2010.

[79] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33:60–100, 1991.

[80] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Dover Publications, New York, 1982.

[81] K. Park, S. Kang, and S. Park. An integer programming approach to the bandwidth packing problem. *Management Science*, 42:1277–1291, 1996.

[82] R. Parker and R. Rardin. *Discrete Optimization.* Academic Press, New York, 1988.

[83] J. Pereira and I. Averbakh. Exact and heuristic algorithms for the interval data robust assignment problem. *Computers & Operations Research*, 38:1153–1163, 2011.

[84] J. Pereira and I. Averbakh. The robust set covering problem with interval data. *Annals of Operations Research*, 207:217–235, 2013.

[85] M. Posta, J. Ferland, and P. Michelon. An exact method with variable fixing for solving the generalized assignment problem. *Computational Optimization and Applications*, 52:629–644, 2012.

[86] E. PrescottGagnon, G. Desaulniers, and L. Rousseau. A branchandpricebased large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, 54:190–204, 2009.

[87] R. Rahmaniani, T. Crainic, M. Gendreau, and W. Rei. The benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 2016. (in press).

[88] G. Raidl, T. Baumhauer, and B. Hu. Speeding up logic-based benders' decomposition by a metaheuristic for a bi-level capacitated vehicle routing problem. In M. Blesa, C. Blum, and Voß., editors, *Hybrid Metaheuristics*, pages 183–197. Springer, 2014.

[89] K. Ruland. A model for aeromedical routing and scheduling. *International Transactions in Operational Research*, 6:57–73, 1999.

[90] R. Sadykov and F. Vanderbeck. Bin packing with conflicts: A generic branch-and-price algorithm. *INFORMS Journal on Computing*, 25:244–255, 2012.

[91] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the Association for Computing Machinery*, 23:555–565, 1976.

[92] M. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Structural and Multidisciplinary Optimization*, 44:259–275, 2001.

[93] G. Scheithauer and J. Terno. The modified integer round-up property of the one-dimensional cutting stock problem. *European Journal of Operational Research*, 84:562–571, 1995.

[94] G. Scheithauer and J. Terno. Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem. *Operations Research Letters*, 20:93–100, 1997.

[95] A. Sethi and G. Thompson. The noncandidate constraint method. In M. Karwan, V. Lofti, S. Zionts, and J. Telgen, editors, *Redundancy in Mathematical Programming*, chapter 11, pages 135–144. Springer, 1983.

[96] A. Sethi and G. Thompson. The pivot and probe algorithm for solving a linear program. *Mathematical Programming*, 29:219–233, 1984.

[97] S. Siddiqui, S. Azarm, and S. Gabriel. A modified benders decomposition method for efficient robust optimization under interval uncertainty. *Operations Research*, 45:831–841, 1997.

[98] H. Üster, G. Easwaran, E. Akçali, and S. Çetinkaya. Benders decomposition with alternative multiple cuts for a multi-product closed-loop supply chain network design model. *Naval Research Logistics*, 54:890–907, 2007.

[99] M. Varnamkhasti. Overview of the algorithms for solving the multidimensional knapsack problem. *Advanced Studies in Biology*, 4:37–47, 2012.

[100] D. Wheatley, F. Gzara, and E. Jewkes. Logic-based benders decomposition for an inventory-location problem with service constraints. *Omega*, 55:10–23, 2015.

[101] W. Wilhelm. A technical review of column generation in integer programming. *Optimization and Engineering*, 2:159–200, 2001.

[102] H. Williams. *Model Building in Mathematical Programming*. Wiley, New York, 1985.

[103] L. Williams, J. Fisher, and A. Willsky. A constraint generation integer programming approach to information theoretic sensor resource management. In *IEEE/SP 14th Workshop On Statistical Signal Processing*, pages 59–63, 8 2007.

[104] M. Yagiura and T. Ibaraki. Generalized assignment problem. In T. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, chapter 48. Chapman & Hall/CRC in the Computer & Information Science Series, 2 edition, 2007.

[105] M. Yagiura, T. Ibaraki, and F. Glover. An ejection chain approach for the generalized assignment problem. *INFORMS Journal on Computing*, 16:133–151, 2004.

[106] M. Yagiura, T. Ibaraki, and F. Glover. A path relinking approach with ejection chains for the generalized assignment problem. *European Journal of Operational Research*, 169:548–569, 2006.

[107] M. Yagiura, M. Kishida, and T. Ibaraki. A 3-flip neighborhood local search for the set covering problem. *European Journal of Operational Research*, 172:472–499, 2006.

[108] H. Yaman, O. Karasan, and M. Pinar. The robust spanning tree problem with interval data. *Operations Research Letters*, 40:29–31, 2001.

[109] F. You and I. Grossmann. Multicut benders decomposition algorithm for process supply chain planning under uncertainty. *Annals of Operations Research*, 210:191–211, 2015.

[110] G. Yu. Min-max optimization of several classical discrete optimization problems. *Journal of Optimization Theory and Applications*, 98:221–242, 1998.