Université de Montréal

Methods for Solving Combinatorial Pricing Problems

par

Quang Minh Bui

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique

2023-12-15

Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

Methods for Solving Combinatorial Pricing Problems

présentée par

Quang Minh Bui

a été évaluée par un jury composé des personnes suivantes :

Emma Frejinger

(présidente-rapporteuse)

Margarida Carvalho

(directrice de recherche)

Jean-Yves Potvin

(membre du jury)

Luce Brotcorne

(examinatrice externe)

Emma Frejinger

(représentante du doyen de la FESP)

Résumé

Le *problème de tarification combinatoire* (CPP) ou le *jeu de tarification de Stackelberg* est une classe de problèmes d'optimisation bi-niveaux comprenant deux décideurs dans un ordre séquentiel. Le premier décideur, le leader, maximise ses revenus en contrôlant les prix d'un ensemble de ressources. Le deuxième décideur, le suiveur, réagit aux prix et sélectionne un sous-ensemble de ressources selon un problème d'optimisation combinatoire. Selon le problème du suiveur, le CPP peut être très difficile à résoudre. Cette thèse présente trois articles couvrant plusieurs méthodes de solution exacte pour le CPP. Le premier article aborde la modélisation et le prétraitement pour une spécialisation du CPP : le *problème de tarification du réseau* (NPP), dans lequel le problème du suiveur est un problème du plus court chemin. Les formulations du NPP sont organisées dans un cadre général qui établit les liens entre elles. Le deuxième article se concentre sur la version à plusieurs marchandises du NPP. À partir des résultats de l'analyse convexe, nous dérivons une nouvelle formulation du NPP et prouvons que le NPP évolue de manière polynomiale par rapport au nombre de marchandises, étant donné que le nombre d'arcs à péage est fixe. Le troisième article nous ramène au CPP général, dans lequel les problèmes du suiveur sont NP-difficiles. En utilisant deux modèles de programmation dynamique différents, les problèmes du suiveur sont convertis en programmes linéaires, auxquels la dualité forte peut être appliquée. En raison de la nature NP-difficile de ces problèmes, des schémas de génération dynamique de contraintes sont proposés. Les méthodes de solution décrites dans chaque article sont étayées par des résultats expérimentaux, montrant leur efficacité en pratique. Cette thèse approfondit notre compréhension de la structure du CPP et introduit des méthodologies innovantes pour y faire face, contribuant ainsi à de nouvelles perspectives pour aborder les problèmes de tarification et bi-niveau en général

Mots-clés : Problème de tarification combinatoire, Jeu de tarification de Stackelberg, Problème de tarification du réseau, Analyse convexe, Programmation dynamique.

Abstract

The *combinatorial pricing problem* (CPP) or *Stackelberg pricing game* is a class of bilevel optimization problems that consist of two decision makers in sequential order. The first decision maker, the leader, maximizes their revenue by controlling the prices of a set of resources. The second decision maker, the follower, reacts to the prices and selects a subset of resources according to a combinatorial optimization problem. Depending on the follower's problem, the CPP can be very challenging to solve. This thesis presents three articles covering several exact solution methods for the CPP. The first article addresses the modeling and preprocessing for a specialization of the CPP: the *network pricing problem* (NPP), in which the follower's problem is a shortest path problem. The formulations of the NPP are organized in a general framework which establishes the links between them. The second article focuses on the multi-commodity version of the NPP. From the results in convex analysis, we derive a novel formulation of the NPP and with it, we prove that the NPP scales polynomially with respect to the number of commodities, given that the number of tolled arcs is fixed. The third article leads us back to the general CPP, in which the follower's problems are NP-hard. By utilizing two different dynamic programming models, the follower's problems are converted into linear programs, to which strong duality can be applied. Due to the NP-hard nature of these problems, dynamic constraint generation schemes are proposed. The solution methods described in each article are backed up with experimental results, showing that they are effective in practice. This thesis deepens our comprehension of the CPP structure and introduces innovative methodologies for addressing it, thereby contributing new perspectives to tackle pricing and bilevel problems in general.

Keywords: Combinatorial pricing problem, Stackelberg pricing game, Network pricing problem, Convex analysis, Dynamic programming.

Contents

List of Tables

List of Figures

List of Notations and Abbreviations

CPP	Combinatorial Pricing Problem
DD	Decision Diagram
DP	Dynamic Programming
ILP	Integer Linear Program
KKT	Karush-Kuhn-Tucker
KPP	Knapsack Pricing Problem
LP	Linear Program
MaxSSPP	Maximum Stable Set Pricing Problem
MILP	Mixed-Integer Linear Program
MinSCPP	Minimum Set Cover Pricing Problem

NPP	Network Pricing Problem
SD	Selection Diagram
SPGM	Shortest-Path Graph Model
VF	Value Function

Acknowledgements

I would like to express my deepest gratitude to my former supervisor, Bernard Gendron, whose guidance was invaluable during my formative years in research. His wisdom and passion for the field continue to inspire me.

I am also immensely grateful to my current supervisor, Margarida Carvalho, whose kindness and expertise have shaped my research and academic journey. Her mentorship has been a source of constant encouragement and learning.

I would like to extend my thanks to José Neto for his collaboration and dedication throughout this project. His insights have significantly enriched our work.

Lastly, I would like to thank my family and friends for their support and understanding.

This work would not have been possible without the support and contributions of these individuals, and for that, I am truly grateful.

Introduction

Pricing is an important task for any successful business. It is one of the four pillars in marketing: Product, Place, Promotion, and Price, which were popularized by Edmund Jerome McCarthy in his 1960 book named *Basic Marketing: A Managerial Approach* [45]. In the field of operations research, there exist numerous models for pricing, for example the revenue management model [5], the dynamic posted-price model [25], and the discrete choice model [32]. In this thesis, we are interested in a bilevel model of pricing, in which the seller sets fixed prices on a set of items first, then the buyer decides deterministically which items to purchase by solving a combinatorial optimization problem. According to the terminology in bilevel optimization, we refer to the seller and the buyer as the leader and the follower respectively. The objective of the leader is to attract the follower by setting reasonable prices. If the prices are too low, the leader gains little profit. But if the prices are too high, the follower will not choose the items and the leader will lose customers. This class of pricing problems is called *combinatorial pricing problems* (CPP) or *Stackelberg pricing games*. We present three articles focusing on the solution methods of the CPP.

We start the thesis with an investigation on the *network pricing problem* (NPP), which is a specialization of the CPP where the follower makes their decision by solving a shortest path problem. For a realistic scenario, the leader in the NPP is a highway authority who controls several tolled roads, and the follower is a traveller who wants to pick the most inexpensive route. In the first article (Chapter 2), we describe a framework to unify all existing single-level formulations of the NPP. We decompose a formulation of the NPP into four components: *primal representation*, *dual representation*, *optimality condition*, and *linearization method*. Each component has several varieties, which can be mixed and matched to create a wide array of possible formulations. In the latter half of the article, we propose a new path enumeration scheme and a path-based preprocessing method that can be applied to all formulations, which is shown experimentally to be effective compared to an existing alternative.

The second article (Chapter 3) explores the multi-commodity version of the NPP, in which there are more than one traveller (called commodity) in the network, each has its own origin and destination nodes. We prove a theorem stating that the NPP scales differently with respect to the number of commodities and the number of tolled arcs. In simpler terms, the difficulty of solving the NPP increases far more dramatically with the number of tolled arcs than with the number of travellers. The proof uses novel tools such as *conjugate model* and *reaction plot*. The mechanism behind this asymmetry is due to *strong bilevel feasibility* which is only satisfied by a limited number of combinations of paths. With strong bilevel feasibility, we derive a multi-commodity preprocessing method. This is in contrast to most preprocessing methods for the NPP in the literature (including the one in Chapter 2) that are applied to each commodity separately. Numerical results show that it can reduce the solution time significantly on small networks with a high number of commodities.

Finally, we return to the general CPP in the third article (Chapter 4), where the follower solves some combinatorial problem other than the shortest path problem. For the general case, the literature is sparse on exact solution methods. To obtain a formulation, we utilize the dynamic programming model of the follower's problem to convert it into a linear program. Then we leverage strong duality to rewrite the CPP as a single-level reformulation similar to the method described in Chapter 2. However, if the given follower's problem is NP-hard, the resulting linear program will be exponential in size (unless P = NP). Thus, we also draft an algorithm to generate its constraints dynamically. Two types of dynamic programming models are studied: *selection diagram* and *decision diagram*, as well as a simpler model called the *value function formulation*. Their performances are tested and suggest that the formulations derived from the two diagrams outperform the value function formulation in most cases.

Chapter 1

Background

In this chapter, we outline several key concepts and results that are used later in the three articles. Linear programming strong duality (Section 1.1) is the main method to convert a bilevel programming formulation of the CPP to a single-level reformulation. To take advantage of strong duality, Section 1.2 shows several ways to rewrite an integer program into an equivalent linear program. An overview on convex analysis is provided in Section 1.3. This is the primary tool of the second article, used to deepen our understanding between the leader's decision and followers' reactions. Section 1.4 describes bilevel optimization in general. Finally, Section 1.5 introduces the main topic of this thesis: *combinatorial pricing problems.*

The reader who is interested in further details on linear programming, integer programming and convex analysis is referred to the text books of Luenberger [44], Conforti et al. [19], and Rockafellar [52], respectively.

1.1. Linear Programming

A *linear program* (LP) is an optimization problem aiming to find a vector $x \in \mathbb{R}^n$ that minimizes a linear functional $c^\top x$ and satisfies $Ax \geq b$, $x \geq 0$ where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$. An LP is usually written in the form:

$$\min_x \; c^\top x \tag{1.1a}$$

$$\text{s.t. } Ax \geq b, \tag{1.1b}$$

$$x \geq 0. \tag{1.1c}$$

LPs are usually solved using the simplex algorithm [21] or the interior-point method [36]. The latter is guaranteed to solve an LP with a polynomial number of variables and constraints in polynomial time. Thus, if a problem can be formulated as an LP, it is considered an "easy" problem.

A central result in linear programming is *strong duality* [44], which states that if Program (1.1) has an optimal solution, then it has the same objective value as its *dual*:

$$\max_{y} b^\top y \tag{1.2a}$$

$$\text{s.t. } A^\top y \leq c, \tag{1.2b}$$

$$y \geq 0. \tag{1.2c}$$

In particular, given a pair of points x^* and y^* that are feasible to Programs (1.1) and (1.2) respectively, x^* and y^* are optimal to their respective programs if and only if $c^\top x^* = b^\top y^*$. Consequently, we can rewrite Program (1.1) as the following constraint satisfaction problem:

$$Ax \geq b, \tag{1.3a}$$

$$A^\top y \leq c, \tag{1.3b}$$

$$c^\top x = b^\top y, \tag{1.3c}$$

$$x, y \geq 0. \tag{1.3d}$$

An equivalent condition to strong duality is *complementary slackness*, which states that $y^\top(Ax - b) = 0$ and $x^\top(c - A^\top y) = 0$. These constraints can be used to replace Constraint (1.3c) and thus, rewrite Program (1.1) as a *linear complementarity problem* [20]. The resulting set of constraints is usually referred to as the Karush–Kuhn–Tucker (KKT) conditions [38].

1.2. Integer Programming

An *integer linear program* (ILP) is an extension of Program (1.1) where x is restricted to be integral:

$$\min_{x} c^\top x \tag{1.4a}$$

$$\text{s.t. } Ax \geq b, \tag{1.4b}$$

$$x \geq 0, \tag{1.4c}$$

$$x \in \mathbb{Z}^n. \tag{1.4d}$$

If some elements of x can be continuous, *i.e.* $x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}$, then the program is a *mixed-integer linear program* (MILP). Many NP-hard problems such as the knapsack problem, the Boolean satisfiability problem, and the travelling saleman problem can be modelled as ILPs or MILPs [29]. As a result, general integer programs are NP-hard. Despite the theoretical intractability of ILPs (and MILPs), sophisticated branch-and-cut algorithms have emerged, implemented within powerful solvers like Gurobi, CPLEX, SCIP and Cbc. These solvers significantly enhance our ability to effectively solve MILPs in practice. Nevertheless, challenges persist within this class of problems.

In this thesis, we are interested in a conversion of Program (1.4) into a set of constraints similar to Program (1.3). Unfortunately, in general, there is no well-defined dual for an ILP. However, if we convert an ILP into an LP, then we can leverage the linear programming duality theory. In particular, if the matrix A in Constraint (1.4b) satisfies a property called *total unimodularity* and if b is an integral vector, then Program (1.4) can be converted directly into an LP by dropping Constraint (1.4d) [19]. A matrix is totally unimodular if and only if every square submatrix has determinant 0, 1, or -1. A well-known class of totally unimodular matrices is the class of incidence matrices of directed graphs. Given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, define the incidence matrix $A \in \mathbb{R}^{\mathcal{V}\times\mathcal{A}}$ such that

$$A_{u,a} = \begin{cases} -1 & \text{if } u \in \mathcal{V} \text{ is the source of } a \in \mathcal{A}, \\ +1 & \text{if } u \in \mathcal{V} \text{ is the target of } a \in \mathcal{A}, \\ 0 & \text{otherwise.} \end{cases}$$

Then, the matrix A is totally unimodular. This technique is commonly used to convert the shortest path problem to an LP. Some problems such as the knapsack problem possess a *dynamic programming* model. A dynamic programming model with a finite number of states and actions is equivalent to a shortest (or longest) path problem in a directed acyclic graph. Thus, we can write such problems as shortest path problems, use total unimodularity to drop the integrality constraint, and take the dual of the resulting LP to compose a set of constraints as in Program (1.3).

In reality, most matrices A are not totally unimodular. In that case, we consult Meyer's theorem [48], which states that the convex hull of $S = \{x \in \mathbb{Z}^n \mid Ax \geq b,\ x \geq 0\}$ is a polyhedron (assuming that A and b are rational). Let $P = \text{conv}(S)$ be this convex hull. Optimizing a linear function over S is equivalent to optimizing the same function over P. Thus, Program (1.4) is equivalent to $\min\{c^\top x \mid x \in P\}$. The polyhedron P has two representations:

— The intersection of half spaces, *i.e.* $P = \{x \in \mathbb{R}^n \mid Cx \geq d\}$ where each row in $Cx \geq d$ defines a facet of P;

— The combination of extreme points and extreme rays, *i.e.* $P = \text{conv}\{v^1, \dots, v^p\} + \text{cone}\{r^1, \dots, r^q\}$ where v^i are the extreme points and r^j are the extreme rays of P.

Using the first representation, Program (1.4) is equivalent to the LP $\min\{c^\top x \mid Cx \geq d\}$. For the second representation, we can rewrite Program (1.4) as:

$$\min \sum_{i=1}^{p} (c^\top v^i)\lambda_i + \sum_{j=1}^{q} (c^\top r^j)\mu_j \tag{1.5a}$$

$$\text{s.t.} \sum_{i=1}^{p} \lambda_i = 1, \tag{1.5b}$$

$$\lambda,\, \mu \geq 0. \tag{1.5c}$$

In both cases, we can derive a dual of Program (1.4) by taking the dual of the corresponding LPs. Note that if the original ILP is NP-hard, then both representations are exponential in size because their LPs cannot be solved in polynomial time (assuming P $\neq$ NP).

As a final remark, one can also take advantage of strong duality by converting ILPs to other classes of convex programs. For example, ILPs (or more generally, mixed-binary quadratic programs) can be reformulated as a *completely positive program* which has a dual that is a *copositive program* [15]. This approach, however, is not explored in this work.

1.3. Convex Conjugate

A function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is convex and lower semicontinuous if and only if its epigraph $\text{epi}(f) = \{(z, x) \in \mathbb{R} \times \mathbb{R}^n \mid z \geq f(x)\}$ is a closed convex set, which can be represented as an intersection of closed half spaces. In other words, a convex lower semicontinuous function f is the pointwise supremum of all hyperplanes (affine functions) h such that $h \leq f$. If two such hyperplanes have the same slope (*i.e.* they are parallel), one will dominate the other. Thus, for each slope $y \in \mathbb{R}^n$, only the highest hyperplane h_y is required to describe f. This hyperplane must be a supporting hyperplane, hence $h_y(x) = y^\top x - g(y)$ where $g(y)$ is the highest value such that $y^\top x - g(y) = f(x)$ for some $x \in \mathbb{R}^n$. In particular, we have:

$$g(y) = \sup_x \{y^\top x - f(x)\}. \tag{1.6}$$

Since f is the pointwise supremum of h_y, $y \in \mathbb{R}^n$, we have:

$$f(x) = \sup_y \{y^\top x - g(y)\}. \tag{1.7}$$

The functions f and g are two different representations of the same convex function. We call g the *convex conjugate* (or *Legendre transform*) of f, denoted as f^* [52]. From Equations (1.6) and (1.7), f is also the convex conjugate of $f^* = g$. This is the content of the Fenchel–Moreau theorem, which states that $(f^*)^* = f$ if and only if f is convex and lower semicontinuous (the other direction is implied by the fact that the convex conjugate is always convex and lower semicontinuous).

It follows from Equation (1.6) that $f(x)+f^*(y) \geq y^\top x$, which is called the Fenchel-Young inequality. If $f(x) + f^*(y) = y^\top x$ for some x and y, then h_y supports f at x. We call y the *subgradient* of f at x. The set of all subgradients of f at x is the *subdifferential* of f at x, denoted as $\partial f(x)$. It holds that $y \in \partial f(x)$ if and only if $x \in \partial f^*(y)$. The subdifferential $\partial f(x)$ is convex for all $x \in \mathbb{R}^n$.

Operations on one representation have dual operations in the other representation. For example, given a constant $\lambda \in \mathbb{R} \setminus \{0\}$, the convex conjugate of $f(\lambda x)$ is $f^*(y/\lambda)$. Table 1.1 lists several common operations and their duals. The dual of the last operation (sum of functions) is the *infimal convolution*, defined as:

$$(g_1 \,\square \cdots \square\, g_m)(y) = \inf\{g_1(y_1) + \cdots + g_m(y_m) \mid y_1 + \cdots + y_m = y\}.$$

Table 1.1 – Common operations and their convex conjugates.

Functions	Convex conjugates	Conditions
$f(\lambda x)$	$f^*(y/\lambda)$	$\lambda \in \mathbb{R} \setminus \{0\}$
$\lambda f(x)$	$\lambda f^*(y/\lambda)$	$\lambda \in \mathbb{R}$, $\lambda > 0$
$f(x+a)$	$f^*(y) - a^\top y$	$a \in \mathbb{R}^n$
$f(x) + \alpha$	$f^*(y) - \alpha$	$\alpha \in \mathbb{R}$
$f_1 + \cdots + f_m$	$f_1^* \,\square \cdots \square\, f_m^*$	

In the second article (Chapter 3), the representations described in this section will enable us to explore a novel representation for the standard bilevel program for the NPP.

1.4. Bilevel Optimization

A *bilevel optimization* problem is a problem with nested structure involving two parties: a leader and a follower. Both parties have their own sets of variables and by controlling these variables, they seek to optimize their respective objective functions. However, this is not a simultaneous game: the leader always decides first, and the follower decides last, knowing what the leader's decision was. In return, the leader can assume that the follower will react rationally, *i.e.* they always choose the optimal solution. This class of games is also called *Stackelberg games* [53].

In this thesis, we consider *the optimistic assumption.* Under this assumption, given a specific leader's decision, if the follower has multiple optimal solutions, we assume that the follower chooses the solution that benefits the leader the most. Denote $x \in \mathcal{X}$ the leader's decision variable, and $y \in \mathcal{Y}$ the follower's decision variable. An optimistic bilevel optimization problem has the form:

$$\begin{aligned}\min_{x,y}\ & F(x,y)\\ \text{s.t.}\ & G(x,y) \geq 0,\\ & x \in \mathcal{X},\\ & y \in \mathbf{R}(x),\end{aligned}$$

where $\mathbf{R}(x)$ is the set of optimal solutions of the *follower's problem* given the leader's decision x:

$$\begin{aligned}\mathbf{R}(x) = \operatorname*{argmin}_{y}\ & f(x,y)\\ \text{s.t.}\ & g(x,y) \geq 0,\\ & y \in \mathcal{Y}.\end{aligned}$$

Bilevel optimization problems are very challenging to solve. Indeed, linear bilevel problems have been shown to be NP-hard [35]. Solution techniques for various classes of bilevel optimization problems can be found in Kleinert et al. [37]. The most popular class in the literature is the class of *mixed-integer bilevel linear programs* (MIBLP) which have the form:

$$\min_{x,y}\ c^\top x + d^\top y \tag{1.10a}$$

$$\text{s.t. } Ax + By \geq a, \tag{1.10b}$$

$$x \in \mathcal{X}, \tag{1.10c}$$

$$y \in \operatorname{argmin}\{e^\top y \mid Cx + Dy \geq b,\ y \in \mathcal{Y}\}, \tag{1.10d}$$

with integrality constraints imposed within $\mathcal{X}$ and $\mathcal{Y}$. A common assumption is that the linking variables (the entries of x that appear in the follower's problem) must be discrete and bounded. By allowing integrality constraints, MIBLPs become Σ_2^p-hard [42], including commonly studied min-max special cases [17]. In practice, this means that no MILP reformulation of polynomial size is expected to exist for MIBLPs. Still, general-purpose solution methods exist, namely, the extension of the branch-and-cut method by Moore and Bard [49]. This method first considers the *high-point relaxation* of Program (1.10), which is a single-level relaxation by omitting the optimality condition of the follower's problem:

$$\begin{aligned}\min_{x,y}\ & c^\top x + d^\top y \\ \text{s.t. } & Ax + By \geq a, \\ & Cx + Dy \geq b, \\ & x \in \mathcal{X}, \\ & y \in \mathcal{Y}.\end{aligned}$$

Then, it gradually cuts off solutions that are not in $\mathbf{R}(x)$ (which are called *bilevel infeasible*), with the no-good cut [22] and the bilevel intersection cut [26] as examples. By accommodating the above cuts with improved pruning and branching, Tahernejad et al. [54] published a solver named `MibS` for MIBLPs. Recently, there has been some progress in solving general mixed-integer bilevel non-linear problems [47].

1.5. Combinatorial Pricing Problem

The central topic of this thesis revolves around a class of bilevel optimization problems called *combinatorial pricing problems* (CPP). This class is also referred to as *Stackelberg pricing games* in the literature. In a CPP, there is a set of *items* $\mathcal{I}$ which is partitioned into the set of *tolled items* $\mathcal{I}_1$ and the set of *toll-free items* $\mathcal{I}_2$. The leader can control the prices of the tolled items by setting the *toll* $t_i \geq 0$ for $i \in \mathcal{I}_1$. Thus, the price of a tolled item $i \in \mathcal{I}_1$ is $c_i + t_i$ while the price of a toll-free item $i \in \mathcal{I}_2$ is c_i where $c \in \mathbb{R}^{\mathcal{I}}$ is the vector of the *base costs* of the items. Given the tolls set by the leader, the follower selects a subset of items in $\mathcal{I}$ that minimizes the total cost according to some combinatorial problem. Denote the follower's decision as $x \in \{0,1\}^{\mathcal{I}}$. The objective of the leader is to maximize the revenue collected on the tolled items, *i.e.* the sum of $t_i x_i$ for all $i \in \mathcal{I}_1$. Mathematically, the CPP can be formulated as:

$$\max_{t,x}\ t^\top x \tag{1.12a}$$

$$\text{s.t. } t \in \mathcal{T}, \tag{1.12b}$$

$$x \in \operatorname{argmin}\{(c+t)^\top x \mid x \in \mathcal{X}\} \tag{1.12c}$$

where $\mathcal{T} = \mathbb{R}_+^{\mathcal{I}_1} \times \{0\}^{\mathcal{I}_2}$ is the set of feasible tolls and $\mathcal{X}$ is the feasible set of the follower's problem.

The only coupling mechanism between the leader and the follower is the bilinear term $t^\top x$, which one side wants to maximize while the other side wants to minimize. This observation may suggest a connection between the CPP and the class of interdiction problems [17, 22]. In an interdiction problem, the objective functions of the leader and the follower are the same, but they are in opposite directions. In the CPP however, thanks to the base cost c, the relation between the leader and the follower in the CPP is not totally adversarial like in an interdiction problem. This relation is closer to the one between a service provider (as the leader) and its customers (as the follower). As long as the service is not obligatory, the customers will only use the service if it is better than the alternatives. Thus, the leader actually seeks to improve of the follower's objective while making a profit in the process. In fact, an upper bound of Program (1.12) is the difference of the follower's objective value with $t \to \infty$ (*i.e.* no tolled items are available) and the follower's objective value with $t = 0$ (*i.e.* all tolled items are available without markups).

There are several differences between the MIBLP (Program (1.10)) and the CPP (Program (1.12)):

— The leader variable t in the CPP is continuous;

— The objective functions of both the leader and the follower in the CPP contain the bilinear term $t^\top x$;

— The leader's decision does not affect the feasible set of the follower's problem $\mathcal{X}$.

Due to the above dissimilarities, the CPP is not compatible with general solvers for MIBLPs such as `MibS` [54].

The most common specialization of the CPP is the *network pricing problem* (NPP), in which the items are arcs in a directed graph and the follower's problem is the shortest path problem. The single- and multi-commodity versions of NPP were introduced in Labbé et al. [40] and Brotcorne et al. [10], respectively. The NPP has been proven to be NP-hard [51]. The preferred method to solve the NPP is to convert Program (1.12) to a single-level reformulation using the KKT conditions (as described in Section 1.1), either with the original shortest path LP [10], or with a path-based representation [6, 24]. Preprocessing techniques such as joining multiple toll-free arcs into one [55] and removing redundant paths [6] can drastically reduce the size of the NPP and improve the solution time of the final program.

Besides the NPP, other specializations of the CPP have been studied, in which the follower's problem is the minimum spanning tree problem [18], the knapsack problem [7, 50], the stable set problem [16], and the bipartite vertex cover problem [8]. In general, the CPP is Σ_2^p-hard [16]. However, existing literature mostly focuses on the computational complexity and approximation schemes for some specific cases of the CPP rather than the solution method in the general case.

The concepts described in this section will be put into practice in the subsequent three articles. Specifically, single-level reformulation using KKT conditions on integer programs is the method of choice in the first and the third articles, while convex conjugate provides a theoretical basis for the second work.

,

Chapter 2 First article

A Catalog of Formulations for the Network Pricing Problem

by

Quang Minh Bui[1], Bernard Gendron[1], and Margarida Carvalho[1]

([1]) CIRRELT and Département d'informatique et de recherche opérationnelle
Université de Montréal

This article has been published in INFORMS Journal on Computing [14].

My contributions and the roles of the coauthors:

— Bernard Gendron suggested the general research direction on the network pricing problem.

— The research ideas were developed jointly by all three authors.

— The experiments were designed, implemented, and run by me.

— The article (including all proofs and algorithms) was written by me and it was revised by Bernard and Margarida.

Abstract. We study the network pricing problem where the leader maximizes their revenue by determining the optimal amounts of tolls to charge on a set of arcs, under the assumption that the followers will react rationally and choose the shortest paths to travel. Many distinct single-level reformulations of this bilevel optimization program have been proposed, however, their relationship has not been established. In this paper, we aim to build a connection between those reformulations and explore the combination of the path representation with various modeling options, allowing us to generate 12 different reformulations of the problem. Moreover, we propose a new path enumeration scheme, path-based preprocessing, and hybrid framework to further improve performance and robustness when solving the final model. We provide numerical results, comparing all the derived reformulations and confirming the efficiency of the novel dimensionality reduction procedures.

Keywords: Networks, Pricing, Bilevel programming, Multi-commodity transportation

2.1. Introduction

The network pricing problem (NPP) is a bilevel optimization program involving two parties: a leader and multiple followers. First, the leader sets the prices of several arcs (called tolled arcs) in a network. Afterward, the followers choose the optimal paths in the network subject to the prices set by the leader. The leader's objective is to maximize their profit, while the goal of the followers is to minimize their own costs of transit by following the respective shortest paths across the network. High prices do not always mean more profit for the leader if only a few followers can afford them. Thus, the leader must seek a balance between the prices and the decisions of the followers reacting to those prices. In practice, the leader could be a highway authority and the followers could be groups of the population residing in different neighborhoods or cities.

2.1.1. Related Literature

The NPP was first introduced in Labbé et al. [40] as a single-commodity problem (one follower), and later extended to the multi-commodity version (multiple followers) by Brotcorne et al. [10]. The problem is proven to be NP-hard even when there is only one follower [51]. The first mixed integer linear program (MILP) formulation was also introduced in Brotcorne et al. [10]. Since then, many improvements to their MILP were proposed such as the shortest-path graph model (SPGM) [55], preprocessing techniques [6], path-based models [6, 24], valid inequalities, and tight bounds for big-M parameters [23]. Other methods were also explored, including multipath enumeration [12] and tabu search [13]. Variants of the network pricing problem include the joint network design and pricing problem [11], the complete toll NPP (clique pricing problem) [34], and the logit NPP [30].

The NPP has a natural bilevel optimization formulation. There are general-purpose methods to solve bilevel optimization problems, *e.g.* , Fischetti et al. [26], Tahernejad et al.

[54]. However, due to their complexity, they are usually considered per case to exploit specific problem structures. A common approach to solve a bilevel optimization problem is to convert it to a single-level reformulation. For the NPP, the follower problems are linear. Thus, there are three methods generally used for this conversion based on optimality conditions: strong duality, complementary slackness, and value function. In the first two methods, the primal and the dual constraints of the follower problems are included in the single-level reformulation and then accompanied by either strong duality or complementary slackness constraints. In the third method, only the primal constraints are needed and the optimality of the follower problems is ensured by the value function constraints.

Strong duality was used in the original formulation proposed in Brotcorne et al. [10]. The discovery of the path representation led to the use of complementary slackness in the path-based formulations developed in Didi-Biha et al. [24]. Value function formulations were mentioned in Heilporn et al. [33] and Bouhtou et al. [6]. However, these models were presented as separate formulations with little connection in between.

2.1.2. Contribution and Paper Organization

In the first part of this work, our aim is to explore the links between all of the previously mentioned single-level reformulations of the NPP. First, we show that in strong duality and complementary slackness reformulations, the path representation can be applied not only to the primal follower problems, but also to the dual problems. This allows us to show that the value function method is a special case of strong duality when the dual problems are written in the path representation. Second, we provide a systematic way to formalize a reformulation by breaking it down into components with different options for each component. This general method can also be used to categorize all the models mentioned above and explain the connection between them.

The second part of this work is dedicated to the path enumeration process. This is important because a well-performed path-based reformulation requires an efficient path enumeration algorithm. Existing methods must enumerate either all the paths [6] or a set of relevant paths but require multiple calls to a linear solver [24]. Based on previous discoveries on the properties of redundant paths in Bouhtou et al. [6] and Didi-Biha et al. [24], we develop a new path enumeration process that can enumerate relevant paths without employing a linear solver. Besides that, based on this set of relevant paths, we also derive a new preprocessing method that can be applied to arc-based reformulations. This preprocessing method is crucial for a fair comparison between the reformulations, since path enumeration is also counted as a preprocessing method for the path-based reformulations.

Regardless of the efficiency of the path enumeration process, the potential size of the set of all relevant paths is exponential, which may require an enormous amount of time to enumerate them. In the last part of this work, we introduce a method to circumvent this problem by taking advantage of the multi-commodity nature of the NPP. By deciding which commodities are worth applying path enumeration and mixing different reformulations into one, we can save time by spending less time enumerating very large sets of paths and more time solving the actual optimization model. In this paper, we refer to this solution as the hybrid framework for multi-commodity problems.

This paper is structured as follows. In Section 2.2, we describe the NPP, namely, its bilevel optimization program, and explain the general method to formalize a reformulation. Section 2.3 shows how path enumeration is performed in the novel preprocessing method. The hybrid framework is introduced in Section 2.4. Section 2.5 presents computational results, including a comparison of all formulations and several experiments to prove the efficiency of the new preprocessing method and the hybrid framework.

2.2. The Network Pricing Problem

In this section, we provide the bilevel programming formulation for the network pricing problem in Section 2.2.1 and a systematic way to generate single-level reformulations through the combination of different components in Section 2.2.2. The conversion from the bilevel formulation to single-level reformulations produces bilinear terms. The linearization of these bilinear terms is the topic of discussion in Section 2.2.3.

2.2.1. Problem Formulation

Let us consider a graph $G = (V, A)$ where A is partitioned into a set of tolled arcs A_1 and a set of toll-free arcs A_2. Each arc $a \in A$ has an initial cost $c_a > 0$. Let K be the set of commodities (O-D pairs) and o^k, d^k, η^k be the origin, the destination, and the demand of commodity $k \in K$, respectively. We define the following variables:

— $T_a, a \in A_1$: the toll of arc a;

— $x_a^k, a \in A_1, k \in K$: the flow of commodity k on tolled arc a;

— $y_a^k, a \in A_2, k \in K$: the flow of commodity k on toll-free arc a.

The network pricing problem is then formulated as a bilevel program:

$$
\text{(NPP)} \quad \max_{T\geq 0, x, y} \sum_{k\in K}\sum_{a\in A_1} \eta^k T_a x_a^k
$$

$$
\text{s.t. } \forall k \in K \left\{
\begin{array}{lr}
(x^k, y^k) \in \arg\min\limits_{\hat{x},\hat{y}} \sum\limits_{a\in A_1}(c_a+T_a)\hat{x}_a + \sum\limits_{a\in A_2} c_a\hat{y}_a & \\
\quad \text{s.t.} \sum\limits_{a\in A_1^+(i)}\hat{x}_a + \sum\limits_{a\in A_2^+(i)}\hat{y}_a - \sum\limits_{a\in A_1^-(i)}\hat{x}_a - \sum\limits_{a\in A_2^-(i)}\hat{y}_a = b_i^k, & i\in V, \\
\quad \hat{x}_a \in \{0,1\}, & a\in A_1, \\
\quad \hat{y}_a \in \{0,1\}, & a\in A_2,
\end{array}
\right.
$$

where $b_i^k = 1$ if $i = o^k$, -1 if $i = d^k$, and 0 otherwise. For a node i, the set of its outgoing tolled (toll-free) arcs is $A_1^+(i)$ ($A_2^+(i)$) and the set of its incoming tolled (toll-free) arcs is $A_1^-(i)$ ($A_2^-(i)$). The leader controls the toll prices T_a, while each follower $k \in K$ decides the taken paths by setting x_a^k and y_a^k equal to 1 if the arc belongs to the path, and 0 otherwise. We consider the optimistic version of this bilevel problem, *i.e.* , if a follower has multiple optimal solutions with respect to the prices set by the leader, then they will choose the solution that benefits the leader the most. For each commodity, we assume that there exists at least a toll-free path (a path without tolled arcs). Otherwise, the leader could gain infinite revenue by exploiting that particular O-D pair.

2.2.2. Generalized Single-Level Reformulations

A generalized single-level reformulation of the NPP consists of three components: *(i)* a primal representation (arc or path), *(ii)* a dual representation (arc or path), and *(iii)* an optimality condition (strong duality or complementary slackness). In this way, the followers' optimization problems can be equivalently replaced by constraints in the NPP.

First, the primal and the dual representations of the follower problems need to be chosen. There are two options for each: the arc representation and the path representation. Consider the follower problem for commodity k:

$$
\begin{array}{llr}
(\text{PA}^k) & \min\limits_{x^k,y^k} \sum\limits_{a\in A_1}(c_a+T_a)x_a^k + \sum\limits_{a\in A_2} c_a y_a^k & \\
& \text{s.t.} \sum\limits_{a\in A_1^+(i)} x_a^k + \sum\limits_{a\in A_2^+(i)} y_a^k - \sum\limits_{a\in A_1^-(i)} x_a^k - \sum\limits_{a\in A_2^-(i)} y_a^k = b_i^k, \quad i\in V, & (2.1) \\
& x_a^k \in \{0,1\}, \quad a\in A_1, & \\
& y_a^k \in \{0,1\}, \quad a\in A_2. &
\end{array}
$$

This is called the *primal-arc representation* of the follower problem. Because the set of constraints of the follower problem forms a totally unimodular matrix, integrality conditions for x_a^k and y_a^k can be dropped (see Wolsey and Nemhauser [56]). This enables us to write the follower problem in the *dual-arc representation*:

$$
\begin{aligned}
(\mathrm{DA}^k) \qquad & \max_{\lambda^k} \ \lambda^k_{o^k} - \lambda^k_{d^k} & & \\
& \text{s.t. } \lambda^k_i - \lambda^k_j \le c_a + T_a, & a \equiv (i,j) \in A_1, & \qquad (2.2) \\
& \qquad \lambda^k_i - \lambda^k_j \le c_a, & a \equiv (i,j) \in A_2. & \qquad (2.3)
\end{aligned}
$$

In Bouhtou et al. [6] and Didi-Biha et al. [24], instead of writing the follower problem using arc-flow x_a^k, y_a^k, the authors replaced these variables with z_p^k representing the path-flow or the selection of paths p in the set P^k consisting of all elementary paths, *i.e.* paths without cycles, which is a finite set. We will refer this as the *primal-path representation*:

$$
\begin{aligned}
(\mathrm{PP}^k) \qquad & \min_{z^k} \sum_{p \in P^k} \left(\sum_{a \in A} \delta^p_a c_a + \sum_{a \in A_1} \delta^p_a T_a \right) z^k_p & & \\
& \text{s.t. } \sum_{p \in P^k} z^k_p = 1, & & \qquad (2.4) \\
& \qquad z^k_p \in \{0,1\}, & p \in P^k. &
\end{aligned}
$$

In the above formulation, $\delta^p_a = 1$ if the path p includes the arc a and $\delta^p_a = 0$ if not. Once again, the constraints form a totally unimodular matrix, hence z^k_p is not required to be binary and thus the follower problem also has a *dual-path representation*:

$$
\begin{aligned}
(\mathrm{DP}^k) \qquad & \max_{L^k} \ L^k & & \\
& \text{s.t. } L^k \le \sum_{a \in A} \delta^p_a c_a + \sum_{a \in A_1} \delta^p_a T_a, & p \in P^k. & \qquad (2.5)
\end{aligned}
$$

In total, there are four combinations of primal-dual representations (arc-arc, arc-path, path-arc, and path-path). Note that the primal and the dual representations are not required to match each other, so primal-arc could also be paired with dual-path.

After deciding the representations, we need to connect the primal and the dual representations by either strong duality or complementary slackness constraints. Strong duality simply connects the primal and the dual objective functions together. Below are the strong duality constraints for the arc-arc, arc-path, path-arc, and path-path combinations, respectively:

$$\sum_{a\in A_1}(c_a+T_a)x_a^k+\sum_{a\in A_2}c_a y_a^k=\lambda_{o^k}^k-\lambda_{d^k}^k, \tag{2.6}$$

$$\sum_{a\in A_1}(c_a+T_a)x_a^k+\sum_{a\in A_2}c_a y_a^k=L^k, \tag{2.7}$$

$$\sum_{p\in P^k}\left(\sum_{a\in A}\delta_a^p c_a+\sum_{a\in A_1}\delta_a^p T_a\right)z_p^k=\lambda_{o^k}^k-\lambda_{d^k}^k, \tag{2.8}$$

$$\sum_{p\in P^k}\left(\sum_{a\in A}\delta_a^p c_a+\sum_{a\in A_1}\delta_a^p T_a\right)z_p^k=L^k. \tag{2.9}$$

Complementary slackness matches bounded variables in the primal with inequality constraints in the dual. The pairing is simple if the primal and the dual have the same representation (either arc-arc or path-path):

$$(c_a+T_a-\lambda_i^k+\lambda_j^k)x_a^k=0, \qquad a\equiv(i,j)\in A_1, \tag{2.10}$$

$$(c_a-\lambda_i^k+\lambda_j^k)y_a^k=0, \qquad a\equiv(i,j)\in A_2, \tag{2.11}$$

$$\left(\sum_{a\in A}\delta_a^p c_a+\sum_{a\in A_1}\delta_a^p T_a-L^k\right)z_p^k=0, \qquad p\in P^k. \tag{2.12}$$

However, if the primal and the dual have different representations, variables in the primal representation must be converted to their dual counterparts. For the path-arc combination, conversion from z to (x,y) is done by using the identities $x_a^k=\sum_{p\in P^k}\delta_a^p z_p^k$ for $a\in A_1$ and $y_a^k=\sum_{p\in P^k}\delta_a^p z_p^k$ for $a\in A_2$:

$$(c_a+T_a-\lambda_i^k+\lambda_j^k)\left(\sum_{p\in P^k}\delta_a^p z_p^k\right)=0, \qquad a\equiv(i,j)\in A_1, \tag{2.13}$$

$$(c_a-\lambda_i^k+\lambda_j^k)\left(\sum_{p\in P^k}\delta_a^p z_p^k\right)=0, \qquad a\equiv(i,j)\in A_2. \tag{2.14}$$

For the arc-path combination, the conversion from (x,y) to z is more complicated. We will use the definition that $z_p^k=1$ if and only if $x_a^k=1$ and $y_a^k=1$ for all arcs a belonging to the path p (given that (x,y) produce an elementary path). Mathematically, $z_p^k=\prod_{a\in A_1|\delta_a^p=1}x_a^k\prod_{a\in A_2|\delta_a^p=1}y_a^k$. The complementary slackness constraint for the arc-path combination is:

$$\left(\sum_{a\in A}\delta_a^p c_a+\sum_{a\in A_1}\delta_a^p T_a-L^k\right)\prod_{\substack{a\in A_1\\ \delta_a^p=1}}x_a^k\prod_{\substack{a\in A_2\\ \delta_a^p=1}}y_a^k=0, \qquad p\in P^k. \tag{2.15}$$

Table 2.1 – Complete map of all single-level reformulations.

(a) Strong duality

Dual \ Primal	Arc	Path
Arc	Standard (STD)	Path-Arc Standard (PASTD)
Path	Value Function (VF)	Path Value Function (PVF)

(b) Complementary slackness

Dual \ Primal	Arc	Path
Arc	Complementary Slackness (CS)	Path-Arc Complementary Slackness (PACS)
Path	Value Function Complementary Slackness (VFCS)	Path Complementary Slackness (PCS)

An important rule of the conversion is that it must cover all possible solutions in the primal representation. Otherwise, the optimality condition can be dodged by choosing the primal solution that is not covered. If we leave the graph and P^k as inputted, this rule is satisfied. However, in Section 2.3, we will introduce methods that reduce the size of the graph and P^k, thus requiring additional attention. We will discuss this in Section 2.3.2.

For completeness, accordingly with the choice of the primal representation, here are the objective function of the single-level formulations:

$$\sum_{k\in K}\sum_{a\in A_1}\eta^k T_a x_a^k, \tag{2.16}$$

$$\sum_{k\in K}\sum_{a\in A_1}\sum_{p\in P^k}\eta^k \delta_a^p T_a z_p^k. \tag{2.17}$$

Based on the primal, the dual, and the optimality condition, we can categorize all the single-level formulations as in Table 2.1. The variables, constraints and objective functions of these formulations are listed in Table 2.2. The variables are bounded implicitly (x, y, z, T are non-negative; λ, L are unrestricted).

Table 2.2 – List of variables, constraints, and objective functions of all single-level reformulations.

			Constraints		
Label	Variables	Objective	Primal	Dual	Opt. Cond.
Strong duality					
STD	T, x, y, λ	(2.16)	(2.1)	(2.2), (2.3)	(2.6)
VF	T, x, y, L	(2.16)	(2.1)	(2.5)	(2.7)
PASTD	T, z, λ	(2.17)	(2.4)	(2.2), (2.3)	(2.8)
PVF	T, z, L	(2.17)	(2.4)	(2.5)	(2.9)
Complementary slackness					
CS	T, x, y, λ	(2.16)	(2.1)	(2.2), (2.3)	(2.10), (2.11)
VFCS	T, x, y, L	(2.16)	(2.1)	(2.5)	(2.15)
PACS	T, z, λ	(2.17)	(2.4)	(2.2), (2.3)	(2.13), (2.14)
PCS	T, z, L	(2.17)	(2.4)	(2.5)	(2.12)

There are some notable reformulations in the list. The most canonical way to convert a bilevel linear program to its single-level version is the standard formulation (STD). See Appendix 2.A.1 for the explicit formulation. This formulation has been referred to in most papers in the literature [33, 24, 6, 23], including the first paper on the NPP [10]. It only uses arc representations, so the need for path enumeration is eliminated. Its linearized version (Section 2.2.3) only requires x to be integer. Furthermore, the number of arcs is fixed, hence, it is reliable compared to the path-based formulations whose sizes depend on the number of paths of each commodity which is unknown. Overall, the standard formulation is the most straightforward method to solve the NPP.

The second reformulation in which we are interested is the value function formulation (VF). The explicit formulation is shown in Appendix 2.A.2. If we combine the constraints (2.5) and (2.7), a new constraint emerges:

$$\sum_{a \in A_1} (c_a + T_a) x_a^k + \sum_{a \in A_2} c_a y_a^k \leq \sum_{a \in A} \delta_a^p c_a + \sum_{a \in A_1} \delta_a^p T_a \qquad p \in P^k. \tag{2.18}$$

The left hand side of Eq. (2.18) is the cost of the current path, while the right hand side is the cost of path p. Eq. (2.18) means that the cost of the current path must not surpass the cost of any path in P^k, which restricts (x, y) to choose the path with least cost. This is called the value function constraint and it is also a popular method to generate the single-level formulation of a bilevel problem. Here, we have shown that the value function method is just a special case of strong duality when we write the dual representation of the follower problems in the solution space (dual-path in the case of the NPP). The value function formulation for the NPP was previously mentioned in Heilporn et al. [33].

The last formulation in which we focus is the path complementary slackness formulation (PCS). See Appendix 2.A.3 for the formulation. This formulation is the path-based model of Didi-Biha et al. [24]. It is the complete opposite of the standard formulation. It only uses path representations, hence its performance heavily relies on the number of paths in P^k. Methods for reducing the size of P^k will be discussed in Section 2.3. In this formulation, only the selection of p is important and any information on the structure of the graph is discarded. The advantage of this formulation is its simplicity, especially when the size of P^k is small.

2.2.3. Linearization

In all eight single-level formulations, there are bilinear terms (or in the case of VFCS, multilinear terms). In order to solve these formulations using MILP solvers, these terms need to be linearized. We will consider formulations using strong duality and those using complementary slackness separately.

2.2.3.1. Linearization of strong duality formulations. In the strong duality formulations, the bilinear terms arise from the leader's revenue, which are either $\sum_{a\in A_1} T_a x_a^k$ or $\sum_{a\in A_1}\sum_{p\in P^k}\delta_a^p T_a z_p^k$. Because x_a^k and z_p^k can only take 0 and 1 as their values, we can force them to be binary and add new variables $t_a^k = T_a x_a^k = T_a \sum_{p\in P^k}\delta_a^p z_p^k$, for $a \in A_1$, with the following constraints:

$$0 \le t_a^k \le M_a^k x_a^k, \qquad a \in A_1, \tag{2.19}$$

$$0 \le T_a - t_a^k \le N_a(1 - x_a^k), \qquad a \in A_1, \tag{2.20}$$

for the primal-arc representation and

$$0 \le t_a^k \le M_a^k \sum_{p\in P^k} \delta_a^p z_p^k, \qquad a \in A_1, \tag{2.21}$$

$$0 \le T_a - t_a^k \le N_a \left(1 - \sum_{p\in P^k} \delta_a^p z_p^k\right), \qquad a \in A_1, \tag{2.22}$$

for the primal-path representation. M_a^k and N_a are big-M parameters. We refer this method of linearization as *direct linearization.* Tight bounds for M_a^k and N_a can be found in Dewez et al. [23]. The leader revenue for each commodity becomes $\sum_{a\in A_1}\eta^k t_a^k$, which we can use to replace the bilinear terms. The strong duality constraints of the four linearized formulations are as follows:

$$\sum_{a\in A_1}(c_a x_a^k + t_a^k) + \sum_{a\in A_2} c_a y_a^k = \lambda_{o^k}^k - \lambda_{d^k}^k, \tag{2.23}$$

$$\sum_{a\in A_1}(c_a x_a^k + t_a^k) + \sum_{a\in A_2} c_a y_a^k = L^k, \tag{2.24}$$

$$\sum_{p\in P^k}\sum_{a\in A}\delta_a^p c_a z_p^k + \sum_{a\in A_1} t_a^k = \lambda_{o^k}^k - \lambda_{d^k}^k, \tag{2.25}$$

$$\sum_{p\in P^k}\sum_{a\in A}\delta_a^p c_a z_p^k + \sum_{a\in A_1} t_a^k = L^k. \tag{2.26}$$

The objective function of the linearized formulation is independent from the primal representation:

$$\sum_{k\in K}\sum_{a\in A_1}\eta^k t_a^k. \tag{2.27}$$

A summary of all linearized strong duality formulations is provided in Table 2.3.

Table 2.3 – List of variables, constraints, and objective functions of all linearized strong duality formulations.

			Constraints			
Label	Variables	Obj.	Primal	Dual	Opt. Cond.	Linearization
STD	T, t, x^*, y, λ	(2.27)	(2.1)	(2.2), (2.3)	(2.23)	(2.19), (2.20)
VF	T, t, x^*, y, L	(2.27)	(2.1)	(2.5)	(2.24)	(2.19), (2.20)
PASTD	T, t, z^*, λ	(2.27)	(2.4)	(2.2), (2.3)	(2.25)	(2.21), (2.22)
PVF	T, t, z^*, L	(2.27)	(2.4)	(2.5)	(2.26)	(2.21), (2.22)

The variables with stars are required to be binary.

2.2.3.2. Linearization of complementary slackness formulations. In complementary slackness formulations, bilinear terms appear in all complementary slackness constraints. In the arc-arc (CS) and path-path (PCS) combinations, each complementary slackness constraint is gated by a primal variable (x, y, or z), which can only take binary values. Thus, we could use them as branching condition for the constraints. Here are the linearized constraints of the arc-arc combination (CS):

$$\lambda_i^k - \lambda_j^k \geq c_a + T_a - R_a^k(1 - x_a^k), \qquad a \equiv (i,j) \in A_1, \tag{2.28}$$

$$\lambda_i^k - \lambda_j^k \geq c_a - R_a^k(1 - y_a^k), \qquad a \equiv (i,j) \in A_2, \tag{2.29}$$

and of the path-path combination (PCS):

$$L^k \geq \sum_{a \in A} \delta_a^p c_a + \sum_{a \in A_1} \delta_a^p T_a - S_p^k(1 - z_p^k), \qquad p \in P^k, \tag{2.30}$$

where R_a^k and S_p^k are big-M parameters depending on the dual representation (R_a^k for dual-arc and S_p^k for dual-path). When x, y or z are equal to 1, the big-M terms are dropped. Combining with the dual constraints, these complementary slackness constraints force them to be active. If the primal variables are equal to 0, the big-M terms make the constraints redundant, essentially removing them from the formulation. To calculate the values of these big-M parameters, recalling that N_a is the upper bound of T_a defined in (2.20), denote:

— $\underline{\lambda}_i^k$: the minimum cost from i to d^k when $T_a = 0$ for all $a \in A_1$;
— $\overline{\lambda}_j^k$: the minimum cost from j to d^k when $T_a = N_a$ for all $a \in A_1$;
— $\underline{L}^k$: the minimum cost from o^k to d^k when $T_a = 0$ for all $a \in A_1$.

Then, the upper bounds for the big-M parameters are:

$$\begin{aligned} R_a^k &= c_a + N_a - \underline{\lambda}_i^k + \overline{\lambda}_j^k, && a \equiv (i,j) \in A_1, \\ R_a^k &= c_a - \underline{\lambda}_i^k + \overline{\lambda}_j^k, && a \equiv (i,j) \in A_2, \\ S_p^k &= \sum_{a \in A} \delta_a^p c_a + \sum_{a \in A_1} \delta_a^p N_a - \underline{L}^k, && p \in P^k. \end{aligned}$$

In the calculation of $\overline{\lambda}_j^k$, excluding all the tolled arcs (set $T = \infty$) is a valid option, nevertheless, it may disconnect j and d^k which may lead to infinite R_a^K. Thus, since we aim for the smallest R_a^k, we do not remove them. A similar technique can be used for the path-arc combination (PACS):

$$\lambda_i^k - \lambda_j^k \geq c_a + T_a - R_a^k \left(1 - \sum_{p \in P^k} \delta_a^p z_p^k \right), \qquad a \equiv (i,j) \in A_1, \tag{2.31}$$

$$\lambda_i^k - \lambda_j^k \geq c_a - R_a^k \left(1 - \sum_{p \in P^k} \delta_a^p z_p^k \right), \qquad a \equiv (i,j) \in A_2. \tag{2.32}$$

For the arc-path combination (VFCS), the constraints have multilinear terms. We will use the expression

$$\sum_{a \in A} \delta_a^p - \sum_{a \in A_1} \delta_a^p x_a^k - \sum_{a \in A_2} \delta_a^p y_a^k \tag{2.33}$$

as the branching condition. The number of arcs in path p is $\sum_{a \in A} \delta_a^p$. Expression (2.33) is only equal to 0 if all the arcs along the path p are active, *i.e.* p is chosen. The linearized constraints of the arc-path combination (VFCS) are:

$$L^k \geq \sum_{a\in A} \delta_a^p c_a + \sum_{a\in A_1} \delta_a^p T_a - S_p^k \left(\sum_{a\in A} \delta_a^p - \sum_{a\in A_1} \delta_a^p x_a^k - \sum_{a\in A_2} \delta_a^p y_a^k \right), \qquad p \in P^k. \tag{2.34}$$

After linearizing all the bilinear terms in the complementary slackness constraints, there are still bilinear terms in the objective function (for the leader's revenue). We could use the direct linearization method as in the strong duality case which adds more big-M constraints to the formulations. However, we can take advantage of the special structure of the NPP: the leader's revenue appears twice, once in the objective function, and once in the strong duality constraint. Furthermore, since strong duality constraints are not utilized in complementary slackness formulations, the leader's revenue can be extracted from these constraints and then be substituted in the objective function. We will refer to this method as *linearization by substitution* [24]. Let $\tau^k = \sum_{a\in A_1} T_a x_a^k = \sum_{a\in A_1} \sum_{p\in P^k} \delta_a^p T_a z_p^k$ be the leader's revenue for commodity k per unit of demand. The strong duality constraints are used to extract τ^k:

$$\sum_{a\in A_1} c_a x_a^k + \sum_{a\in A_2} c_a y_a^k + \tau^k = \lambda_{o^k}^k - \lambda_{d^k}^k, \tag{2.35}$$

$$\sum_{a\in A_1} c_a x_a^k + \sum_{a\in A_2} c_a y_a^k + \tau^k = L^k, \tag{2.36}$$

$$\sum_{p\in P^k} \sum_{a\in A} \delta_a^p c_a z_p^k + \tau^k = \lambda_{o^k}^k - \lambda_{d^k}^k, \tag{2.37}$$

$$\sum_{p\in P^k} \sum_{a\in A} \delta_a^p c_a z_p^k + \tau^k = L^k. \tag{2.38}$$

The objective function becomes:

$$\sum_{k\in K} \eta^k \tau^k. \tag{2.39}$$

Table 2.4 is the summary of all linearized complementary slackness formulations. Since there are two linearization methods for the bilinear terms in the objective function, a suffix is added to the label, where 1 means direct linearization and 2 means linearization by substitution. Overall, the linearization of the complementary slackness formulations is more complicated than that of the strong duality formulations. Complementary slackness formulations usually have more binary variables and more constraints, which may lead to worse performance.

Combining with the four strong duality formulations, in total, there are 12 different MILP formulations for the NPP. Some of them are completely new: (PASTD), (CS), and (VFCS), while others are already mentioned in other works. However, the performance of the new formulations is generally worse, as we will show later in the computational experiments. Table 2.5 shows how these reformulations fit in the big picture. In the left and the middle

Table 2.4 – List of variables, constraints, and objective functions of all linearized complementary slackness formulations.

			Constraints			
Label	Variables	Obj.	Primal	Dual	Opt. Cond.	Linearization
Direct linearization						
CS1	T, t, x^*, y^*, λ	(2.27)	(2.1)	(2.2), (2.3)	(2.28), (2.29)	(2.19), (2.20)
VFCS1	T, t, x^*, y^*, L	(2.27)	(2.1)	(2.5)	(2.34)	(2.19), (2.20)
PACS1	T, t, z^*, λ	(2.27)	(2.4)	(2.2), (2.3)	(2.31), (2.32)	(2.21), (2.22)
PCS1	T, t, z^*, L	(2.27)	(2.4)	(2.5)	(2.30)	(2.21), (2.22)
Linearization by substitution						
CS2	$T, \tau, x^*, y^*, \lambda$	(2.39)	(2.1)	(2.2), (2.3)	(2.28), (2.29)	(2.35)
VFCS2	T, τ, x^*, y^*, L	(2.39)	(2.1)	(2.5)	(2.34)	(2.36)
PACS2	T, τ, z^*, λ	(2.39)	(2.4)	(2.2), (2.3)	(2.31), (2.32)	(2.37)
PCS2	T, τ, z^*, L	(2.39)	(2.4)	(2.5)	(2.30)	(2.38)

The variables with stars are required to be binary.

columns, the names and the labels used in the original works are listed. The corresponding labels used in this paper are shown in the right column. We remark that in Bouhtou et al. [6], the path-based formulation (PMIP) is a path value function formulation (PVF) with a minor modification: the bilinear terms are linearized separately per path by replacing $t_a^k = T_a x_a^k$ with $r_{pa}^k = T_a x_a^k z_p^k$.

2.3. Path Enumeration and Preprocessing

This section describes a new path enumeration process in detail, combining the main ingredients previously explored in the literature. Then, we discuss the use of this process as a preprocessing method for arc-based formulations. The latter will enable us to fairly compare arc and path-based reformulations, since we make preprocessing available for both.

2.3.1. Path Enumeration

Formulations using path representations require the set P^k, and thus, the explicit enumeration of all paths from o^k to d^k. The size of P^k can be exponential. However, not all paths are relevant. We can eliminate many of them by using the dominance rule in Bouhtou et al. [6].

Definition 2.1. *Given a commodity k, a path is bilevel feasible if it is optimal in the follower problem for some value of T.*

Table 2.5 – The mapping of previous formulations to the general catalog.

Name in original work	Original label	General label
Brotcorne et al. [10]		
Mixed integer formulation	CPLEX	STD
Heilporn et al. [33]		
Mixed integer formulation	TP2	STD
New formulation	TP3	VF
Didi-Biha et al. [24]		
Arc-based formulation	MIP I	STD
Arc-path formulation	MIP II	PACS1
Path-based formulation	MIP III	PCS2
Bouhtou et al. [6]		
Arc-based formulation	AMIP	STD
Path-based formulation	PMIP	PVF*
Dewez et al. [23]		
Arc-based formulation	TOP-ARCS	STD
Path-based formulation	PATH	PCS2

Mathematically, if p is bilevel feasible, then there exists T such that for all $q \in P^k$:

$$\sum_{a\in A} \delta_a^p c_a + \sum_{a\in A_1} \delta_a^p T_a \leq \sum_{a\in A} \delta_a^q c_a + \sum_{a\in A_1} \delta_a^q T_a.$$

Lemma 2.2 (Bouhtou et al. [6]). *Consider any commodity $k \in K$. Let p and q be two different paths in P^k. If for all $a \in A_1$, $\delta_a^p \leq \delta_a^q$ and*

$$\sum_{a\in A} \delta_a^p c_a < \sum_{a\in A} \delta_a^q c_a,$$

then q is not bilevel feasible, i.e., q cannot be the optimal path for any value of T.

Proof. Suppose that q is the optimal path for some fixed value of T. The follower's cost of q must be minimal: $\sum_{a\in A} \delta_a^p c_a + \sum_{a\in A_1} \delta_a^p T_a \geq \sum_{a\in A} \delta_a^q c_a + \sum_{a\in A_1} \delta_a^q T_a$. However, because $\delta_a^p \leq \delta_a^q$, this means $\sum_{a\in A_1} \delta_a^p T_a \leq \sum_{a\in A_1} \delta_a^q T_a$; and because $\sum_{a\in A} \delta_a^p c_a < \sum_{a\in A} \delta_a^q c_a$, this leads to a contradiction. $\square$

The dominance rule in Lemma 2.2 is the only necessary rule to eliminate all non-bilevel-feasible paths. Any remaining path is bilevel feasible.

Theorem 2.3. *Any path which is not eliminated by the dominance rule in Lemma 2.2 is bilevel feasible.*

Proof. We will prove that if path p is not eliminated, then it is optimal for the following value of T:

$$T_a = \begin{cases} 0 & \text{if } \delta_a^p = 1, \\ \infty & \text{otherwise.} \end{cases}$$

Suppose that p is not optimal for the above value of T and let q be the optimal path. Consequently, q can only use arcs with $\delta_a^p = 1$ as all other tolled arcs are disabled. In other words, $\delta_a^q \leq \delta_a^p$. Also because $T_a = 0$ for $\delta_a^p = 1$, the costs of p and q in this case are exactly their initial costs, hence $\sum_{a\in A} \delta_a^q c_a < \sum_{a\in A} \delta_a^p c_a$. By Lemma 2.2, p should have been eliminated which is a contradiction. Thus, p must be optimal for the given value of T. □

Definition 2.1 alone does not ensure that any bilevel feasible path is relevant to solve the NPP. Suppose we have two different paths with the same initial cost and the same set of tolled arcs. Then they can be both bilevel feasible. In such case, however, only one path is required. Thus, we employ Assumption 2.4 to make the term "bilevel feasible" coincide with the term "relevant".

Assumption 2.4. *Two different bilevel feasible paths must have different initial costs.*

This assumption also implies that two different bilevel feasible paths will have different sets of tolled arcs. If two different paths have the same set of tolled arcs but different initial costs (Assumption 2.4), then by Definition 2.1, the shorter path will dominate the longer one and only the former can be bilevel feasible. Assumption 2.4 can be satisfied by simply adding a small random perturbation to the cost of each arc so that the total cost of every path is unique.

Yen's algorithm [57] is usually used to enumerate the set of paths of a commodity. The algorithm outputs the shortest path, the second shortest path, etc, up to the K-th shortest path between two nodes in a graph. In this case, the shortest path is the path with the minimum initial cost (the cost when $T = 0$). The algorithm should be stopped when it has found the first toll-free path as in the following corollary:

Corollary 2.5. *Given a commodity $k \in K$, a path p cannot be bilevel feasible if its initial cost is greater than that of the shortest toll-free path π^k.*

In Bouhtou et al. [6], the authors reduced the size of the graph by transforming the original graph to the shortest path graph model (SPGM). In SPGM, all nodes that are not incident to any tolled arc will be removed and the toll-free arcs connecting to it are replaced by the outer product of the list of incoming arcs and the list of outgoing arcs. Then, some elimination rules (which are only applicable on the SPGM) are applied to reduce the number of arcs. Finally, the set of paths is enumerated using the Yen's algorithm on the SPGM and the dominance rule is applied afterward. A drawback of the SPGM is the inflation of the set of arcs which is caused by the replacement of nodes by arcs. Besides that, the elimination rules in SPGM still leave many redundant nodes and arcs. Thus, there is a sizable number of irrelevant paths generated by the Yen's algorithm.

In Didi-Biha et al. [24], the authors used a modified version of the Lawler's procedure [41], a general form of the Yen's algorithm, to enumerate the set of bilevel feasible paths directly without using the SPGM. SPGM is not needed here because it is only a method to reduce the size of the graph before Yen's algorithm is applied which is not used by the authors. Due to Assumption 2.4, only paths with different combinations of tolled arcs are explored, ignoring most redundant paths. However, the algorithm in Didi-Biha et al. [24] requires solving a shortest path problem with some arcs fixed to 1 which cannot be accomplished by the Dijsktra's algorithm and requires a linear program solver. We propose an improved version of this algorithm which just needs a shortest path algorithm. Algorithm 2.1 shows this enumeration process in details. In the algorithm, C is the set of candidate paths, N is the maximum number of paths we want to enumerate, $p^{(j)}$ is the j-th shortest path. We also save two extra properties along with each path q^i: $s(q^i)$ which is called the spur node of path q^i, and $R(q^i)$ which is the set of excluded tolled arcs of path q^i.

Algorithm 2.1 is a version of the Lawler's algorithm [41] which is an enumeration scheme for binary problems such as the shortest path problem. Let $S(p^{(j)})$ be the set of tolled arcs of $p^{(j)}$ from o^k to $s(p^{(j)})$. Consider the problem, denoted as $(R(p^{(j)}), S(p^{(j)}))$, where we need to find the shortest path when the variables associated to the arcs in $R(p^{(j)})$ are fixed to 0 and to the arcs in $S(p^{(j)})$ are fixed to 1. It is simple to verify that $p^{(j)}$ is a point in the feasible region of the problem $(R(p^{(j)}), S(p^{(j)}))$. The path $p^{(j)}$ is not necessarily optimal to the problem, but this is not required as we show later in this section. The Lawler's algorithm tells us that we need to spawn n subproblems where n is the number of non-fixed variables. We can re-order these non-fixed variables in any order. Hence, the variables with value 1 (x_{a_1} to x_{a_m}) come first and the variables with value 0 ($x_{a_{m+1}}$ to x_{a_n}) follow. The former m variables are the tolled arcs of $p^{(j)}$ from $s(p^{(j)})$ to d^k (all the tolled arcs preceding $s(p^{(j)})$ are fixed by assumption). These m variables must also be in order of appearance as mentioned in Algorithm 2.1 (line 11). According to this order, the n subproblems of the Lawler's algorithm are generated by fixing all the previously fixed variables with these additional constraints (one constraint for each subproblem):

$$\begin{array}{ll}
(1) & x_{a_1} = 0 \\
(2) & x_{a_1} = 1, x_{a_2} = 0 \\
\vdots & \vdots \\
(m) & x_{a_1} = x_{a_2} = \ldots = x_{a_{m-1}} = 1, x_{a_m} = 0
\end{array}$$

Algorithm 2.1 Path enumeration

```
Input: The graph $G = (V, A)$, the O-D pair $o^k, d^k$, the initial costs $c_a$, the maximum number of
       paths to be enumerated $N$.
Output: A list of $N$ paths, most of which are bilevel feasible.
 1: Find the first shortest path $p^{(1)}$
 2: $C \leftarrow \{p^{(1)}\}$
 3: $s(p^{(1)}) \leftarrow o^k$
 4: $R(p^{(1)}) \leftarrow \varnothing$
 5: $j \leftarrow 1$
 6: while $j \leq N$ do
 7:     Output the path with minimum cost in $C$, call it $p^{(j)}$
 8:     Remove $p^{(j)}$ from $C$
 9:     if $p^{(j)}$ has no tolled arcs then
10:         Stop the algorithm
11:     Let $a_1, a_2, \ldots, a_m$ be the ordered tolled arcs of $p^{(j)}$ from $s(p^{(j)})$ to $d^k$
12:     $\hat{s} \leftarrow s(p^{(j)})$
13:     for $i$ from 1 to $m$ do
14:         $q^i \leftarrow$ subpath of $p^{(j)}$ from $o^k$ to $\hat{s}$
15:         $s(q^i) \leftarrow \hat{s}$
16:         $R(q^i) \leftarrow R(p^{(j)}) \cup \{a_i\}$
17:         Remove nodes in $q^i$ (for this loop only)
18:         Remove arcs in $R(q^i)$ (for this loop only)
19:         if $\hat{s}$ and $d^k$ are connected then
20:             Append to $q^i$ the shortest path from $\hat{s}$ to $d^k$
21:             Add $q^i$ to $C$
22:         $\hat{s} \leftarrow$ target of $a_i$
23:     $j \leftarrow j + 1$
```

$$\begin{array}{ll} (m+1) & x_{a_1} = \ldots = x_{a_m} = 1; x_{a_{m+1}} = 1 \\ (m+2) & x_{a_1} = \ldots = x_{a_m} = 1; x_{a_{m+1}} = 0, x_{a_{m+2}} = 1 \\ \vdots & \vdots \\ (n) & x_{a_1} = \ldots = x_{a_m} = 1; x_{a_{m+1}} = \ldots = x_{a_{n-1}} = 0, x_{a_n} = 1. \end{array}$$

We can verify that the feasible regions of all subproblems are mutually exclusive and their union is equivalent to the feasible region of the problem $(R(p^{(j)}), S(p^{(j)}))$ except $p^{(j)}$. The subproblems in the second group have a common constraint $x_{a_1} = \ldots = x_{a_m} = 1$. However, because of Lemma 2.2, $x_{a_1} = \ldots = x_{a_m} = 1$ implies that the resulting paths cannot be bilevel feasible (dominated by $p^{(j)}$). Therefore, we only need to consider the first m subproblems (line 13).

Consider the subproblem i. Let q^i be the path constructed as in Algorithm 2.1 (lines 14, 20). We call q^i a child of $p^{(j)}$. There are two cases (depending on the condition in line 19): we can construct q^i and we cannot (there is no path from $\hat{s}$ to d^k). If we can and q^i exists, then similar to $p^{(j)}$, q^i is a feasible point of the problem $(R(q^i), S(q^i))$ where $R(q^i) = R(p^{(j)}) \cup \{a_i\}$ (line 16) and $S(q^i) = S(p^{(j)}) \cup \{a_1, \ldots, a_{i-1}\}$ (lines 12, 15, 22). Although we do not assume that $p^{(j)}$ and q^i are optimal in their own subproblems, they satisfy two special properties. First, the costs of $p^{(j)}$ and its child q^i follow the correct enumeration order:

Lemma 2.6. *The cost of q^i is at least the cost of $p^{(j)}$.*

Hereafter, we will use the term *part* to refer to a subpath of $p^{(j)}$ or q^i of which one end is either the origin o^k or the destination d^k. The term *segment* will refer to any other subpath.

Proof of Lemma 2.6. Because $p^{(j)}$ is constructed by Algorithm 2.1, its part from $s(p^{(j)})$ to d^k is the shortest path excluding arcs in $R(p^{(j)})$ and all nodes preceding $s(p^{(j)})$. As a result, its part from $s(q^i)$ to d^k is also the shortest path with respect to the same set of constraints. Now, consider the path q^i. Its part from o^k to $s(q^i)$ is identical to that of $p^{(j)}$. The other part from $s(q^i)$ to d^k is the shortest path, while excluding arcs in $R(q^i)$ and all nodes preceding $s(q^i)$. Given that $R(p^{(j)}) \subset R(q^i)$ and $s(p^{(j)})$ comes before $s(q^i)$, the new shortest path problem is a restriction. Thus, the cost of this part of q^i must be at least the cost of the same part of $p^{(j)}$. $\square$

Lemma 2.6 is fundamental for using Corollary 2.5 as the stopping condition (line 9). If the enumeration order is not maintained, then we may stop prematurely after encountering the first toll-free path. The other property is expressed in the lemma below and it will be used for proving the correctness of another pruning condition.

Lemma 2.7. *Any toll-free segment of q^i (or any path generated by the algorithm) is the shortest segment while excluding all arcs in $R(q^i)$ and all the preceding nodes of that segment.*

Proof. Assume that q^i is a child of $p^{(j)}$ and the lemma is true for $p^{(j)}$. The part of $p^{(j)}$ from o^k to $s(q^i)$ is reused for q^i and does not contain any arc in $R(q^i)$, so the lemma is true for this part. Consider the other part of q^i from $s(q^i)$ to d^k. By construction in Algorithm 2.1, this part is the shortest path, while excluding arcs in $R(q^i)$ and nodes preceding $s(q^i)$. As a result, any subpath of this part is also the shortest subpath with respect to the same set of constraints. Later, for any toll-free segment in this part, we can extent the set of excluded nodes from the set of nodes that precedes $s(q^i)$ to the set of nodes that precedes that segment. This means that the lemma is also true for this part of q^i. Recall that $s(q^i)$ is the target of some tolled arc, thus there are no toll-free segments containing $s(q^i)$ in the middle and the lemma is true for q^i. To finish the induction proof, we consider the base case of the first shortest path for which the lemma is also true. □

Now, we consider the other case where we cannot construct q^i (condition in line 19 is false). This means $s(q^i)$ and d^k are disconnected when we remove arcs in $R(q^i)$ and nodes preceding $s(q^i)$. We will prove that there is no bilevel feasible path in the feasible region of $(R(q^i), S(q^i))$:

Lemma 2.8. *If q^i does not exists, any path in the feasible region of $(R(q^i), S(q^i))$ is not bilevel feasible.*

See Appendix 2.B for the proof of Lemma 2.8.

Theorem 2.9. *Given a sufficiently large value of N, the output of Algorithm 2.1 includes all bilevel feasible paths.*

Proof. At any step of the algorithm, the subproblems generated by the Lawler's procedure do not remove any part of the feasible region of the original problem. The subproblems are mutually exclusive and there is a finite number of paths, thus the algorithm is guaranteed to stop in finite time. A subproblem i is pruned if and only if it is in the second group of subproblems or if q^i does not exists. Both cases imply that all paths in the feasible region of the subproblem are not bilevel feasible (Lemma 2.8). Besides that, the stopping condition in Corollary 2.5 is ensured by Lemma 2.6. Therefore, no bilevel feasible path is ruled out. □

Although the set of paths enumerated in Algorithm 2.1 is close to the final set of bilevel feasible paths, there are still redundant paths. To obtain the final set of bilevel feasible paths, the dominance rule in Lemma 2.2 still needs to be applied once more. For an example of path enumeration that produces redundant paths, see Appendix 2.C.

2.3.2. Path-based Preprocessing

In Didi-Biha et al. [24], Bouhtou et al. [6], the authors compare their proposed path-based formulations to an unprocessed or SPGM-processed standard formulation. In these comparisons, path-based formulations are more advantageous compared to their counterparts. Although SPGM can reduce the complexity of a graph to some degree, path-based formulations essentially bypass all the redundant arcs and nodes in the graph, hence they have more preprocessing power. However, we could harness this preprocessing power of path enumeration and apply it to arc-based formulations. We propose a new preprocessing method which uses the set of enumerated paths to remove all redundant arcs and nodes completely. The idea is simple: given the set of bilevel feasible paths, any arc or node which does not appear in any path will be removed. Then, to further reduce the size of the graph, chains of toll-free arcs (*i.e.* connected subgraph containing only 1-out-degree nodes) will be replaced with a single toll-free arc with cost equal to the sum of the costs of the associated arcs. This preprocessing method will let the arc-based formulations use the information generated from the path enumeration process and enable us to compare all the formulations in a fair manner.

In addition, SPGM can still be applied on top of the processed graph. This does not remove any additional tolled arcs because the set of tolled arcs obtained from path-based preprocessing is already minimal. However, SPGM transforms the graph by replacing nodes with toll-free arcs, which can sometimes reduce both the number of nodes and the number of toll-free arcs in sparse graphs.

As mentioned in Section 2.2.3, the conversions used in the (PACS) and the (VFCS) formulations must cover all solutions in the primal representation. When we apply the path-based preprocessing to the arc representation and use the set of bilevel feasible paths for the path representation, the (PACS) formulation still satisfies this rule. This is because the set of primal solutions of (PACS) is the set of bilevel feasible paths, all of them can be converted to arcs in the processed graph. On the other hand, the (VFCS) formulation does not satisfy this rule anymore. In the processed graph, there are many paths which are not bilevel feasible. If these paths are not covered in the (VFCS) formulation, the optimality condition can be bypassed by selecting these paths as the primal solutions. However, enumerating all paths (including non-bilevel-feasible) in the processed graph would be too costly. Therefore, a special treatment is necessary and we propose a cutting-plane method just for (VFCS): the processed graph and the set of bilevel feasible paths will still be used to generate the initial set of constraints. Then, when we encounter a solution, we will examine if the path in that solution is a bilevel feasible path. If it is not, we will use it to generate an additional complementary slackness constraint to cover the new path and continue solving.

2.4. Hybrid Model for Multi-Commodity Problems

Although path-based formulations and preprocessing can give the MIP solver a boost by removing redundant variables, they require time to enumerate the paths. Since the number of paths can be exponential, sometimes, it is faster to just solve the problem without relying on path enumeration (unprocessed or SPGM-processed STD or CS). However, it is unknown which option is better until the enumeration process is finished. Instead of a full enumeration, we could enumerate until a predetermined number of paths and then decide if it is worth to continue enumerating or to cancel the enumeration process and fallback to a formulation which does not require path enumeration. Since we have a multi-commodity problem, this probing method can be applied for each commodity separately. The end result is a hybrid model, assigning different formulations and preprocessing for different commodities depending on their numbers of paths. This process is shown in Algorithm 2.2. In the algorithm, N is called the breakpoint, which is a threshold to decide whether the enumeration process should be continued. The commodities with only one path are redundant, because the only path must be a toll-free path which does not bring any profit to the leader. Commodities with less than N paths are assigned to a path-based formulation or an arc-based formulation with path preprocessing, while commodities with more than N paths are assigned to a fallback formulation which does not need path enumeration. All variables are managed independently according to their corresponding commodities, except for T_a which is shared. An example is provided in Appendix 2.A.4.

Algorithm 2.2 Hybrid model

```
Input: The graph $G = (V, A)$ with costs $c_a$, all commodities $(\eta^k, o^k, d^k)$, the breakpoint $N$.
Output: A hybrid model $(obj, constraints)$.
  $obj \leftarrow 0$
  $constraints \leftarrow \varnothing$
  for $k \in K$ do
      Enumerate the first $N + 1$ paths of commodity $k$
      Let $\hat{P}^k$ be the set of enumerated paths
      if $|\hat{P}^k| > 1$ then
          if $|\hat{P}^k| \leq N$ then
              Assign a formulation $F^k$ which is either path-based or arc-based with path-based pre-
              processing
          else
              Assign an arc-based formulation $F^k$ without path-based preprocessing
          $obj \leftarrow obj + obj(F^k)$
          $constraints \leftarrow constraints \cup constraints(F^k)$
  Solve $(obj, constraints)$
```

The hybrid model can be extended to have multiple breakpoints and different assignment schemes. For example, path-based formulations are less effective for commodities which have many paths because the more paths a commodity has, the more variables and constraints are added. In contrast, arc-based formulations with path preprocessing only remove arcs and nodes which guarantee complexity reduction. We could use the hybrid model to balance between these two kinds of formulations, by assigning path-based formulations to commodities with few paths and arc-based formulations for those with many paths.

2.5. Experiments

We conducted computational experiments to evaluate in practice the reformulations and preprocessing presented in the previous sections. In Section 2.5.1, the experimental methodology is described, including the implementation details, the generation and properties of the instances used. Section 2.5.2 provides a comparison between the 12 reformulations listed in Section 2.2. Section 2.5.3 validates the efficiency of the new path-based preprocessing. In Section 2.5.4 we justify the use of the hybrid model introduced in Section 2.4. All instances, results and code are publicly available [1].

2.5.1. Methodology

The instances used in the tests are randomly generated. We employ a generation method similar to the one described in Brotcorne et al. [9] to make the problems challenging. We recall this generation process in Appendix 2.D.

There are 200 generated problem instances (in short, problems) divided into four sets which are summarized in Table 2.6. Each set has 50 problems with five different numbers of commodities: 30, 35, 40, 45, and 50 (10 problems generated for each value). The topologies are inspired from Brotcorne et al. [11] and are illustrated in Figure 2.1. The set G provides a dataset similar to the one used in Brotcorne et al. [9, 10], while the three other sets have more paths and are more challenging to solve. We note that the four sets of random instances used in our experiments contain a large variety of network topologies: (i) class G (and H) reproduces the network topologies broadly used in the NPP community, namely, in the papers on Table 2.5, (ii) class D, proposed in Brotcorne et al. [13], introduces a denser network, expected to have many paths, (iii) class V, also described in Brotcorne et al. [13], introduces a less dense network. We remark that these four sets of instances correspond to various planar graph topologies. Thus, we expect our results to be insightful from a practical point of view since, for example, road and rail networks are typically planar [2]. The cumulative distribution of the number of bilevel feasible paths of each set is shown in Figure 2.2. The horizontal axis represents the breakpoint N while the vertical axis shows the proportion of commodities with no more than N bilevel feasible paths.

Our code is implemented in C++. All test runs are executed single-threaded on the Béluga cluster from Compute Canada (Intel Xeon 2.4 GHz) under Linux. All tested models are instantiations of the hybrid model described in Algorithm 2.2. Given a problem, the path enumeration will be applied first, then a formulation will be assigned to each commodity and the single-level reformulation is synthesized. Finally, the single-level reformulation is solved

1. `https://github.com/minhcly95/netpricing`

Table 2.6 – Properties of the generated instances.

Label	Topology	Dimensions	$\|V\|$	Avg. $\|A\|$
G	Grid	5×12 nodes	60	206
H	Grid	12×12 nodes	144	528
D	Delaunay	-	144	832
V	Voronoi	-	144	410

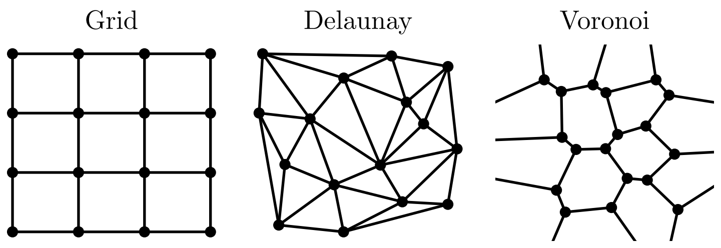


Figure 2.1 – Illustration of the three types of topology.

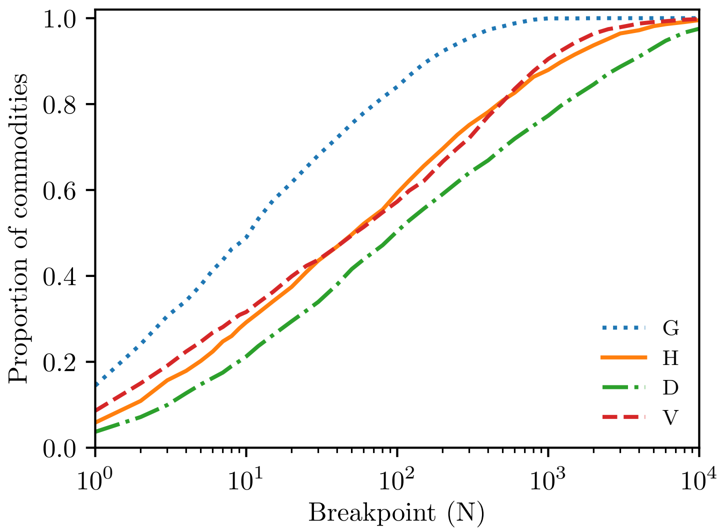


Figure 2.2 – Cumulative distribution of the number of bilevel feasible paths.

directly by CPLEX 12.9. If a model does not rely on path enumeration, the breakpoint N is set to 1 and path enumeration is still applied. This will remove the commodities with only one path and provide basic preprocessing for all models. The main formulation will be assigned for the commodities with less than N bilevel feasible paths. The default fallback formulation for commodities with more than N paths is the (STD) with no preprocessing.

By increasing N, we essentially replace the fallback formulation with the main formulation. If the main formulation is better, then we should see an improvement in performance when increasing N and vice versa. Some formulations are only good up to a certain breakpoint, which can also be observed if there is a change in the direction of their performance over N.

We imposed a time limit of one hour for each problem. The time for path enumeration is included in the total time of a test run. If the run exceeds one hour, it is stopped and the optimality gap is recorded. The problems are divided into two groups: easy and hard. The easy group consists of 108 problems which can be solved by at least one run. All the remaining 92 problems form the hard group, for which an optimal solution is not available. When we compare the performance of any two models, we are interested in three criteria: the number of problems solved by each model, the average time it takes to solve the easy group, and the average optimality gap of the hard group.

2.5.2. Comparison of all Formulations

In this section, we compare the performance of all 12 formulations. These will be assigned as the main formulations of the hybrid models. If the main formulation is arc-based, then path-based preprocessing is applied. If the main formulation is a mixed arc-path or path-arc formulation, then we use path-based preprocessing on the arc representation. If the main formulation is a path-based formulation, no further preprocessing is applied. In all cases, path enumeration is required for the main formulations.

Table 2.7 shows the performance of all models over N. At first look, we can see that formulations with suffix 1 worsen over N. This means they are even worse than the fallback formulation. These are the formulations with complementary slackness as optimality condition and direct linearization. The reason may be because both complementary slackness and direct linearization use big-M constraints. All other formulations perform roughly the same (except for (VFCS2)), with the (STD) taking the lead in all three criteria.

Figure 2.3 plots the performance over N of four notable formulations: (STD), (VF), (CS2), and (PCS2). (STD), (VF), and (CS2) represent three basic paradigms to convert a bilevel linear problem to a single-level reformulation. (PCS2) is a path-based formulation which is mentioned in Didi-Biha et al. [24] and Dewez et al. [23]. In these papers, their experiments suggest that (PCS2) outperforms (STD), but this comparison is done when the (STD) formulation is preprocessed by the SPGM method. Here, we observe the opposite result: the standard formulation outperformed (PCS2). The key is the new path-based preprocessing, which theoretically provides arc-based formulations with the same preprocessing power of the path enumeration, thus it levels the playing field. Figure 2.3 also shows a drawback of path-based formulations: the larger N is, the more complex they become. (PCS2)

Table 2.7 – Performance of all formulations over N.

(a) Number of problems solved

N	10	20	50	100	200	500	1000
STD	61	64	69	72	78	92	94
VF	59	64	68	68	75	90	93
PASTD	56	64	68	71	76	86	85
PVF	59	64	67	69	79	80	81
CS1	47	32	8	1	0	0	0
CS2	58	62	67	67	70	71	73
PACS1	49	33	8	1	0	0	0
PACS2	58	62	63	68	70	76	69
VFCS1	43	10	0	0	0	0	0
VFCS2	56	55	44	36	29	23	21
PCS1	48	33	9	2	0	0	0
PCS2	61	61	72	71	77	80	79

(b) CPU time of the easy group (s)

N	10	20	50	100	200	500	1000
STD	1999	1864	1739	1604	1430	1127	1029
VF	2022	1870	1730	1644	1520	1231	1155
PASTD	2056	1863	1724	1662	1498	1315	1220
PVF	1980	1867	1770	1699	1486	1409	1429
CS1	2396	2915	3495	3585	3601	3601	3601
CS2	2069	1973	1787	1822	1697	1599	1518
PACS1	2566	3368	3601	3601	3601	3601	3601
PACS2	2135	2189	2438	2687	2890	2986	3050
VFCS1	2359	2818	3491	3587	3601	3601	3601
VFCS2	1992	1916	1852	1712	1673	1572	1676
PCS1	2351	2861	3448	3576	3601	3601	3601
PCS2	2012	1903	1696	1599	1544	1529	1607

(c) Gap of the hard group (%)

N	10	20	50	100	200	500	1000
STD	12.8	12.7	11.9	11.3	10.5	9.3	8.5
VF	13.3	12.6	12.1	11.4	11.2	10.0	9.8
PASTD	13.2	12.8	11.7	11.7	10.8	9.7	9.2
PVF	13.6	12.7	12.1	11.7	11.1	10.1	10.6
CS1	16.3	21.8	42.1	70.0	94.2	99.6	100.0
CS2	13.3	12.9	12.8	12.2	11.9	10.9	10.9
PACS1	18.9	34.4	70.3	92.6	99.1	100.0	100.0
PACS2	13.7	13.4	14.6	15.1	15.5	17.1	19.2
VFCS1	16.3	20.3	36.3	58.2	86.2	97.7	99.5
VFCS2	13.6	13.5	13.5	13.3	12.8	12.1	12.1
PCS1	15.6	20.3	39.5	63.2	88.4	98.7	99.9
PCS2	13.2	13.3	12.4	11.8	10.8	10.6	11.1

must add an extra binary variable for every enumerated path, while (STD) and (CS2) use the set of paths to eliminate arcs and nodes. Thus, the complexity of (PCS2) increases over N while the complexity of the other two formulations is capped by its unprocessed version. In the graph, we can see that the performance of (PCS2) starts dropping after $N = 500$. The other formulations using primal-path representation ((PASTD), (PVF), (PACS)) also suffer from this problem. (VF) uses dual-path, so it only adds more constraints instead of binary variables which allows it to continue improving over N.

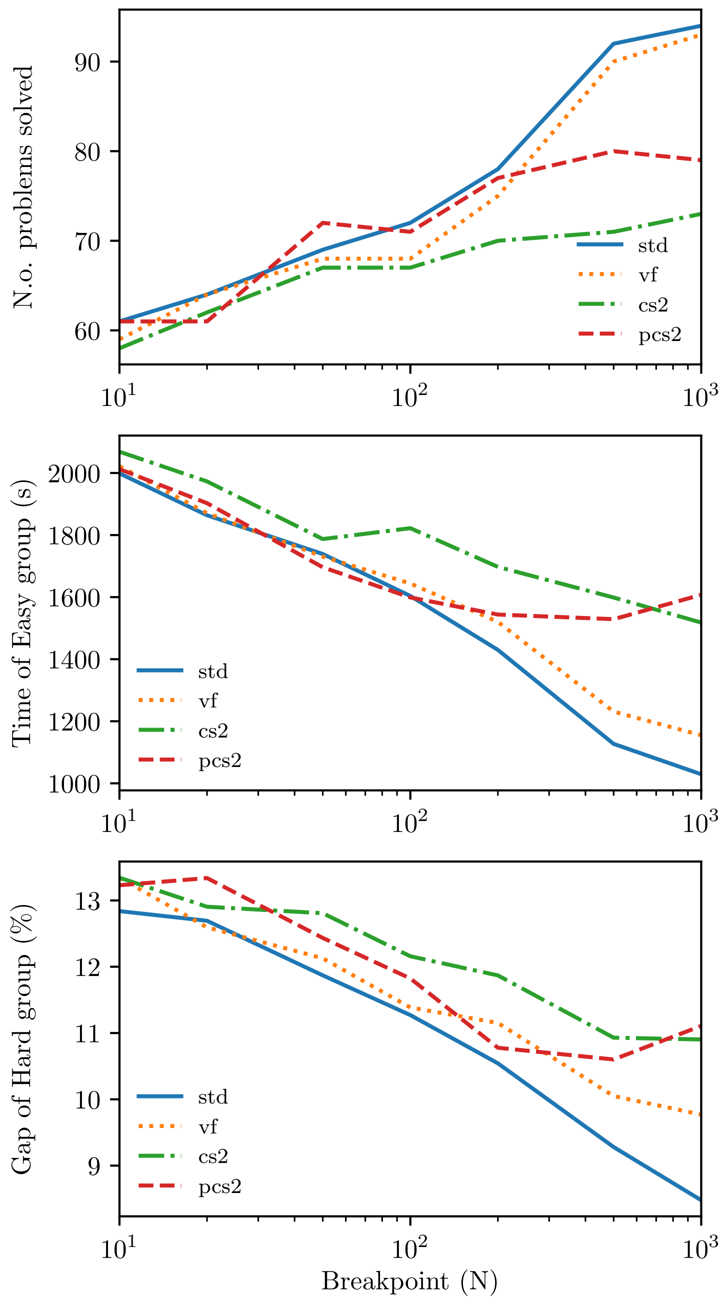


Figure 2.3 – Performance of notable formulations over N.

2.5.3. Path-based Preprocessing

In this section, we compare the new path-based preprocessing with the state-of-the-art preprocessing method (SPGM). Figure 2.4 shows the reduction ratio between those two preprocessing methods. Given a breakpoint N (horizontal axis), the reduction ratio is the quotient of the total number of nodes/arcs/tolled arcs of all commodities with no more

Table 2.8 – Impact of the reduction ratio criteria.

Criteria	Impact
Node	• Number of constraints in primal-arc
	• Number of variables in dual-arc
Arc	• Number of variables in primal-arc (strong duality)
	• Number of binary variables in primal-arc (complementary slackness)
	• Number of constraints in dual-arc
	• Number of constraints in complementary slackness
Tolled arc	• Number of binary variables in primal-arc
	• Number of variables and constraints in direct linearization

than N bilevel feasible paths after being processed by the path-based preprocessing over the same sum in the original graph (absolute ratio) or in the SPGM-processed graph (relative ratio). Each criterion (node, arc, tolled arc) has a different impact on the final single-level reformulation, which is summarized in Table 2.8. The tolled arc criterion is the most important because it is linked with the number of binary variables in the final formulation. In Figure 2.4, we can see that the path-based preprocessing excels in all three criteria. On average, graphs processed by the path-based preprocessing have 10% less nodes, 66% less arcs, and 49% less tolled arcs compare to the SPGM method. Compared to the original graph, the path-based preprocessing removes 75% of all tolled arcs, which reduces the size of the problems significantly. The efficiency varies between different instance sets, with the path-based preprocessing being more favorable in more connected graphs such as in dataset D and H.

Figure 2.5 shows the real performance of the two preprocessing methods over N. The main formulation in both models is the (STD), but one is matched with path-based preprocessing, while the other is matched with SPGM preprocessing. Once again, we can observe that the path-based preprocessing excels in all three criteria. It can solve 30 more problems (over 200 in total), in half the time and half the gap compared to the SPGM model. Besides that, the SPGM model hardly improves when $N > 1000$ while path-based preprocessing keeps thriving beyond this threshold. In our experiments, applying SPGM on top of the path-based preprocessing does not produce significantly better results, and sometimes can even be detrimental. This concludes that path-based preprocessing is the superior preprocessing method.

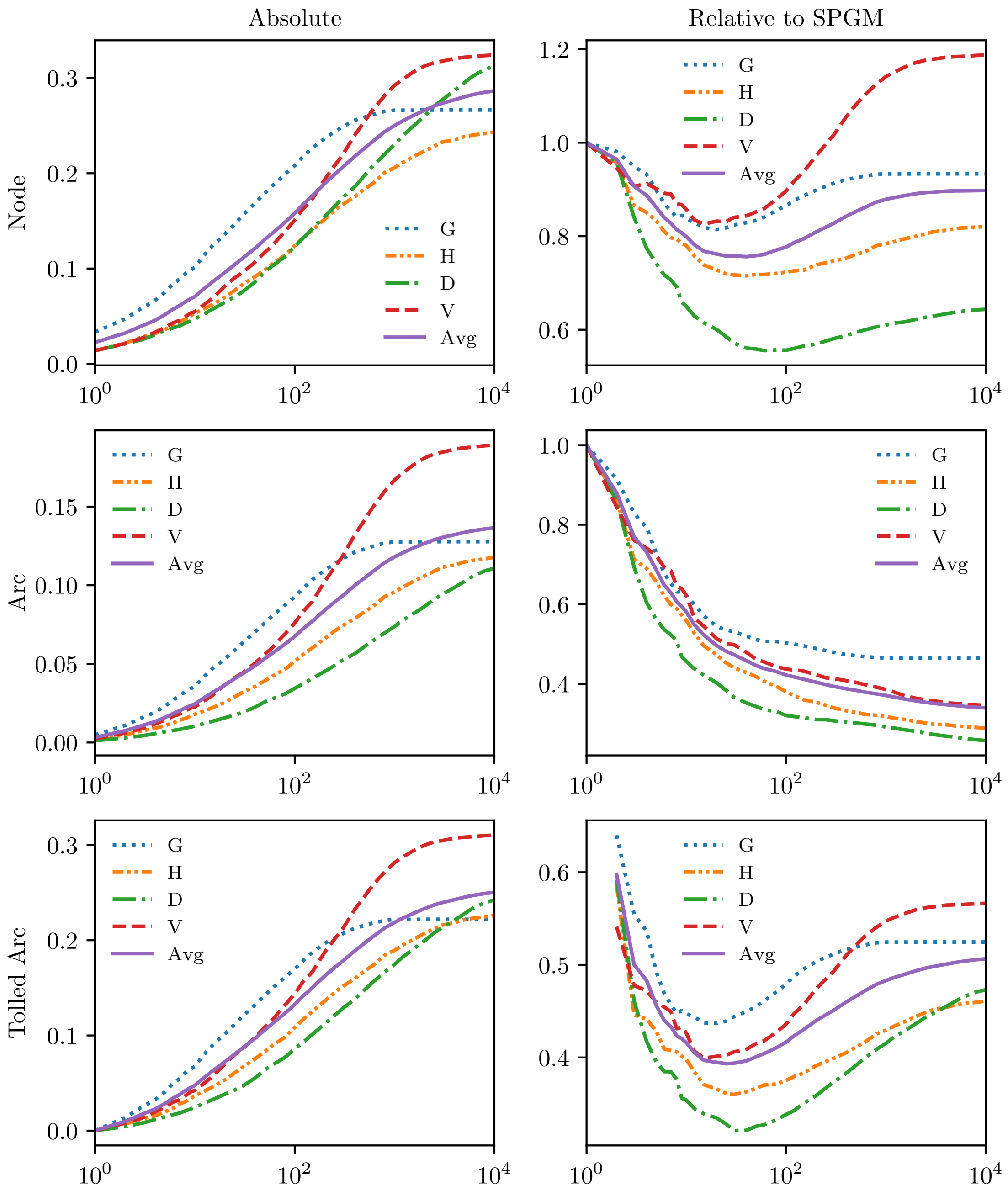


Figure 2.4 – Reduction ratio of path-based preprocessing.

2.5.4. Hybrid Model

In this work, we show numerical results to justify the idea of the hybrid model introduced in Section 2.4 as a way to compromise between the time spent for path enumeration and the time spent to solve the reformulation. We will use the best reformulation which is the (STD) for all commodities in this test. Figure 2.6 shows its performance with N varying from 10

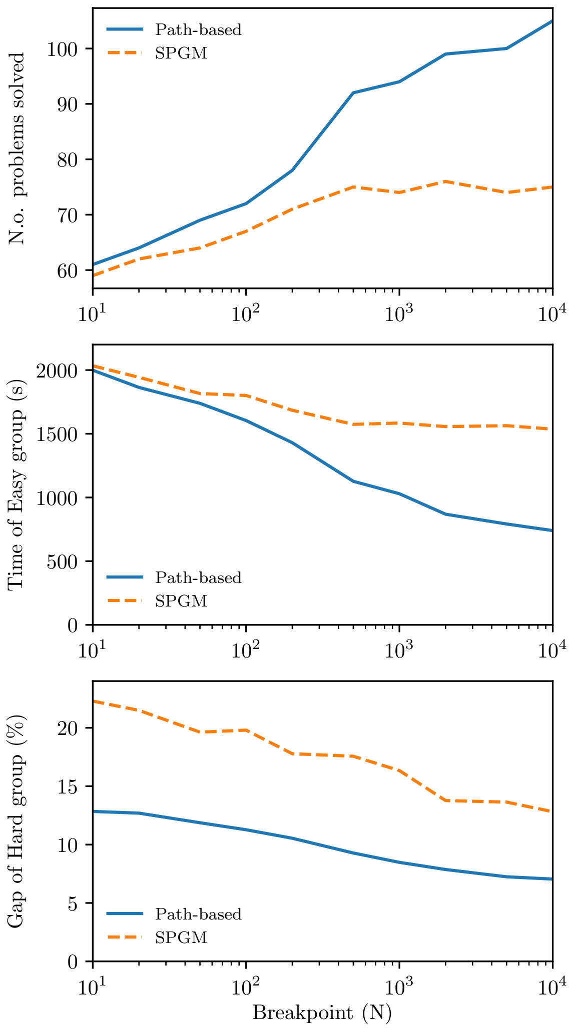


Figure 2.5 – Performance comparison between path-based preprocessing and SPGM preprocessing.

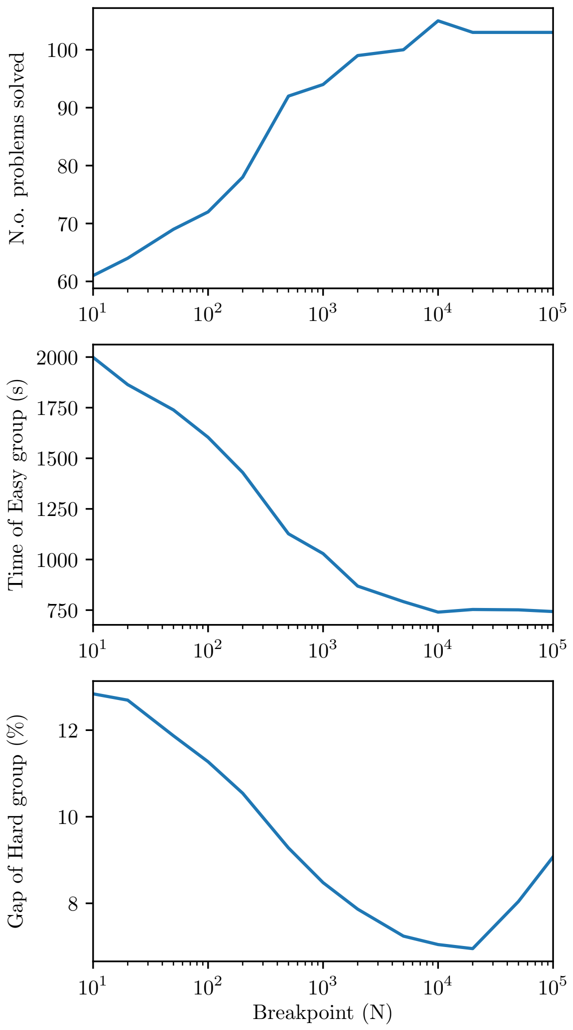


Figure 2.6 – Performance of the standard model over N.

to 100000 (horizontal axis). If a commodity has no more than N bilevel feasible paths, then path-based preprocessing is applied. Otherwise, the unprocessed graph is used instead. The larger N is, the longer it takes to enumerate, but more commodities will be processed. If N is very large, all commodities will be processed and all paths are enumerated. If N is set to 0, then it is identical to an unprocessed model. From Figure 2.6, we observe that larger N generally leads to better performance until a certain point around 10000 paths. After that, the time spent for path enumeration does not have a positive impact on the performance anymore. The average time of the easy group is not affected by large N because they do not have many paths to enumerate in the first place. At $N = 100000$, there are two unsolved problems in class D which require more than one hour for path enumeration and this contributes to the rise at the end in the average optimality gap of the hard group. In conclusion, for the given instances, stopping the path enumeration at $N = 10000$ provides a good balance. Stopping early also improves the robustness of the program, because although commodities with more than 10000 paths are not common, encountering one could make the program stuck for a long time.

2.6. Conclusion

In this work, we related many modeling concepts which enrich and unify the general toolbox to solve bilevel linear problems. Indeed, if the follower problem can be formulated by different linear programs, then we can mix the primal and the dual of any two formulations to write the single-level formulation. Remark that this alternative formulation always exists: *(i)* if the leader's variables appear in the objective of the follower, then the follower problem can be written as a linear combination of the extreme points of the convex hull of the feasible set, *(ii)* if the leader's variables appear in the right-hand-side of the follower's constraints, then its dual can be reformulated using the extreme points of the dual feasible region and a new primal formulation can be obtained. When *(i)* holds, bilevel feasibility, and its enumeration, can be used as a preprocessing method. The hybrid framework is applicable in any multi-commodity problem with multiple reformulations. The network pricing problem is a good example to demonstrate all these modeling techniques together.

The NPP makes some theoretical assumptions whose relaxation enables a more close mirroring of reality. From a practical point of view, future work must consider additional real-life features, including arc construction cost, capacity on arcs and nodes, congestion, competition, and uncertainty, many of which are highly non-linear and make the problem significantly harder.

2.A. Reformulations

2.A.1. Standard Formulation

$$
\begin{aligned}
\text{(STD)} \quad \max \; & \sum_{k\in K}\sum_{a\in A_1} \eta^k T_a x_a^k \\
\text{s.t.} \; & \sum_{a\in A_1^+(i)} x_a^k + \sum_{a\in A_2^+(i)} y_a^k - \sum_{a\in A_1^-(i)} x_a^k - \sum_{a\in A_2^-(i)} y_a^k = b_i^k, && k\in K, i\in V, \\
& \lambda_i^k - \lambda_j^k \le c_a + T_a, && k\in K, a\equiv(i,j)\in A_1, \\
& \lambda_i^k - \lambda_j^k \le c_a, && k\in K, a\equiv(i,j)\in A_2, \\
& \sum_{a\in A_1}(c_a+T_a)x_a^k + \sum_{a\in A_2} c_a y_a^k = \lambda_{o^k}^k - \lambda_{d^k}^k, && k\in K, \\
& x_a^k \ge 0, && k\in K, a\in A_1, \\
& y_a^k \ge 0, && k\in K, a\in A_2, \\
& T_a \ge 0, && a\in A_1.
\end{aligned}
$$

2.A.2. Value Function Formulation

$$
\begin{aligned}
\text{(VF)} \quad \max \; & \sum_{k\in K}\sum_{a\in A_1} \eta^k T_a x_a^k \\
\text{s.t.} \; & \sum_{a\in A_1^+(i)} x_a^k + \sum_{a\in A_2^+(i)} y_a^k - \sum_{a\in A_1^-(i)} x_a^k - \sum_{a\in A_2^-(i)} y_a^k = b_i^k, && k\in K, i\in V, \\
& L^k \le \sum_{a\in A} \delta_a^p c_a + \sum_{a\in A_1} \delta_a^p T_a, && k\in K, p\in P^k, \\
& (c_a+T_a)x_a^k + \sum_{a\in A_2} c_a y_a^k = L^k, && k\in K, \\
& x_a^k \ge 0, && k\in K, a\in A_1, \\
& y_a^k \ge 0, && k\in K, a\in A_2, \\
& T_a \ge 0, && a\in A_1.
\end{aligned}
$$

2.A.3. Path Complementary Slackness Formulation

$$
\begin{aligned}
\text{(PCS)} \qquad & \max \sum_{k\in K}\sum_{a\in A_1}\sum_{p\in P^k} \eta^k \delta_a^p T_a z_p^k \\
& \text{s.t.} \sum_{p\in P^k} z_p^k = 1, && k\in K,\\
& L^k \le \sum_{a\in A}\delta_a^p c_a + \sum_{a\in A_1}\delta_a^p T_a, && k\in K, p\in P^k,\\
& \left(\sum_{a\in A}\delta_a^p c_a + \sum_{a\in A_1}\delta_a^p T_a - L^k\right) z_p^k = 0, && k\in K, p\in P^k,\\
& z_p^k \ge 0, && k\in K, p\in P^k,\\
& T_a \ge 0, && a\in A_1.
\end{aligned}
$$

2.A.4. Hybrid Model

Suppose we divide the set of commodities K into three parts based on the number of paths: K_1 for commodities having only one path, K_2 for commodities with less than N paths, and K_3 for commodities with more than N paths. If we assign (PCS2) to K_2 and (STD) to K_3, then we get the following hybrid model:

$$
\begin{aligned}
& \max \sum_{k\in K_2}\eta^k\tau^k + \sum_{k\in K_3}\sum_{a\in A_1}\eta^k t_a^k \\
& \text{s.t. } (2.4), (2.5), && k\in K_2,\\
& L^k \ge \sum_{a\in A}\delta_a^p c_a + \sum_{a\in A_1}\delta_a^p T_a - S_p^k(1-z_p^k), && k\in K_2, p\in P^k,\\
& \sum_{p\in P^k}\sum_{a\in A}\delta_a^p c_a z_p^k + \tau^k = L^k, && k\in K_2,\\
& z_p^k \in \{0,1\}, && k\in K_2, p\in P^k,\\
& (2.1), (2.2), (2.3), (2.19), (2.20), && k\in K_3,\\
& \sum_{a\in A_1}(c_a x_a^k + t_a^k) + \sum_{a\in A_2} c_a y_a^k = \lambda_{o^k}^k - \lambda_{d^k}^k, && k\in K_3,\\
& x_a^k \in \{0,1\}, && k\in K_3, a\in A_1.
\end{aligned}
$$

2.B. Proof of Lemma 2.8

If the problem $(R(q^i), S(q^i))$ is infeasible, then the above statement is trivially true. Suppose it has a feasible solution $\hat{q}^i$ which is a simple path (path with no loops). We will construct another path (not necessarily a solution of $(R(q^i), S(q^i))$) which dominates $\hat{q}^i$ by replacing a segment in $\hat{q}^i$ by a shorter segment in $p^{(j)}$. First, we need to sort the arcs in $S(q^i)$ by the order of appearance in $p^{(j)}$:

$$\hat{a}_1, \hat{a}_2, \hat{a}_3, \ldots, \hat{a}_{n-1}, \hat{a}_n$$

where $n = |S(q^i)|$. We also define $\hat{a}_0$ as a virtual arc whose target is o^k (its source is not relevant). Next, for $0 \leq i < n$, we will try to replace the segment of $\hat{q}^i$ between $\hat{a}_i$ and $\hat{a}_{i+1}$ with the same segment of $p^{(j)}$ if $\hat{a}_{i+1}$ comes after $\hat{a}_i$ in $\hat{q}^i$ and if the latter segment has smaller cost. If we are able to make such a replacement, due to the definition of $S(q^i)$, the segment connecting $\hat{a}_i$ and $\hat{a}_{i+1}$ in $p^{(j)}$ is toll-free; hence, we only remove tolled arcs while building a better path, and thus the resulting path dominates $\hat{q}^i$. If we cannot replace any segment, this means that either all segments in $\hat{q}^i$ are identical to those in $p^{(j)}$ or there is a segment in $\hat{q}^i$ with smaller cost.

Consider the first case: all segments in $\hat{q}^i$ are identical to those in $p^{(j)}$. In this case, the parts of $\hat{q}^i$ and $p^{(j)}$ from o^k to $s(q^i)$ are identical. The other part of $\hat{q}^i$ must be a path from $s(q^i)$ to d^k. Because we assume that $\hat{q}^i$ is a simple path, we must exclude all nodes preceding $s(q^i)$. However, since we also assume that q^i does not exist, this implies $s(q^i)$ and d^k to be disconnected if we exclude those nodes. This is a contradiction.

Consider the second case: there is a segment in $\hat{q}^i$ with smaller cost. Let $\hat{a}_m$ be the first arc such that the segment of $\hat{q}^i$ from $\hat{a}_m$ to $\hat{a}_{m+1}$ is cheaper than its counterpart of $p^{(j)}$. All segments preceding $\hat{a}_m$ in both $\hat{q}^i$ and $p^{(j)}$ must be identical since we assume that we cannot replace any segment. Consequently, the part from o^k to $\hat{a}_m$ of both paths are the same. By Lemma 2.7, the segment of $p^{(j)}$ from $\hat{a}_m$ to $\hat{a}_{m+1}$ is the optimal path while excluding all preceding nodes. However, the same segment of $\hat{q}^i$ is better, which means it must violate that constraint and must repeat some node preceding $\hat{a}_m$. This is again a contradiction as we assumed that $\hat{q}^i$ is a simple path.

Since both cases result in a contradiction, we can always replace some segment in $\hat{q}^i$ with a better segment in $p^{(j)}$. Therefore, $\hat{q}^i$ cannot be bilevel feasible.

2.C. Example of Path Enumeration Producing Redundant Paths

Consider the graph in Figure 2.7. The number of each arc represents the initial cost c_a. The first shortest path is $o^k - u - v - d^k$ with the cost of 3. By Algorithm 2.1, three subproblems are generated, each producing a candidate path. They are the second shortest path $o^k - u - d^k$ (cost 4), the third shortest path $o^k - u - v - w - d^k$ (cost 6), and the toll-free path $o^k - d^k$ (cost 10). All four paths will be returned, however, the path $o^k - u - d^k$

dominates $o^k - u - v - w - d^k$. The final set of bilevel feasible paths only has three paths: $o^k - u - v - d^k$, $o^k - u - d^k$, and $o^k - d^k$.

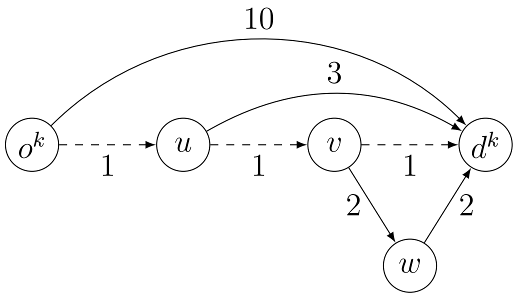


Figure 2.7 – Graph with redundant paths (dashed arcs are tolled arcs).

2.D. Instance Generation by Brotcorne et al.

For a given graph and a set of O-D pairs, first, the shortest path of each commodity is found. Next, we count the number of paths passing through each arc and sort them in the descending order according to that count. Following that order, each arc is converted into a tolled arc until 2/3 of the desired number of tolled arcs is reached. The last 1/3 is selected randomly among all remaining arcs. An arc is converted only if it does not remove the last toll-free path for all commodities. To make the generated data more realistic, we also enforce the properties of the arc to be symmetrical, which means that any arc and its reversed arc will have the same cost, and both must be either tolled or toll-free. The cost of 80% of all arcs will be distributed uniformly from 5 to 35, while the remaining 20% will have the maximum cost of 35. The cost of tolled arcs are halved after the conversion. The proportion of tolled arcs is 20%.

Chapter 3 Second article

Asymmetry in the Complexity of the Multi-Commodity Network Pricing Problem

by

Quang Minh Bui[1], Margarida Carvalho[1], and José Neto[2]

([1]) CIRRELT and Département d'informatique et de recherche opérationnelle
Université de Montréal
([2]) Télécom SudParis
Institut Polytechnique de Paris

This article was accepted for publication in Mathematical Programming.

My contributions and the roles of the coauthors:

— José Neto observed that some pairs of paths are redundant if they share the same structure in the network.
— I came up with the ideas of the reaction plot and the conjugate model. Later, I applied them to José's observation and derived the concept of strong bilevel feasibility.
— I devised the theoretical results which were revised by Margarida and José.
— The experiments were designed, implemented, and run by me.
— The article (including all proofs and examples) was written by me and it was revised by Margarida and José.

Abstract. The network pricing problem (NPP) is a bilevel problem, where the leader optimizes its revenue by deciding on the prices of certain arcs in a graph, while expecting the followers (also known as the commodities) to choose a shortest path based on those prices. In this paper, we investigate the complexity of the NPP with respect to two parameters: the number of tolled arcs, and the number of commodities. We devise a simple algorithm showing that if the number of tolled arcs is fixed, then the problem can be solved in polynomial time with respect to the number of commodities. In contrast, even if there is only one commodity, once the number of tolled arcs is not fixed, the problem becomes NP-hard. We characterize this asymmetry in the complexity with a novel property named strong bilevel feasibility. Finally, we describe an algorithm to generate valid inequalities to the NPP based on this property, whose numerical results illustrate its potential for effectively solving the NPP with a high number of commodities.

Keywords: Strong bilevel feasibility, Conjugate model, Network pricing problem, Complexity asymmetry

3.1. Introduction

In the most basic version of the network pricing problem (NPP), we have an optimization problem involving a graph and two decision makers: the leader and the follower. First, the leader adjusts the prices of some arcs in the graph (called tolled arcs). Subsequently, the follower finds the shortest path between its origin and destination according to the prices set by the leader. The objective of the leader is to maximize the overall revenue which depends on whether the follower uses the tolled arcs or not. To accomplish this goal, the leader wants to set the prices as high as possible, but not too high so that the follower is still incentivized to use the leader's service. In this work, we are interested in the multi-commodity variant of the NPP, in which there are multiple followers, each one has its own origin/destination, and each one chooses its own shortest path across the network.

Motivation. The single-commodity NPP was first introduced by Labbé et al. [40]. Since the follower's problem is the shortest path problem, it can be written as a linear program. Using the Karush-Kuhn-Tucker (KKT) conditions, the NPP and its multi-commodity variant can be formalized as a mixed-integer linear program (MILP) [10]. Bui et al. [14] summarized and extended various techniques to derive an MILP for the NPP, which covers path-based formulations [24, 6] and preprocessing [6, 55]. Non-MILP methods include multipath enumeration [12] and tabu search [13].

The single-commodity NPP has been proven to be NP-hard [51]. The complexity of the NPP is tied to the number of tolled arcs in the graph. Indeed, if we consider the case where there is only one tolled arc and multiple commodities, then the NPP can be solved in polynomial time [39]. Thus, an asymmetry exists in the complexity of the NPP between the number of tolled arcs (denoted as $|\mathcal{A}_1|$) and the number of commodities (denoted as

$|\mathcal{K}|$). For the case with multiple commodities and multiple tolled arcs, van Hoesel et al. [55] proved that if $|\mathcal{A}_1|$ is fixed, then the NPP can be solved in polynomial time with respect to $|\mathcal{K}|$. Specifically, the NPP can be decomposed into $|\mathcal{K}|^{f(|\mathcal{A}_1|)}g(|\mathcal{A}_1|)$ linear programs of size $h(|\mathcal{A}_1|)$ each, where $f(\cdot), g(\cdot), h(\cdot)$ are functions of exponential order. Even with very small $|\mathcal{A}_1|$, the number of linear programs required to be solved is enormous, thus the algorithm described in [55] has no practical use.

Contributions and Paper Organization. In this paper, we improve the result regarding the asymmetry in the complexity with the following theorem:

Theorem 3.1. *If the number of tolled arcs $|\mathcal{A}_1|$ is fixed, then the multi-commodity NPP can be solved in polynomial time with respect to the number of commodities $|\mathcal{K}|$, specifically by solving $(|\mathcal{K}|+1)^{|\mathcal{A}_1|}$ linear programs, each with size polynomial in $|\mathcal{A}_1|$.*

Compared to [55], the number of linear programs is much smaller. Proving Theorem 3.1 is the main focus of Section 3.2. In that section, we first revise the complexity of the single-commodity and the single-tolled-arc cases. Then, we extend the asymmetry in the complexity to the multi-commodity case in an intuitive way by using *reaction plots*. Finally, we provide a rigorous proof of this asymmetry via a new reformulation of the NPP called the *conjugate model*.

With the key result proven, we aim to exploit this asymmetry in a practical manner. In Section 3.3, we introduce and utilize a new concept named *strong bilevel feasibility* to generate cuts to existing MILP formulations of the NPP. Strong bilevel feasibility is a property of a composition of paths across multiple followers. We show that there is always a solution of the NPP that is strongly bilevel feasible, thus it is sufficient to enumerate only the strongly bilevel feasible points. This is analogous to the fact that it is sufficient to enumerate only the extreme points of the feasible set to solve a linear program. We also derive the characteristics of strong bilevel feasibility with the help of convex conjugates.

Section 3.4 describes a cut generation procedure using strong bilevel feasibility. We conduct numerical experiments on grid graphs to demonstrate that the asymmetry in the complexity is relevant in practice and that these cuts hold potential to effectively accelerate the solution of the problem instances with a very high number of commodities. Section 3.5 concludes the paper.

3.2. Asymmetry in the Complexity

Theorem 3.1 implies that the multi-commodity NPP scales differently with respect to two parameters: the number of commodities or the number of tolled arcs. To understand the difference between these two parameters, it is helpful to build some intuition from simpler cases: the single-commodity case (Section 3.2.1), and the single-tolled-arc case (Section 3.2.2). Then, in Section 3.2.3, we introduce an illustration tool called reaction plot, and expand the intuition to the multi-commodity variant by superimposing these reaction plots on top of each other. Finally, we materialize this intuition and provide a proof of Theorem 3.1 in Section 3.2.4.

3.2.1. Single-Commodity Case

First, we need to describe the NPP in its single-commodity form. Let us consider a directed graph $G = (\mathcal{V}, \mathcal{A})$ where $\mathcal{V}$ and $\mathcal{A}$ are the set of vertices and the set of arcs, respectively. The leader controls the costs of a subset of arcs $\mathcal{A}_1 \subsetneq \mathcal{A}$, designated as tolled arcs. All the other arcs form the set of toll-free arcs $\mathcal{A}_2 = \mathcal{A} \setminus \mathcal{A}_1$. The cost of a toll-free arc $a \in \mathcal{A}_2$ is always $c_a \geq 0$, while the cost of a tolled arc $a \in \mathcal{A}_1$ is $c_a + t_a$ where $t_a \geq 0$ is the price set by the leader (we only consider non-negative toll prices).

The follower wants to travel from an origin node $o \in \mathcal{V}$ to a destination $d \in \mathcal{V}$ via the shortest path across the graph. After the follower has chosen the path, the leader collects a revenue equal to the sum of t_a over the arcs that the follower uses. If there are multiple shortest paths, the follower chooses the path that produces the highest revenue for the leader (optimistic assumption). To prevent the leader to extract infinite revenue from the follower, we assume that there always exists a toll-free path from o to d.

Let $x \in \{0,1\}^{\mathcal{A}}$ represent the selection of arcs corresponding to the shortest path chosen by the follower, where $x_a = 1$ if arc $a \in \mathcal{A}$ is in the path and $x_a = 0$ otherwise. The NPP can be formulated as a bilevel program:

$$\max_{t,x}\{t^\top x \mid t \in \mathcal{T}, x \in \mathbf{R}(t)\}$$

where $\mathbf{R}(t)$ is the reaction set of the follower, defined by:

$$\mathbf{R}(t) = \operatorname*{argmin}_{x}\{(c+t)^\top x \mid Ax = b, x \geq 0\}.$$

The set $\mathcal{T} = \{t \geq 0 \mid t_a = 0, \forall a \in \mathcal{A}_2\}$ contains all vectors of feasible toll prices. The matrix A is the incidence matrix of the graph, while b is the source-sink vector of the shortest path problem: $b_o = 1$, $b_d = -1$, and $b_i = 0$ for all other nodes $i \in \mathcal{V}$.

The canonical way to solve the NPP is to convert the bilevel program into a single-level reformulation using the KKT conditions (with strong duality) [10]:

$$\max_{t,x,y} \; t^\top x \tag{3.1a}$$

$$\text{s.t. } t \in \mathcal{T}, \tag{3.1b}$$

$$Ax = b, \tag{3.1c}$$

$$A^\top y \leq c + t, \tag{3.1d}$$

$$(c+t)^\top x = b^\top y, \tag{3.1e}$$

$$x \geq 0. \tag{3.1f}$$

To solve this problem as an MILP, we need to linearize the bilinear term $t^\top x$ appearing in the objective function (3.1a) and the strong duality constraint (3.1e). Since the optimal solution of the shortest path problem needs x to be binary, the McCormick envelope [46] can be applied by replacing $t_a x_a$ with s_a and appending for all $a \in \mathcal{A}_1$:

$$0 \leq s_a \leq M_a x_a, \qquad 0 \leq t_a - s_a \leq M_a(1 - x_a), \qquad x_a \in \{0, 1\},$$

to Program (3.1) [10]. Appropriate values of the big-M parameters M_a can be found in Dewez et al. [23].

In this form, we can see that the variables x_a, $a \in \mathcal{A}_1$, are the only binary variables in the MILP. Thus, there is an indication that the difficulty of the NPP relies on the number of tolled arcs $|\mathcal{A}_1|$. We denote this truncated vector $x_{\mathcal{A}_1} \in \{0,1\}^{|\mathcal{A}_1|}$. Similarly, we denote the truncated vector for the toll-free arcs $x_{\mathcal{A}_2} \in \{0,1\}^{|\mathcal{A}_2|}$. Once we fix $x_{\mathcal{A}_1}$, then the remaining program becomes a linear program, which can be solved in polynomial time [36]. As mentioned before, the single-commodity NPP is NP-hard [51], therefore, the number of values of $x_{\mathcal{A}_1}$ that we need to enumerate cannot be polynomial (assuming P $\neq$ NP).

Note that after $x_{\mathcal{A}_1}$ is fixed, we can decompose Program (3.1) into two steps: (i) Filling x with toll-free arcs $x_{\mathcal{A}_2}$ to form the shortest path; (ii) Finding the toll prices t that make x optimal. If either step is infeasible, then $x_{\mathcal{A}_1}$ is infeasible in Program (3.1). In step (i), the prices t are not required because the costs of the toll-free arcs do not depend on t. The problem of finding $x_{\mathcal{A}_2}$ becomes a minimum-cost flow problem with multiple sources and sinks, which can be solved by the following linear program:

$$\min_{x_{\mathcal{A}_2}} \{(c^\top x)_{\mathcal{A}_2} \mid (Ax)_{\mathcal{A}_2} = b - (Ax)_{\mathcal{A}_1}, x_{\mathcal{A}_2} \geq 0\}.$$

Thus, once $x_{\mathcal{A}_1}$ is chosen, we know $x_{\mathcal{A}_2}$ immediately, and a path x emerges without the need of knowing t. It does not matter if there are multiple $x_{\mathcal{A}_2}$ for a single $x_{\mathcal{A}_1}$, since $x_{\mathcal{A}_2}$ does not affect the leader's revenue. Having x, the toll prices t can be computed in step (ii) with

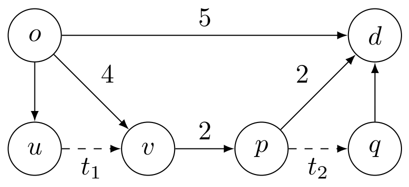


Figure 3.1 – Graph for Example 3.2.

another linear program:

$$\max_{t,y}\{x^\top t \mid t \in \mathcal{T}, A^\top y - t \leq c, b^\top y - x^\top t = c^\top x\}.$$

From this perspective, the single-commodity NPP is a combinatorial problem, which is less about the choice of the continuous prices, but more about the choice of the discrete paths (more specifically, the choice of $x_{\mathcal{A}_1}$). This observation will become more apparent once we introduce the conjugate model in Section 3.2.4.

Example 3.2. *Consider a single-commodity NPP with the graph shown in Figure 3.1. Dashed arcs are tolled arcs while solid arcs are toll-free. The numbers in the middle of the arcs represent their costs (arcs without numbers have no costs). In this graph, there are only two tolled arcs, hence we can compute the optimal solution by enumerating all four combinations of $x_{\mathcal{A}_1} = (x_1, x_2)$ (x_a corresponds to $t_a, a \in \{1, 2\}$):*

- *For $x_{\mathcal{A}_1} = (0, 0)$, the shortest path is $(o - d)$ with cost 5. The leader receives no revenue.*
- *For $x_{\mathcal{A}_1} = (1, 0)$, the shortest path is $(o - u - v - p - d)$ with cost $4 + t_1$. Solving for t in step (ii) gives us $t_1 = 1, t_2 \to \infty$. Thus, the leader's revenue is 1.*
- *For $x_{\mathcal{A}_1} = (0, 1)$, the shortest path is $(o - v - p - q - d)$ with cost $6 + t_2$. However, the linear program in step (ii) is infeasible.*
- *For $x_{\mathcal{A}_1} = (1, 1)$, the shortest path is $(o - u - v - p - q - d)$ with cost $2 + t_1 + t_2$. The revenue is 3 with $t_1 = 3, t_2 = 0$ (there are multiple solutions for t, all of them produce the same revenue).*

Choosing the highest revenue from all four cases, we obtain the optimal value of 3, corresponding to $x_{\mathcal{A}_1} = (1, 1)$ and the path $(o - u - v - p - q - d)$. Two remarks are in order from this example. First, in the case with $x_{\mathcal{A}_1} = (1, 0)$, the tolled arc 2 is not selected ($x_2 = 0$). Thus, this arc can be removed from the graph. We effectively achieved this by setting t_2 to a very large number (basically to infinity). Second, in the case with $x_{\mathcal{A}_1} = (0, 1)$, the reason why step (ii) is infeasible is that the cost of this case ($6 + t_2$) is already larger than the cost of the first case (which is 5). Hence, no value $t_2 \geq 0$ can make this path optimal. We call such a path bilevel infeasible. *Bilevel feasibility is the topic of discussion in Section 3.3.1.*

3.2.2. Single-Tolled-Arc Case

We switch the focus on the other extreme case of the NPP, where there is a single tolled arc. To stay away from triviality, we assume that there are several followers $k \in \mathcal{K}$, each has its own origin o^k and destination d^k and optimizes its own shortest path x^k across the network. The leader aims to maximize the sum of revenues from all followers. The decision process of the leader can be thought of as the balancing act between two strategies: sell high to a few, and sell low to everyone. We consider that every follower has the same demand $\eta^k = 1$.[1]

Following the process described in Section 3.2.1, the single-level reformulation of the single-tolled-arc NPP is:

$$\max_{t,x^k,y^k} \sum_{k \in \mathcal{K}} t^\top x^k \tag{3.2a}$$

$$\text{s.t. } t \in \mathcal{T}, \tag{3.2b}$$

$$Ax^k = b^k, \qquad k \in \mathcal{K}, \tag{3.2c}$$

$$A^\top y^k \leq c + t, \qquad k \in \mathcal{K}, \tag{3.2d}$$

$$(c+t)^\top x^k = (b^k)^\top y^k, \qquad k \in \mathcal{K}, \tag{3.2e}$$

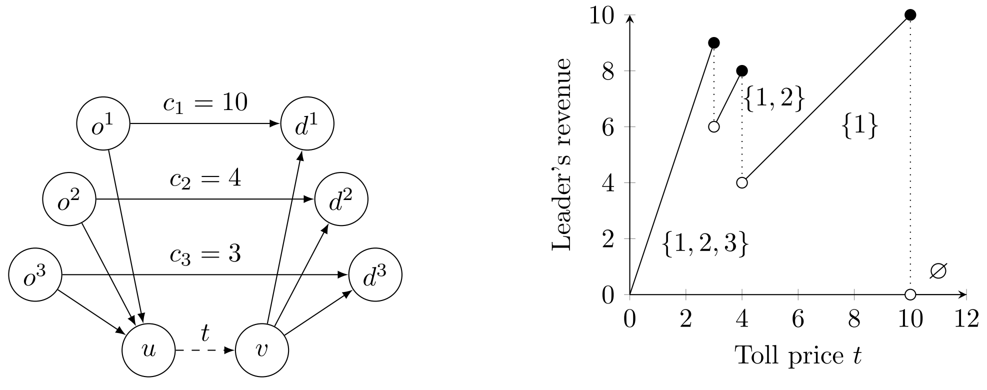
$$x^k \geq 0, \qquad k \in \mathcal{K}. \tag{3.2f}$$

If we linearize this program as done in Section 3.2.1, then the number of binary variables will be equal to the number of commodities $|\mathcal{K}|$, with one binary variable per commodity corresponding to the tolled arc in x^k. We set the index of this tolled arc to 0, and let $x_0 = (x_0^1, x_0^2, \ldots, x_0^{|\mathcal{K}|}) \in \{0,1\}^{|\mathcal{K}|}$ be the aggregated vector of the selections of the tolled arc from all commodities. Since the number of binary variables is proportional to $|\mathcal{K}|$, is the single-tolled-arc NPP an NP-hard problem as well?

It turns out that the single-tolled-arc NPP can be solved in polynomial time. Understanding the algorithm used to solve this variant is key to comprehend the asymmetry between $|\mathcal{A}_1|$ and $|\mathcal{K}|$. It is best to demonstrate this algorithm with an example.

Example 3.3. *Consider the single-tolled-arc NPP described in Figure 3.2a. In this example, we only have one tolled arc which is shared by three different followers, each travels from o^k to d^k for $k \in \mathcal{K} = \{1,2,3\}$. Each follower can choose to use its own toll-free path with different cost (10, 4, and 3) or the tolled arc with cost t. In fact, all instances of the single-tolled-arc NPP can be represented in this form. In Section 3.2.1, we already discussed that if*

1. All results presented in this paper can be extended to the non-unit demand case in a straightforward manner. See Appendix 3.A for details.



(a) Graph ***(b)*** Revenue plot

Figure 3.2 – Illustrations for Example 3.3.

$x^k_{\mathcal{A}_1}$ is determined, so is $x^k_{\mathcal{A}_2}$, and since $|\mathcal{A}_1| = 1$, for each follower, there are only two values of $x^k_{\mathcal{A}_1}$ to consider: $x^k_0 = 1$ and $x^k_0 = 0$.

*Therefore, follower k will only use the tolled arc when $t \le c_k$ and use the toll-free path when $t > c_k$. Now, we enumerate all eight combinations of $x_0 = (x^1_0, x^2_0, x^3_0) \in \{0,1\}^3$. A quick observation tells us that we can never have $x^1_0 = 0$ and $x^2_0 = 1$ at the same time, since this implies $t > 10$ and $t \le 4$ which is a contradiction. Generally, if $c_{k_1} \ge c_{k_2}$ for some $k_1, k_2 \in \mathcal{K}$, then the combination $x^{k_1}_0 = 0$ and $x^{k_2}_0 = 1$ is infeasible. This property actually eliminates most of the combinations except for exactly $|\mathcal{K}| + 1 = 4$ combinations. Each combination corresponds to an interval of the price t: $[0,3]$, $(3,4]$, $(4,10]$, and $(10,\infty)$. These intervals are illustrated in the revenue plot in Figure 3.2b. The label of each interval denotes the set of commodities that use the tolled arc (*e.g. *if $t \in (3,4]$, then the tolled arc is used by followers 1 and 2).*

From the revenue plot, we can conclude that the optimal revenue is 10, the optimal solution is $t = 10$ and $(x^1_0, x^2_0, x^3_0) = (1,0,0)$. A remark regarding the four combinations is the following: each combination is characterized solely by the number of commodities that use the tolled arc, which happens to be 3, 2, 1, and 0 in this example (this explains the number $|\mathcal{K}| + 1$ above). This is the key difference between the single-commodity case and the single-tolled-arc case. In the single-commodity NPP, all (or most) combinations of $x_{\mathcal{A}_1}$ matter, whose number is $2^{|\mathcal{A}_1|}$ in total. In the single-tolled-arc NPP, only the number of commodities that use the tolled arc matters, so only $|\mathcal{K}| + 1$ cases need to be considered.

After having our intuition established, we proceed to formalize this process in Algorithm 3.1. The first part of the algorithm (lines 1-4) calculates the maximum price that follower k will remain using the tolled arc. If $t \le c_k$, then the follower will use the tolled arc and vice-versa. In some cases, $c_k = 0$ due to $\overline{c}_k = \underline{c}_k$, then the tolled path is never as good as the toll-free path and no values of $t \geq 0$ will persuade this follower to use the tolled arc (another example of *bilevel infeasibility*). Next, we sort all c_k (line 5) so that when $t = c_{k_1}$, then only k_1 uses the tolled arc, when $t = c_{k_2}$, then k_1 and k_2 use the tolled arc, and so on. Then, we enumerate all cases (lines 6-9) according to the number of commodities that use the tolled arc, denoted as w (line 7). The revenue r_w of the case w is the product of the number of commodities w and the maximum price c_{k_w} such that w commodities still use the tolled arc (line 8). Finally, we collect the maximum of all r_w (line 9) and return it (line 10). Note that in Algorithm 3.1, we skip the case $w = 0$ since it does not produce any revenue, but theoretically, this case still exists and it will become relevant in the general case.

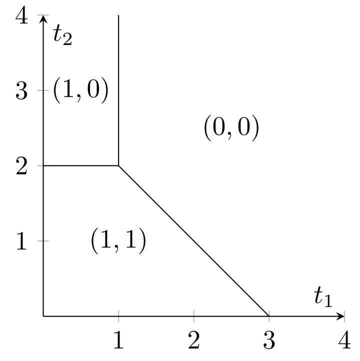
Algorithm 3.1 Single-tolled-arc NPP in polynomial time [39]

```
Input: The graph $G = (\mathcal{V}, \mathcal{A})$ with a single tolled arc, set of commodities $\mathcal{K}$ with their O-D
       pairs $o^k, d^k$.
Output: The optimal leader's revenue.
 1: for all $k \in \mathcal{K}$ do
 2:     $\overline{c}_k \leftarrow$ cost of the shortest path when $t \to \infty$
 3:     $\underline{c}_k \leftarrow$ cost of the shortest path when $t = 0$
 4:     $c_k \leftarrow \overline{c}_k - \underline{c}_k$
 5: Sort $c_k$ in descending order $c_{k_1}, c_{k_2}, \ldots, c_{k_{|\mathcal{K}|}}$
 6: $R \leftarrow 0$                                            ▷ Maximum revenue
 7: for $w \leftarrow 1, 2, \ldots, |\mathcal{K}|$ do
 8:     $r_w = w c_{k_w}$
 9:     $R \leftarrow \max\{R, r_w\}$
10: return $R$
```

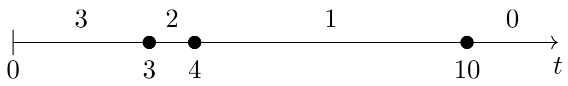
3.2.3. Reaction Plot

In the remaining of Section 3.2, we will consider the general case where there are multiple commodities and multiple tolled arcs. Because the single-commodity case can be reduced to the general case, the general case is also an NP-hard problem. Extending Program (3.1) to the general case results in a number of binary variables equal to $|\mathcal{A}_1||K|$, consisting of x_a^k for each $a \in \mathcal{A}_1$ and $k \in \mathcal{K}$. However, as shown in Section 3.2.2, this number is not a good indicator for the complexity.

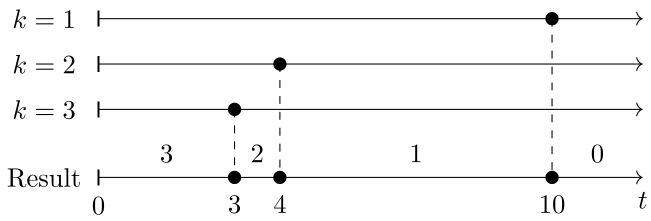
To extend the result presented in Section 3.2.2, we employ an illustration tool called *reaction plot*. In the reaction plot, the price of each tolled arc is represented by an axis (n tolled arcs means n axes). Then, for each value of t, we plot and group the reaction of the followers given t, *i.e.* the shortest path in the NPP. The result is a partition of the values



(a) Reaction plot of Example 3.2



(b) Reaction plot of Example 3.3



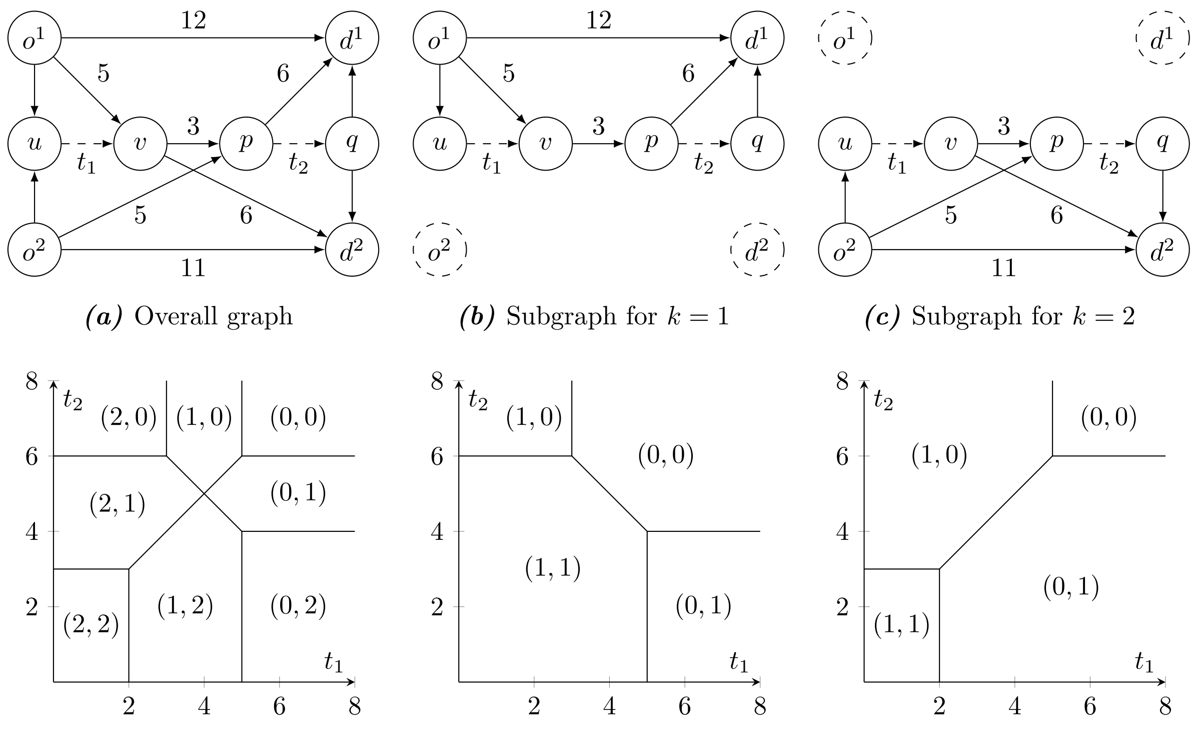
(c) Composition of the reaction plot in Example 3.3

Figure 3.3 – Examples of reaction plots.

of t into separate sections corresponding to different reactions of the followers. The reaction plot of Example 3.3 is shown in Figure 3.3b. The axis represents the price t of the tolled arc. The numbers below the axis are the meeting points of the intervals (which are c_k), and the numbers above the axis are the number of commodities that use the tolled arc for each interval (which is w in Algorithm 3.1). The reaction plot of Example 3.2, displayed in Figure 3.3a, is a 2-dimensional plot instead, since there are two tolled arcs in this case. The label of each region is $x_{\mathcal{A}_1}$. Note that there is no region for $x_{\mathcal{A}_1} = (0, 1)$ since, as we remarked in Section 3.2.1, there is no value of $t \geq 0$ that makes $x_{\mathcal{A}_1} = (0, 1)$ an optimal follower's solution (bilevel infeasible).

We have explored the reaction plots of the single-commodity and the single-tolled-arc case. Now, the question is: How do we draw the reaction plot of the general case? One could solve the followers' problems for each value of t, then group the reactions as we did in Figures 3.3a and 3.3b. Instead, we will compose the reaction plot of the general case from the plots of multiple single-commodity subproblems, one subproblem per commodity. Consider the reaction plot in Figure 3.3b. This plot is actually a composition of three individual reaction plots, superimposed upon each other as described in Figure 3.3c. The reaction plot of a single commodity $k \in \mathcal{K}$ in this case contains only two intervals: $[0, c_k]$ and (c_k, ∞) (corresponding to $x_0^k = 1$ and $x_0^k = 0$, respectively). We are allowed to stack up the reaction plots because once t is set, each follower solves its own follower's problem separately, thus their reactions are independent. This principle still applies if we have multiple tolled arcs, as shown in the next example.

Example 3.4. *Consider a multi-commodity NPP with graph in Figure 3.4a. There are two commodities travelling from o^k to d^k, $k \in \{1, 2\}$. There are also two tolled arcs with prices t_1 and t_2, shared by both commodities. With respect to each follower, only half the graph is relevant. Figures 3.4b and 3.4c are the subgraphs in the perspectives of followers 1 and 2,*



(a) Overall graph ***(b)*** Subgraph for $k = 1$ ***(c)*** Subgraph for $k = 2$

(d) Composed reaction plot ***(e)*** Reaction plot for $k = 1$ ***(f)*** Reaction plot for $k = 2$

Figure 3.4 – Composition of the reaction plot in Example 3.4.

respectively. Irrelevant nodes are drawn with dashes border while irrelevant arcs are hidden. The reaction plots of individual commodities are displayed in Figures 3.4e and 3.4f. By stacking Figure 3.4e on top of Figure 3.4f, we have the overall reaction plot in Figure 3.4d. Although each individual reaction plot has four regions, the composed plot only has eight regions in total rather than $4 \times 4 = 16$.

The labels of the regions in Figure 3.4d are in the format (w_1, w_2), *where* $w_1 = x_1^1 + x_1^2$ *is the number of commodities that uses the first tolled arc, while* $w_2 = x_2^1 + x_2^2$ *is the same but for the second tolled arc. Similar to the single-tolled-arc case in Section 3.2.2, when stacking the reaction plots, some combinations are prohibited due to the geometry of the individual plots. For example, we only have* $x_{\mathcal{A}_1}^1 = (1, 1)$ *in combination with* $x_{\mathcal{A}_1}^2 = (0, 1)$, *but not* $x_{\mathcal{A}_1}^1 = (0, 1)$ *and* $x_{\mathcal{A}_1}^2 = (1, 1)$. *Thus, the label* $(1, 2)$ *in Figure 3.4d always indicates the former combination, not the latter. Using this labeling system, intuitively, we can only have a maximum of* $(|\mathcal{K}| + 1)^{|\mathcal{A}_1|}$ *different regions in the reaction plot. The maximum number of regions in this particular example is* $3^2 = 9$, *but we can have less,* e.g. *in Figure 3.4d, we only have eight regions (the label* $(1, 1)$ *is missing, to which we will come back in Example 3.6).*

A side note is that the graph in this example is specifically designed as a showcase of two different structures in the case of two tolled arcs. The graph in Figure 3.4b has the series *structure, whose reaction plot (Figure 3.4e) contains a diagonal line from the top-left corner to the bottom-right. On the other hand, Figure 3.4c represents the* parallel *structure (remove the arc $v-p$ for more clarity), whose reaction plot (Figure 3.4f) also has a diagonal line but in the other direction (bottom-left to top-right). The overall graph demonstrates the complex nature of the NPP, where a pair of tolled arcs can be in series for one commodity, while in parallel for another.*

3.2.4. Conjugate Model

The labeling system $(w_1, w_2, \dots, w_{|\mathcal{A}_1|})$ from Example 3.4 gives us a hint: the number of discrete cases that we need to consider is only $(|\mathcal{K}|+1)^{|\mathcal{A}_1|}$, rather than $2^{|\mathcal{A}_1||K|}$ different combinations of the binary variables x_a^k. The next step is to formalize our intuition and prove Theorem 3.1. For this task, we developed a new reformulation of the NPP called the *conjugate model*.

We start with the bilevel program of the multi-commodity case:

$$\max_{t,x^k} \Big\{ \sum_{k\in\mathcal{K}} t^\top x^k \mid t \in \mathcal{T}, x^k \in \mathbf{R}^k(t) \quad \forall k \in \mathcal{K} \Big\}, \tag{3.3}$$

where $\mathbf{R}^k(t)$ is the reaction set of follower $k \in \mathcal{K}$:

$$\mathbf{R}^k(t) = \operatorname*{argmin}_{x^k} \Big\{ (c+t)^\top x^k \mid Ax^k = b^k, x^k \geq 0 \Big\}. \tag{3.4}$$

Once t is set by the leader, all followers' problems are independent. Thus, we can combine all of these problems into a single aggregated problem:

$$\overline{\mathbf{R}}(t) = \operatorname*{argmin}_{x^k} \Big\{ \sum_{k\in\mathcal{K}} (c+t)^\top x^k \mid Ax^k = b^k \quad \forall k \in \mathcal{K}, x^k \geq 0 \quad \forall k \in \mathcal{K} \Big\}.$$

The aggregated reaction set $\overline{\mathbf{R}}(t)$ is equivalent to the Cartesian product of all the individual reaction sets: $\overline{\mathbf{R}}(t) = \prod_{k\in\mathcal{K}} \mathbf{R}^k(t)$.

Next, we introduce a new variable $w = \sum_{k\in\mathcal{K}} x^k_{\mathcal{A}_1}$. The vector w has the same meaning as $(w_1, w_2, \dots, w_{|\mathcal{A}_1|})$ used in the labeling system, where w_a is the number of commodities that use the arc $a \in \mathcal{A}_1$. However, we do not replace $\sum_{k\in\mathcal{K}} x^k_{\mathcal{A}_1}$ by w with an equality, but rather with an inequality:

$$\min_{w,x^k} \sum_{k\in\mathcal{K}} \left(c^\top x^k\right) + t^\top w \tag{3.5a}$$

$$\text{s.t.} \sum_{k\in\mathcal{K}} x^k_{\mathcal{A}_1} \le w, \tag{3.5b}$$

$$Ax^k = b^k, \qquad k \in \mathcal{K}, \tag{3.5c}$$

$$w \ge 0, \tag{3.5d}$$

$$x^k \ge 0, \qquad k \in \mathcal{K}. \tag{3.5e}$$

As an abuse of notation, we used $t = t_{\mathcal{A}_1}$ in the objective function (3.5a), which is the truncated version of the original t (the version of t is implied by the context). Note that the objective function minimizes $t^\top w$, hence the inequality (3.5b) is always active whenever $t > 0$. The dual of Program (3.5) is:

$$\max_{y^k}\left\{\sum_{k\in\mathcal{K}} (b^k)^\top y^k \mid A^\top y^k \le c + t \quad \forall k \in \mathcal{K}\right\}. \tag{3.6}$$

Given $w \ge 0$, we introduce the *conjugate follower formulation*, defined as:

$$\max_{t,y^k}\left\{\sum_{k\in\mathcal{K}} \left((b^k)^\top y^k\right) - w^\top t \mid A^\top y^k - t \le c \quad \forall k \in \mathcal{K}, t \ge 0\right\}. \tag{3.7}$$

Program (3.7) is similar to Program (3.6), except for the following changes: (i) Program (3.7) is parameterized by w, while Program (3.6) is parameterized by t; (ii) The toll prices t become variables, and $t \ge 0$ is added as a constraint; (iii) The term $-w^\top t$ is added to the objective function of Program (3.7).

The dual of Program (3.7) is:

$$\min_{x^k}\left\{\sum_{k\in\mathcal{K}} c^\top x^k \mid \sum_{k\in\mathcal{K}} x^k_{\mathcal{A}_1} \le w, Ax^k = b^k \quad \forall k \in \mathcal{K}, x^k \ge 0 \quad \forall k \in \mathcal{K}\right\}. \tag{3.8}$$

Let $\mathbf{T}(w)$ be the set of t that are optimal to Program (3.7) given $w \ge 0$. We define the *conjugate bilevel formulation* as:

$$\max_{w,t}\{w^\top t \mid w \ge 0, t \in \mathbf{T}(w)\}. \tag{3.9}$$

Proposition 3.5. *Programs (3.3) and (3.9) are equivalent, in the sense that their optimal objective values are equal.*

Proof. We use the KKT conditions together with strong duality on Programs (3.5) and (3.6) to convert Program (3.3) to the following single-level reformulation:

$$\max_{t,w,x,y} \; t^\top w \tag{3.10a}$$

$$\text{s.t.} \; \sum_{k\in\mathcal{K}} x^k_{\mathcal{A}_1} \le w, \tag{3.10b}$$

$$Ax^k = b^k, \qquad k \in \mathcal{K}, \tag{3.10c}$$

$$A^\top y^k \le c + t, \qquad k \in \mathcal{K}, \tag{3.10d}$$

$$\sum_{k\in\mathcal{K}} \left(c^\top x^k\right) + t^\top w = \sum_{k\in\mathcal{K}} \left(b^k\right)^\top y^k, \tag{3.10e}$$

$$t \ge 0, \tag{3.10f}$$

$$w \ge 0, \tag{3.10g}$$

$$x^k \ge 0, \qquad k \in \mathcal{K}. \tag{3.10h}$$

Applying the same technique to Programs (3.7) and (3.8) produces the same single-level reformulation for Program (3.9). Therefore, the optimal objective values of Programs (3.3) and (3.9) are the same. □

The *conjugate model* provides a different perspective to NPP, where the leader controls w in contrast to t in the original model. Looking at Program (3.8), the role of w is that of the capacities of the tolled arcs. Program (3.8) is similar to the shortest path problem, but with capacities limited to w. Note that this program is not totally unimodular, thus there is no guarantee that x^k will be binary even if w is integral.

Hereafter, we refer to $\mathbf{T}(w)$ as the *action set* and reserve the term *reaction set* for $\mathbf{R}(t)$. Similarly, we call t the *action*, x the *reaction*, and w the *reduced reaction.*

Proof of Theorem 3.1. Proposition 3.5 implies that we can use Program (3.9) to find the optimal revenue of the NPP. For each commodity $k \in \mathcal{K}$, we know that there is always an optimal reaction x^k that is binary (since the followers' problems are shortest path problems), and since $w = \sum_{k\in\mathcal{K}} x^k_{\mathcal{A}_1}$, it is sufficient to enumerate w in the set $\{0, 1, \ldots, |\mathcal{K}|\}^{|\mathcal{A}_1|}$. For each w, we solve Program (3.7) in polynomial time to obtain t, which can be multiplied with w to compute the revenue. The total number of cases that we need to enumerate is $(|\mathcal{K}| + 1)^{|\mathcal{A}_1|}$. Thus, if the number of tolled arcs $|\mathcal{A}_1|$ is fixed, this process solves the multi-commodity NPP in polynomial time with respect to the number of commodities $|\mathcal{K}|$. □

Although we can solve the NPP in polynomial time given that $|\mathcal{A}_1|$ is fixed, the approach described in the proof of Theorem 3.1 is not of practical interest. For $|\mathcal{A}_1| = 2$, the complexity of the algorithm is in $O(|\mathcal{K}|^2)$, for $|\mathcal{A}_1| = 10$, it is in $O(|\mathcal{K}|^{10})$, and so on. What Theorem 3.1

Table 3.1 – Enumeration of all cases in Example 3.4.

		Actions		Reactions				Revenue
w_1	w_2	t_1	t_2	x_1^1	x_2^1	x_1^2	x_2^2	$w^\top t$
0	0	∞	∞	0	0	0	0	0
1	0	5	∞	0	0	1	0	5
2	0	3	∞	1	0	1	0	6
0	1	∞	6	0	0	0	1	6
1	1	4	5	0.5	0.5	0.5	0.5	9
2	1	4	5	1	1	1	0	13
0	2	∞	4	0	1	0	1	8
1	**2**	**4**	**5**	**1**	**1**	**0**	**1**	**14**
2	2	2	3	1	1	1	1	10

emphasizes is that there exists an asymmetry in the complexity, which we can exploit to tackle problems with a high number of commodities. This will be attempted in the next section, with the help of the conjugate model.

Example 3.6. *In this example, we compute the optimal revenue of the NPP described in Example 3.4 using the algorithm in the proof of Theorem 3.1. Since $|\mathcal{K}| = 2$ and $|\mathcal{A}_1| = 2$, we have nine possible values for w, which are listed in Table 3.1. The optimal solution is highlighted in bold. The optimal revenue is 14, corresponding to $(w_1, w_2) = (1, 2)$ and $(t_1, t_2) = (4, 5)$. The case $w = (1, 1)$ is special, since Program (3.8) returns a fractional x_a^k reaction instead of a binary result. Using Figure 3.4d as a reference, this case is not represented by a full-dimensional region in the reaction plot, but by just a single point at $t = (4, 5)$. We call this degenerate case* weakly bilevel feasible*, as opposed to the other eight cases which are* strongly bilevel feasible*. Note that weakly bilevel feasible cases still appear in the reaction plot, but they do not manifest into full-dimensional regions. This contrasts with bilevel infeasible cases which are completely absent from the reaction plot. Weak and strong bilevel feasibility are discussed in Section 3.3.3.*

3.3. Strong Bilevel Feasibility

Section 3.2 tells us that the number of reduced reactions w that we need to enumerate is $(|\mathcal{K}| + 1)^{|\mathcal{A}_1|}$. This number is far smaller than the number of all compositions of integral values of $x_{\mathcal{A}_1}^k$ which is $2^{|\mathcal{A}_1||\mathcal{K}|}$. On the one hand, given a composition of $x_{\mathcal{A}_1}^k$, we can associate a value of w which is $w = \sum_{k\in\mathcal{K}} x_{\mathcal{A}_1}^k$. On the other hand, given a value of w, we may associate many compositions of $x^k_{\mathcal{A}_1}$. This raises several questions: How do we map $(|\mathcal{K}|+1)^{|\mathcal{A}_1|}$ values of w back into $2^{|\mathcal{A}_1||\mathcal{K}|}$ compositions of $x^k_{\mathcal{A}_1}$? Can all compositions of $x^k_{\mathcal{A}_1}$ be mapped from w, or are there some compositions that we can eliminate? What are the properties that a composition needs to satisfy to not be eliminated and how do we test them?

To answer these questions, we further explore the notion of the reduced reaction w, discuss the duality between t and w, from which we derive the concept of *strong bilevel feasibility*. Strong bilevel feasibility is the property that will eliminate most of $x^k_{\mathcal{A}_1}$ and bring the number $2^{|\mathcal{A}_1||\mathcal{K}|}$ down to $(|\mathcal{K}|+1)^{|\mathcal{A}_1|}$. We return to the single-commodity case to define three concepts: *bilevel feasibility* (Section 3.3.1), *action-reaction duality* (Section 3.3.2), and *strong bilevel feasibility* (Section 3.3.3). Finally, we extend these concepts to the multi-commodity case via the composition of individual commodities (Section 3.3.4).

3.3.1. Bilevel Feasibility

In this and the next two sections, we only consider the single-commodity case. Denote $\mathcal{X}$ the feasible set of the follower's problem. Note that $\mathcal{X}$ is independent from t. Let x be a point in $\mathcal{X}$. For each $t \geq 0$, the cost of x is $c^\top x + t^\top x_{\mathcal{A}_1}$. We call $g = c^\top x$ the *base cost*, and $w = x_{\mathcal{A}_1}$ the *reduced reaction*; given x, we know directly its mapping into the variable w and hence, in those contexts, we abuse notation, and we use w equal to $x_{\mathcal{A}_1}$. If two points in $\mathcal{X}$ share the same base cost and reduced reaction, they are equivalent in the NPP, because they produce the same follower's cost and leader's revenue regardless of t. Let $\mathcal{X}^* = \{(c^\top x, x_{\mathcal{A}_1}) \mid x \in \mathcal{X}\}$ be the set of all (g, w) pairs that are feasible. Then, each pair of $(g, w) \in \mathcal{X}^*$ represents an equivalence class of reactions.

Let $f(t)$ be the function of the follower's cost, specifically,

$$f(t) = \min_x \Big\{ (c+t)^\top x \mid Ax = b,\ x \geq 0 \Big\}.$$

We remark that $f(t)$ coincides with the optimal value of Program (3.5) restricted to a single commodity. For $t \not\geq 0$, assign $f(t) = -\infty$. It is well-known that f is a concave function. For the ease of analysis, we investigate $-f$ instead of f due to its convexity. For $t \geq 0$, $-f(t)$ is the maximum of $-g - t^\top w$ for all pairs of $(g, w) \in \mathcal{X}^*$. Another perspective is to say that the epigraph $\operatorname{epi}(-f)$ is the intersection of all the closed half-spaces of the form:

$$H(g, w) = \{(t, z) \mid z \geq -g - t^\top w\} \tag{3.11}$$

where $(g, w) \in \mathcal{X}^*$. Also denote $H_=(g, w)$ the hyperplane corresponding to $H(g, w)$. By construction, a point $x \in \mathcal{X}$ is optimal in the follower's problem for some t if and only if the hyperplane $H_=(c^\top x, x_{\mathcal{A}_1})$ supports $\operatorname{epi}(-f)$. We call such point *bilevel feasible*.

Definition 3.7. *A reaction x is* bilevel feasible *when $H_=(c^\top x, x_{\mathcal{A}_1})$ supports* $\operatorname{epi}(-f)$. *Otherwise, it is* bilevel infeasible.

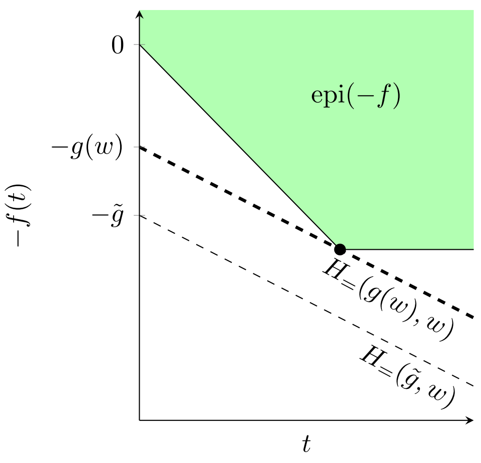


Figure 3.5 – $H_=(g(w), w)$ is the highest hyperplane given w.

Lemma 3.8 (Bui et al. [14]). *A reaction x is bilevel feasible if and only if there exists $t \geq 0$ such that x is optimal in the follower's problem,* i.e. $f(t) = c^\top x + t^\top x_{\mathcal{A}_1}$.

Proof.

$$\begin{aligned} & H_=(c^\top x, x_{\mathcal{A}_1}) \text{ supports epi}(-f) \\ \Longleftrightarrow\ & \text{There exists } (t, z) \text{ such that } z = -c^\top x - t^\top x_{\mathcal{A}_1} \text{ and } z = -f(t) \\ \Longleftrightarrow\ & -f(t) = -c^\top x - t^\top x_{\mathcal{A}_1}. \end{aligned}$$

□

Given w, in order for a hyperplane $H_=(g, w)$ to support epi$(-f)$, g must be the smallest possible value such that $(g, w) \in \mathcal{X}^*$, so that $H_=(g, w)$ is the highest of all hyperplanes with slope w. Thus, we can optimize g given w. Define

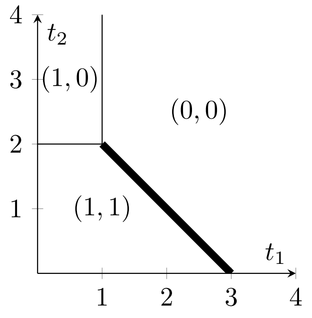
$$\begin{aligned} g(w) &= \sup_t \left\{ t^\top w - (-f(-t)) \right\} \\ &= \sup_t \left\{ f(t) - t^\top w \right\}. \end{aligned} \tag{3.12}$$

Equation (3.12) tells us that $g(w)$ is the smallest g such that $g \geq f(t) - t^\top w$ for all t, or equivalently, $-f(t) \geq -g - t^\top w$. Recall from Equation (3.11) that $H_=(g(w), w)$ supports epi$(-f)$ (see Figure 3.5). The function $g(w)$ is called the *convex conjugate* of $-f(-t)$ [52]. Since $-f$ is lower semi-continuous, by the Fenchel-Moreau theorem [52], the conjugate of the conjugate is the original function, so we also have:

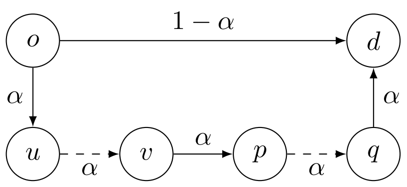
$$\begin{aligned} -f(-t) &= \sup_w \left\{ t^\top w - g(w) \right\} \\ \Leftrightarrow \qquad f(t) &= \inf_w \left\{ t^\top w + g(w) \right\}. \end{aligned} \tag{3.13}$$

Comparing Equation (3.12) to Programs (3.6) and (3.7), we can deduce that $g(w)$ is the optimal value of Program (3.7) restricted to a single commodity. In particular,

$$g(w) = \max_{t,y} \left\{ b^\top y - w^\top t \mid A^\top y - t \leq c,\ t \geq 0 \right\}.$$



(a) Reaction plot



(b) Value of x with $w = (\alpha, \alpha)$

Figure 3.6 – Illustrations for Example 3.10.

The same conclusion can be derived from Equation (3.13) and Programs (3.5) and (3.8). Therefore, the purpose of the conjugate follower model is to find $g(w)$, the smallest base cost given a reduced reaction. This leads us to a redefinition of bilevel feasibility:

Lemma 3.9. *A reaction x is* bilevel feasible *if and only if $c^\top x = g(x_{\mathcal{A}_1})$.*

We consider Lemmas 3.8 and 3.9 dual of each other, where one characterizes bilevel feasibility based on $f(t)$, and the other based on $g(w)$. Using the new definition, to test the bilevel feasibility of a particular $x \in \mathcal{X}$, we solve the Program (3.8) (or Program (3.7)) with w set to $x_{\mathcal{A}_1}$. If the optimal objective value $g(w)$ is equal to $c^\top x$, then it is bilevel feasible. Otherwise, it is bilevel infeasible.

Example 3.10. *We revisit Example 3.2 with the reaction plot redrawn in Figure 3.6a. We mentioned the case $x_{\mathcal{A}_1} = (0,1)$ as an example of bilevel infeasible reaction. Indeed, set $w = (0,1)$ and solve Program (3.7), we get $g(w) = 5$. The shortest path with $x_{\mathcal{A}_1} = (0,1)$ has a base cost of $6 > g(w)$, thus it is bilevel infeasible.*

In Example 3.2, we only listed four binary values of $x_{\mathcal{A}_1}$. It raises the question: Is there a bilevel feasible x such that $x_{\mathcal{A}_1}$ is non-binary? Solving Program (3.7) with $w = (0.5, 0.5)$ produces $g(w) = 3.5$. In general, if $w = (\alpha, \alpha)$ where $0 < \alpha < 1$, then $g(w) = 5(1-\alpha) + 2\alpha$. Program (3.8) gives us the reaction x described in Figure 3.6b. This is the convex combination of two simple paths corresponding to $x_{\mathcal{A}_1} = (1,1)$ and $x_{\mathcal{A}_1} = (0,0)$, respectively. By Lemma 3.8, if this x is bilevel feasible, then it must be optimal for some t. What is this set of t? Program (3.7) tells us that this reaction is optimal when t is in the intersection of the 2 regions labeled as $(1,1)$ and $(0,0)$ (the thick segment in Figure 3.6a). It seems that there is a relation between convex combinations of paths and intersections of regions in the reaction plot, which we will explore in the next two sections.

Bilevel feasibility is useful in the preprocessing step to prune away bilevel infeasible paths. Bilevel infeasible paths, by definition, are not optimal to the follower's problem for any $t \geq 0$, thus the follower never considers them regardless of the leader's decision. This process is demonstrated in Bui et al. [14], and we will reuse it in our experiments described in Section 3.4.

3.3.2. Action-Reaction Duality

The relationship between $f(t)$ and $g(w)$ is more than just finding the smallest base cost. Because $f(t)$ is a polyhedral function, so is $g(w)$, meaning their epigraphs can be defined as the intersection of a finite number of closed half-spaces. We will show that there is a bijection between the set of optimal t of $g(w)$ and the set of optimal w of $f(t)$. Furthermore, the dimensions of a pair of such sets sum up to $|\mathcal{A}_1|$. This relation is better illustrated with an example.

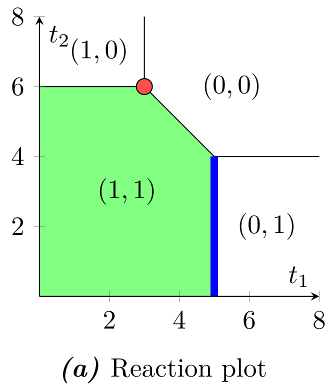
Example 3.11. *We reuse the graph in Figure 3.4b. In Section 3.2.3, we introduced the reaction plot as a tool to describe the optimal solution of Program (3.5) given t. The reaction plot of the graph in Figure 3.4b is redrawn in Figure 3.7a Another way to derive the reaction plot is through the projection of* $\operatorname{epi}(-f)$ *to the space of t (see Figure 3.8). Each full-dimensional region in the reaction plot corresponds to a facet of* $\operatorname{epi}(-f)$*, and its label is the (negative) gradient of $-f(t)$ when t is in the interior of the region. We can apply the same process to* $\operatorname{epi}(g)$ *as well, and the result is called the* action plot*, shown in Figure 3.7b. The labels in the action plot are $(t_1, \ldots, t_{|\mathcal{A}_1|})$ which are the gradients of $g(w)$, similar to the labels in the reaction plot.*

There is a correspondence between "features" (defined later) of one plot to those of the other. Every label representing a full-dimensional region in one plot corresponds to a vertex of the other plot (the red/green vertices and polygons). Edges separating two polygons in one plot become edges connecting the corresponding vertices in the other (the blue edges). As a general rule, the sum of the dimensions of a pair of features is always $|\mathcal{A}_1|$, which is 2 in this example.

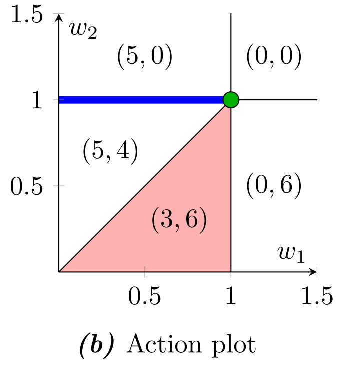
As remarked in Example 3.11, the reaction plot is the projection of $\operatorname{epi}(-f)$. Thus, the "features" (polygons, edges, vertices) in the reaction plot correspond to faces of $\operatorname{epi}(-f)$. Conversely, every non-vertical face of $\operatorname{epi}(-f)$ corresponds to a feature of the reaction plot with the same dimension. The same applies to faces of $\operatorname{epi}(g)$. In this section, we want to establish a bijection between faces of $\operatorname{epi}(-f)$ and faces of $\operatorname{epi}(g)$, along with their dimensional relation. Hereafter, let $n = |\mathcal{A}_1|$ be the dimension of the space of t (and of w).

Definition 3.12. *An* action set [2] *is the projection of a non-vertical face of* $\operatorname{epi}(-f)$ *to the space of t. To put it concretely, a set $T \subseteq \mathbb{R}^n$ is an action set when there exists a face F of* $\operatorname{epi}(-f)$ *such that:*

2. In Section 3.2, we call the set of t that are optimal to Program (3.7) action set. The term action set here has the same meaning as in Section 3.2, despite of their different definitions.



(a) Reaction plot



(b) Action plot

Figure 3.7 – Correspondence of features between the action plot and the reaction plot.

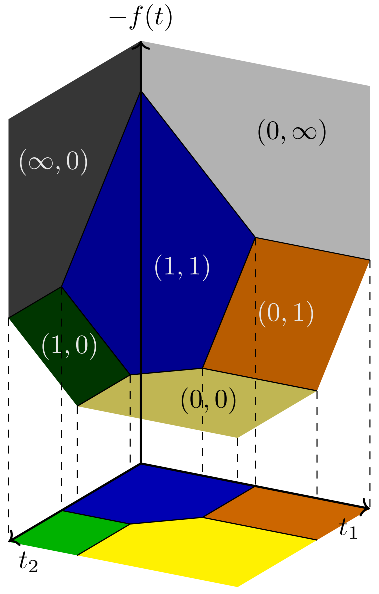


Figure 3.8 – Reaction plot as a projection of epi$(-f)$.

(i) The direction $(t, z) = (0, 1)$ *is not in* $\operatorname{aff}(F)$ *(F is non-vertical);*

(ii) $T = \{t \in \mathbb{R}^n \mid (t, z) \in F$ *for some* $z\}$ *(T is the projection of F).*

Condition (i) implies that given $t \in T$, there is only one value of z such that $(t, z) \in F$. Condition (ii) forces z to be $-f(t)$. Thus, given T, we can derive $F = \{(t, -f(t)) \in \mathbb{R}^n \times \mathbb{R} \mid t \in T\}$, making the mapping from F to T bijective. A *reaction set*[3] W is defined similarly with epi(g).

Lemma 3.13. *Given an action set T, if T' is a face of T, then T' is an action set.*

Proof. Let $F = \{(t, -f(t)) \in \mathbb{R}^n \times \mathbb{R} \mid t \in T\}$ be the face of epi$(-f)$ corresponding to T. It is easy to verify that $t \in \operatorname{ri}(T)$ if and only if $(t, -f(t)) \in \operatorname{ri}(F)$, where $\operatorname{ri}(\cdot)$ denotes the relative interior of a set. Since T' is a face of T, it is the intersection of T and the half-space $\{t \mid \alpha^\top t \leq \beta\}$ for some $\alpha \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$. Consider $F' = \{(t, -f(t)) \in F \mid \alpha^\top t \leq \beta\}$. Then, $t \in T'$ if and only if $(t, -f(t)) \in F'$, and since $T' \cap \operatorname{ri}(T) = \varnothing$, we have $F' \cap \operatorname{ri}(F) = \varnothing$. Thus, F' is a face of F which in turn is a non-vertical face of epi$(-f)$, meaning F' is a non-vertical face of epi$(-f)$. Also, T' is the projection of F', making it an action set. $\square$

Given $t \geq 0$, let $\mathbf{W}(t)$ be the set of w that are optimal to Program (3.5). Recall from Equation (3.13) that $w \in \mathbf{W}(t)$ if and only if

$$f(t) = t^\top w + g(w). \tag{3.14}$$

3. This also has the same meaning as in Section 3.2, but $\mathbf{R}(t)$ is for x, while W (and later $\mathbf{W}(t)$) is for w.

Recall the definition of $\mathbf{T}(w)$ in Section 3.2.4 as the set of t that are optimal to Program (3.7), we also obtain the equality above. Therefore, $w \in \mathbf{W}(t)$ if and only if $t \in \mathbf{T}(w)$.

Given $t \geq 0$, we rewrite Equation (3.13) as:

$$f(t) = \inf_{w,z} \left\{ t^\top w + z \mid z \geq g(w) \right\} = \inf_{(w,z)} \left\{ t^\top w + z \mid (w,z) \in \operatorname{epi}(g) \right\}.$$

By treating epi(g) as the feasible set of some optimization problem, the set of (w, z) that is optimal forms a non-vertical face of epi(g). As a result, $\mathbf{W}(t)$ is a reaction set. Conversely, given a reaction set W, there always exist $t \geq 0$ such that $W = \mathbf{W}(t)$, since every face is optimal for some objective function (which we can adjust with t). By a similar reasoning, $\mathbf{T}(w)$ is an action set for all $w \geq 0$.

Given an action set $T \subseteq \mathbb{R}^n$, define $\mathbf{W}(T) = \bigcap_{t\in T} \mathbf{W}(t)$.

Proposition 3.14. *If T is an action set, then $\mathbf{W}(T)$ is a reaction set.*

Proof. Define $\mathbf{G}(W) = \{(w, g(w)) \in \mathbb{R}^n \times \mathbb{R} \mid w \in W\}$ as the face of epi(g) corresponding to the reaction set W. Consider $G = \bigcap_{t\in T} \mathbf{G}(\mathbf{W}(t))$. Since all points in $\mathbf{G}(\mathbf{W}(t))$ are in the form $(w, g(w))$ for some $w \geq 0$, so are all points in G. Given w, all of the following statements are equivalent:

$$w \in \mathbf{W}(T) \Leftrightarrow w \in \mathbf{W}(t), \forall t \in T \Leftrightarrow (w,g(w)) \in \mathbf{G}(\mathbf{W}(t)), \forall t \in T \Leftrightarrow (w,g(w)) \in G.$$

Thus, $\mathbf{W}(T)$ is the projection of G onto the w-space. Because $\mathbf{G}(\mathbf{W}(t))$ are all non-vertical faces of epi(g), their intersection G is either empty or a non-vertical face of epi(g). Since T is an action set, there exists w such that $T = \mathbf{T}(w)$. This means that for all $t \in T = \mathbf{T}(w)$, $w \in \mathbf{W}(t)$, hence $w \in \mathbf{W}(T)$ and $\mathbf{W}(T)$ is non-empty. Therefore, G is non-empty and is a non-vertical face of epi(g), meaning that its projection $\mathbf{W}(T)$ is a reaction set. ☐

Given a reaction set W, define $\mathbf{T}(W)$ similarly. Then, $\mathbf{T}$ is the inverse mapping of $\mathbf{W}$.

Proposition 3.15. *Given an action set T, then $\mathbf{T}(\mathbf{W}(T)) = T$.*

Proof. For all $t \in T$, $w \in \mathbf{W}(T)$, since $\mathbf{W}(T) \subseteq \mathbf{W}(t)$, we have $w \in \mathbf{W}(t)$ implying $t \in \mathbf{T}(w)$. Hence, $t \in \bigcap_{w\in\mathbf{W}(T)} \mathbf{T}(w) = \mathbf{T}(\mathbf{W}(T))$, meaning $T \subseteq \mathbf{T}(\mathbf{W}(T))$.

Conversely, since T is an action set, there exists $w \geq 0$ such that $\mathbf{T}(w) = T$, hence $w \in \mathbf{W}(t)$ for all $t \in \mathbf{T}(w) = T$, implying $w \in \mathbf{W}(T)$. It follows that $\mathbf{T}(\mathbf{W}(T)) \subseteq \mathbf{T}(w) = T$ (by definition of $\mathbf{T}$). We conclude that $\mathbf{T}(\mathbf{W}(T)) = T$. ☐

Corollary 3.16. *Given two action sets T_1 and T_2, then $T_1 \subseteq T_2$ if and only if $\mathbf{W}(T_1) \supseteq \mathbf{W}(T_2)$.*

Proof. The forward direction is true due to the definition of $\mathbf{W}$. For the reverse direction, $\mathbf{W}(T_1) \supseteq \mathbf{W}(T_2)$ implies $\mathbf{T}(\mathbf{W}(T_1)) \subseteq \mathbf{T}(\mathbf{W}(T_2))$. By Proposition 3.15, $T_1 \subseteq T_2$. ☐

Together, $\mathbf{W}$ and $\mathbf{T}$ define a bijection between the action sets and the reaction sets. Given $t \geq 0$, the smallest action set containing t is $\mathbf{T}(\mathbf{W}(t))$. The same applies to all $w \geq 0$. This mapping can be extended to a bijection between non-vertical faces of $\operatorname{epi}(-f)$ and those of $\operatorname{epi}(g)$. The dimensional property linking them can be expressed as:

Theorem 3.17. *Given an action set T, it holds $\dim T + \dim \mathbf{W}(T) = n$.*

Proof. Since T is an action set, there exists w such that $T = \mathbf{T}(w)$, which implies $w \in \mathbf{W}(T)$. Let S be the subspace parallel to $\operatorname{aff}(T)$ and $S^\perp$ be the orthogonal subspace of S. Their dimensional relation is $\dim S^\perp = n - \dim S = n - \dim T$. Choose a basis of $S^\perp$ so that $S^\perp = \operatorname{span}\{u_1, u_2, \ldots, u_m\}$ where $m = \dim S^\perp$. For each $i = 1, \ldots, m$, we will prove that there exists $\delta_i \in \operatorname{span}\{u_i\}$, $\delta_i \neq 0$ such that $w + \delta_i \in \mathbf{W}(T)$.

Suppose that the above statement is false, meaning there exists i such that for all $\delta_i \in \operatorname{span}\{u_i\} \setminus \{0\}$, $w + \delta_i \notin \mathbf{W}(T)$. We can build a sequence $\delta_i^{(j)} \to 0$ in $\operatorname{span}\{u_i\} \setminus \{0\}$ such that $w + \delta_i^{(j)} \geq 0$ so that each $w + \delta_i^{(j)}$ is contained in some reaction set.[4] Since $\operatorname{epi}(g)$ is a polyhedron, it has a finite number of faces, hence a finite number of reaction sets.[5] Thus, there must be a reaction set W' that contains infinitely many terms of $w + \delta_i^{(j)}$. Reaction sets are closed (they are projections of faces), hence the limit $w = \lim(w + \delta_i^{(j)})$ is in W' as well, implying that $\mathbf{T}(w) = T$ contains $\mathbf{T}(W')$.

Because $u_i \in S^\perp$, ${u_i}^\top(t - t') = 0$ for all $t, t' \in T$. Choose $t \in T, t' \in \mathbf{T}(W')$, then for all $\delta_i \in \operatorname{span}\{u_i\} \setminus \{0\}$ (also recall Equation (3.14)):

$$\begin{aligned} & (w + \delta_i)^\top (t - t') = w^\top (t - t') = f(t) - f(t') \qquad \text{(since } t, t' \in \mathbf{T}(w)\text{)} \\ \Leftrightarrow \quad & f(t) - (w + \delta_i)^\top t = f(t') - (w + \delta_i)^\top t'. \end{aligned} \tag{3.15}$$

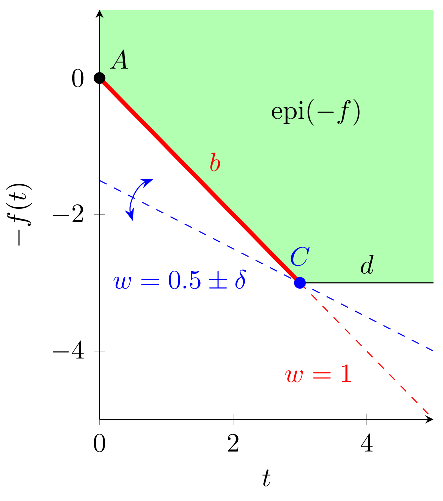
Notice that $t' \in \mathbf{T}(W')$ and $w + \delta_i^{(j)} \in W'$ for some j, hence $t' \in \mathbf{T}(w + \delta_i^{(j)})$ and $f(t') - (w + \delta_i^{(j)})^\top t' = g(w + \delta_i^{(j)})$ (by Equation (3.14)). Thus, the left-hand side of Equation (3.15) is also equal to $g(w + \delta_i^{(j)})$ which implies that $w + \delta_i^{(j)} \in \mathbf{W}(t)$ for all $t \in T$. It follows that $w + \delta_i^{(j)} \in \mathbf{W}(T)$ which is a contradiction.

To conclude the proof, since there exists $\delta_i \neq 0$ such that $w + \delta_i \in \mathbf{W}(T)$ in every direction $u_i \in S^\perp$, w along with all the $w + \delta_i$ form an affinely independent set in $\mathbf{W}(T)$, thus $\dim \mathbf{W}(T) \geq m$. For all $u \notin S^\perp$, there exists $t, t' \in T$ such that $u^\top(t - t') \neq 0$ which implies $w + u \notin \mathbf{W}(T)$. Therefore, $\dim \mathbf{W}(T) = m = n - \dim T$. ☐

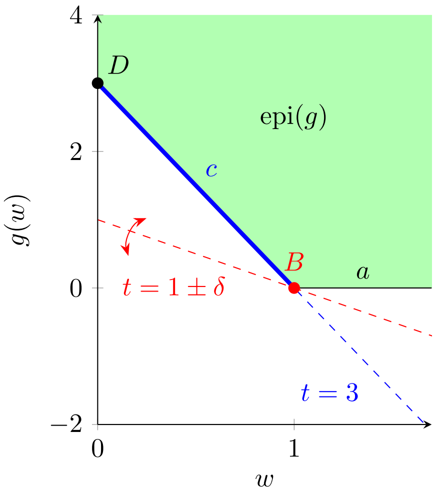
Example 3.18. *We consider the NPP described in Example 3.3, restricted to commodity $k = 3$. The plots of $-f(t)$ and $g(w)$ are shown in Figures 3.9a and 3.9b. Each non-vertical face*

4. If $w > 0$, then the sequence $w + \delta_i^{(j)} \geq 0$ is trivial to construct. If $w_a = 0$ for some $a \in \mathcal{A}_1$, then t_a for some $t \in T$ can get arbitrary large. Since $\delta_i \in S^\perp$, $\delta_{i,a} = 0$ and $w_a + \delta_{i,a} = 0$ for all i which do not violate the condition $w + \delta_i^{(j)} \geq 0$.

5. As suggested, this dimensional property does not hold when f and g are not polyhedral functions.



(a) Plot of $-f(t)$



(b) Plot of $g(w)$

Figure 3.9 – Plots for Example 3.18.

of epi$(-f)$ *(encoded as A, b, C, d) has a counterpart in* epi(g) *(encoded as a, B, c, D) through the mappings* $\mathbf{W}$ *and* $\mathbf{T}$*. Consider the 1-dimensional face b of* epi$(-f)$ *(the red segment in Figure 3.9a), which is the intersection of* epi$(-f)$ *and the hyperplane corresponding to* $(g, w) = (0, 1)$*. We can verify that* $g(1) = 0$ *(the red vertex in Figure 3.9b). Perturbing w by any small amount changes the optimal face from b into either A or C, hence B is a 0-dimensional face of* epi(g)*. In contrast, the 0-dimensional face C of* epi$(-f)$ *(the blue vertex in Figure 3.9a) corresponds to not only* $w = 0.5$*, but also any w in its neighborhood. Informally, we can "wiggle" the supporting hyperplane (the blue dashed line in Figure 3.9a) while keeping C as the optimal point, which we cannot do with b. As a consequence, c is a 1-dimensional face of* epi(g)*. In both cases, the sum of the dimensions is* $|\mathcal{A}_1| = 1$*.*

3.3.3. Strong Bilevel Feasibility

Example 3.10 shows that fractional reactions x can be bilevel feasible. However, we can write fractional reactions as convex combinations of simple paths (which are the extreme points of $\mathcal{X}$), hence if a fractional reaction is optimal to the leader's problem, then there must be an optimal simple path as well. Simple paths always have binary reduced reactions $w = x_{\mathcal{A}_1}$. Moreover, as remarked before, the labels $(w_1, \ldots, w_n)$ of full-dimensional regions in the reaction plot are always binary. From the discussion in Section 3.3.2, these regions are action sets, and full-dimensional action sets correspond to 0-dimensional reaction sets (Theorem 3.17), which, in turn, correspond to 0-dimensional faces of epi(g). Thus, these labels are the projections of extreme points of epi(g). In this section, we explore the relations between simple paths, binary w, and extreme points of $\mathcal{X}$ and epi(g).

Definition 3.19. *A reduced reaction w is* strongly bilevel feasible *when $\{w\}$ is a reaction set. Otherwise, it is* weakly bilevel feasible.

Lemma 3.20. *Given $w \geq 0$, the following statements are equivalent:*

(a) w is strongly bilevel feasible*;*

(b) $\{w\}$ is a reaction set;

(c) $\mathbf{W}(\mathbf{T}(w)) = \{w\}$

(d) $(w, g(w))$ *is an extreme point of* $\operatorname{epi}(g)$*;*

(e) $\dim \mathbf{T}(w) = n$.

Proof. Direct results of Definitions 3.12 and 3.19 and Theorem 3.17. □

Note that strong bilevel feasibility is a property of w, while bilevel feasibility is a property of x. It is not incidental that extreme points of $\mathcal{X}$ are related to extreme points of $\operatorname{epi}(g)$.

Proposition 3.21. *If w is strongly bilevel feasible, there exists an extreme point x of $\mathcal{X}$ such that x is bilevel feasible and satisfies $x_{\mathcal{A}_1} = w$.*

Proof. First, we prove that there exists a bilevel feasible x such that $x_{\mathcal{A}_1} = w$. We solve for $g(w)$ using Program (3.8) to obtain x. Suppose that $x_a < w_a$ for some $a \in \mathcal{A}_1$. By complementary slackness, all $t \in \mathbf{T}(w)$, which are optimal solutions of Program (3.7), must have $t_a = 0$. Consequently, $\dim \mathbf{T}(w) < n$. Thus, by Lemma 3.20, w is weakly bilevel feasible. Therefore, if w is strongly bilevel feasible, all optimal solutions of Program (3.8) must satisfy $x_{\mathcal{A}_1} = w$.

Now that we have x that is bilevel feasible and $x_{\mathcal{A}_1} = w$, we will construct an extreme point $\hat{x}$ of $\mathcal{X}$ that also satisfies these properties. Suppose that x is not an extreme point, we can write x as a strict combination of extreme points $x^{(i)}$ and extreme rays.[6] Choose any i, let $\hat{x} = x^{(i)}_{\mathcal{A}_1}$ and $\hat{w} = \hat{x}_{\mathcal{A}_1}$. For all $t \in \mathbf{T}(w)$, t makes x optimal to the follower's problem. Then, t must make all $x^{(i)}$ optimal (since the combination is strict), hence $\hat{x}$ is bilevel feasible and $t \in \mathbf{T}(\hat{w})$. Following the same reasoning in the first part of the proof, we conclude that $\hat{x}_{\mathcal{A}_1} = \hat{w} = w$. □

As a result, all strongly bilevel feasible w are binary since extreme points of $\mathcal{X}$ are simple paths and they are binary. Similar to the fact that there is an extreme point of $\mathcal{X}$ that is optimal to the follower's problem, there is an extreme point of $\operatorname{epi}(g)$ that is optimal to the leader's problem.

Theorem 3.22. *There exists a strongly bilevel feasible w that is optimal to the NPP,* i.e. *it solves Program (3.9).*

6. In the context of the shortest path problem, extreme points correspond to simple paths and extreme rays correspond to loops.

Proof. Let (w',t) be an optimal solution of Program (3.9). In order to be an optimal solution, first, $t \in \mathbf{T}(w')$, hence $w' \in \mathbf{W}(t)$. If w' is not an extreme point of $\mathbf{W}(t)$, we write w' as a strict combination of extreme points $w^{(i)}$ and extreme rays. Since $w^{(i)} \in \mathbf{W}(t)$, the pairs $(w^{(i)}, t)$ are also feasible to Program (3.9). Because ${w'}^\top t$ is the highest revenue to the NPP, so are $\left(w^{(i)}\right)^\top t$ for all i, thus $w^{(i)}$ are optimal to Program (3.9). Extreme points of $\mathbf{W}(t)$ are 0-dimensional faces of a reaction set, hence they are 0-dimensional reaction sets (Lemma 3.13). Therefore, $w^{(i)}$ are strongly bilevel feasible. $\square$

Proposition 3.21 and Theorem 3.22 imply that NPP can be solved by enumerating only simple paths. When we extend this concept to multi-commodity problems in Section 3.3.4.2, strong bilevel feasibility proves to be a much more powerful property than just a tool to eliminate fractional reactions.

3.3.4. Composition

In this section, we return to the multi-commodity case and generalize the concepts introduced in the previous sections via the process of composition. Given a commodity $k \in \mathcal{K}$, we add the superscript k to all notations that are related to this commodity (including $\mathcal{X}^k, x^k, w^k, f^k, g^k, \mathbf{T}^k, \mathbf{W}^k$). The objective function f of the aggregated follower's problem (Program (3.5)) is the sum of all f^k:

$$f(t) = \sum_{k \in \mathcal{K}} f^k(t).$$

The conjugate $g(w)$ is defined as in Equation (3.12). Since the conjugate operator of summation is infimal convolution [52], we also have:

$$g(w) = \inf \left\{ \sum_{k \in \mathcal{K}} g^k(w^k) \;\middle|\; \sum_{k \in \mathcal{K}} w^k = w \right\}. \tag{3.16}$$

Denote $\mathcal{X} = \mathcal{X}^1 \times \cdots \times \mathcal{X}^{|\mathcal{K}|}$ the feasible set of Program (3.5) and $(x^k) = (x^1, \ldots, x^{|\mathcal{K}|}) \in \mathcal{X}$ a feasible point of that program. Hereafter, we call tuples such as $(x^k) = (x^1, \ldots, x^{|\mathcal{K}|})$ and $(w^k) = (w^1, \ldots, w^{|\mathcal{K}|})$ *compositions* or *composed reactions*.

3.3.4.1. Bilevel Feasibility. With the new definitions of $f(t)$ and $g(w)$ for the multi-commodity case, Lemma 3.9 is straightforward to extend:

Definition 3.23. *A composed reaction* $(x^k) \in \mathcal{X}$ *is* bilevel feasible *when*

$$\sum_{k \in \mathcal{K}} c^\top x^k = g\left(\sum_{k \in \mathcal{K}} x^k_{\mathcal{A}_1} \right). \tag{3.17}$$

We can still use the conjugate model (Programs (3.7) and (3.8)) to test bilevel feasibility. The only change is to set $w = \sum_{k \in \mathcal{K}} x^k_{\mathcal{A}_1}$. This will be the common theme of this section.

Lemma 3.8 is stated similarly. What are more interesting are the conditions that it implies when we decompose f and g into f^k and g^k.

Proposition 3.24. *Given a composed reaction $(x^k) \in \mathcal{X}$, the following statements are equivalent:*

(a) (x^k) is bilevel feasible;

(b) Equation (3.17) is satisfied;

(c) Each x^k is bilevel feasible individually, and $g\left(\sum_{k\in\mathcal{K}} x^k_{\mathcal{A}_1}\right) = \sum_{k\in\mathcal{K}} g^k(x^k_{\mathcal{A}_1})$;

(d) There exists $t \geq 0$ such that $f(t) = \sum_{k\in\mathcal{K}} \left(c^\top x^k + t^\top x^k_{\mathcal{A}_1}\right)$;

(e) There exists $t \geq 0$ such that for all $k \in \mathcal{K}$, $f^k(t) = c^\top x^k + t^\top x^k_{\mathcal{A}_1}$.

Proof. (a) $\Leftrightarrow$ (b) is Definition 3.23. (a) $\Leftrightarrow$ (d) is Lemma 3.8 rewritten with the new f. (d) $\Leftarrow$ (e) is trivial. Since $f^k(t) \leq c^\top x^k + t^\top x^k_{\mathcal{A}_1}$ for all $k \in \mathcal{K}$, (d) $\Rightarrow$ (e).

(b) $\Rightarrow$ (c): Since $c^\top x^k \geq g^k(x^k_{\mathcal{A}_1})$, from Equation (3.16),

$$\sum_{k\in\mathcal{K}} c^\top x^k \geq \sum_{k\in\mathcal{K}} g^k(x^k_{\mathcal{A}_1}) \geq g\left(\sum_{k\in\mathcal{K}} x^k_{\mathcal{A}_1}\right). \tag{3.18}$$

If (b) is true, then the inequalities become equalities, and $c^\top x^k = g^k(x^k_{\mathcal{A}_1})$ means that x^k is bilevel feasible individually (Lemma 3.9). Thus, (c) is true.

(b) $\Leftarrow$ (c): Each x^k is bilevel feasible individually, hence $c^\top x^k = g^k(x^k)$. Then, all inequalities in Equation (3.18) are equalities which implies (b). ☐

Statements (c) and (e) are the decomposed forms of statements (b) and (d), respectively. Statements (b) and (d) are considered dual of each other, as well as statements (c) and (e). Statement (e) means that if (x^k) is bilevel feasible, then there exists $t \geq 0$ such that all x^k are simultaneously optimal to their own followers' problems. Statement (c) gives us an extra condition that a bilevel feasible (x^k) needs to satisfy even when all x^k are bilevel feasible individually. In what follows, let $w^k = x^k_{\mathcal{A}_1}$.

Definition 3.25. *A composition (w^k) is* bilevel feasible *when*

$$g\left(\sum_{k\in\mathcal{K}} w^k\right) = \sum_{k\in\mathcal{K}} g^k(w^k). \tag{3.19}$$

Then, statement (c) of Proposition 3.24 can be rewritten as: each x^k is bilevel feasible individually and the composition (w^k) is bilevel feasible where $w^k = x^k_{\mathcal{A}_1}$. Note that although, by convention, each w^k is bilevel feasible, their composition may not. In terms of w^k, statement (e) can be restated exactly the same as statement (c) but with an alternative definition of bilevel feasible (w^k).

Lemma 3.26. *A composition (w^k) is bilevel feasible if and only if*

$$\mathbf{T}\left(\sum_{k\in\mathcal{K}} w^k\right) = \bigcap_{k\in\mathcal{K}} \mathbf{T}^k(w^k).$$

Proof. Given (w^k), for all $t \geq 0$, we have:

$$\sum_{k\in\mathcal{K}} g^k(w^k) \overset{\text{(a)}}{\geq} g\left(\sum_{k\in\mathcal{K}} w^k\right) \overset{\text{(b)}}{\geq} f(t) - \sum_{k\in\mathcal{K}} t^\top w^k = \sum_{k\in\mathcal{K}} \left(f^k(t) - t^\top w^k\right). \tag{3.20}$$

($\Rightarrow$) Assuming Equation (3.19), then the inequality at (a) is an equality. For all $t \in \mathbf{T}\left(\sum_{k\in\mathcal{K}} w^k\right)$, the inequality at (b) is satisfied with equality, hence $g^k(w^k) = f^k(t) - t^\top w^k$ and $t \in \mathbf{T}^k(w^k)$ for all $k \in \mathcal{K}$. For all $t \in \bigcap_{k\in\mathcal{K}} \mathbf{T}^k(w^k)$, all inequalities in Equation (3.20) become equalities and the equality at (b) implies $t \in \mathbf{T}\left(\sum_{k\in\mathcal{K}} w^k\right)$.

($\Leftarrow$) Since $\mathbf{T}\left(\sum_{k\in\mathcal{K}} w^k\right)$ is not empty, there exists $t \geq 0$ such that $t \in \mathbf{T}^k(w^k)$ for all $k \in \mathcal{K}$, hence $g^k(w^k) = f^k(t) - t^\top w^k$. Then, all inequalities in Equation (3.20) become equalities and the equality at (a) implies Equation (3.19). $\square$

The interpretation of Lemma 3.26 is that a composition (w^k) is bilevel feasible if and only if all action sets of its components w^k in the individual reaction plots intersect. Recall from Example 3.4 that the compositions $w^1 = (1,1)$ (Figure 3.4e) and $w^2 = (0,1)$ (Figure 3.4f) are bilevel feasible because their action sets intersect to form the aggregated action set with label $w = (1,2)$ (Figure 3.4d). On the other hand, the composition $w^1 = (0,1)$ and $w^2 = (1,1)$ is bilevel infeasible since their action sets do not intersect despite having the same w.

3.3.4.2. Strong Bilevel Feasibility. All concepts and propositions in Sections 3.3.2 and 3.3.3 are defined based solely on $f(t)$ and $g(w)$, thus they are still valid to the multi-commodity case (except for Proposition 3.21 where we need to replace $w = x_{\mathcal{A}_1}$ with $w = \sum_{k\in\mathcal{K}} x^k_{\mathcal{A}_1}$). Given $w \geq 0$, we call (w^k) a *decomposition* of w if (w^k) is a composition that satisfies:

$$w = \sum_{k\in\mathcal{K}} w^k.$$

A *bilevel feasible decomposition* of w is a decomposition which is also bilevel feasible. By Equation (3.16), any $w \geq 0$ has at least one bilevel feasible decomposition.

Lemma 3.27. *Given $w \geq 0$, the set of all bilevel feasible decompositions of w is convex.*

Proof. Let $(\hat{w}^k)$ and $(\check{w}^k)$ be two bilevel feasible decompositions of w and (w^k) be a convex combination of them, *i.e.* $w^k = \lambda\hat{w}^k + (1-\lambda)\check{w}^k$ for all $k \in \mathcal{K}$ and some $0 \leq \lambda \leq 1$. Clearly, $\sum_{k\in\mathcal{K}} w^k = w$, hence (w^k) is a decomposition of w. Since g^k are convex,

$$\sum_{k\in\mathcal{K}} g^k(w^k) \leq \sum_{k\in\mathcal{K}} \left(\lambda g^k(\hat{w}^k) + (1-\lambda) g^k(\check{w}^k)\right) = \lambda g(w) + (1-\lambda) g(w) = g(w).$$

Because $\sum_{k\in\mathcal{K}} g^k(w^k) \geq g(w)$ (by Equation (3.16)), we conclude that $\sum_{k\in\mathcal{K}} g^k(w^k) = g(w)$ and (w^k) is bilevel feasible. $\square$

Similar to the discussion in Section 3.3.4.1, even when all w^k are strongly bilevel feasible individually, it is not guaranteed that w would be strongly bilevel feasible. Moreover, w has infinitely many decompositions (w^k), and not all of them will satisfy the above property. Then, under what conditions is w strongly bilevel feasible in relation to its decomposition (w^k)?

Proposition 3.28. *A reduced reaction $w \geq 0$ is strongly bilevel feasible if and only if for all bilevel feasible decompositions (w^k) of w, each w^k is strongly bilevel feasible individually.*

Proof. ($\Rightarrow$) For all bilevel feasible decompositions (w^k) of w, Lemma 3.26 states that $\mathbf{T}(w) \subseteq \mathbf{T}^k(w^k)$ for all $k \in \mathcal{K}$. If w is strongly bilevel feasible, then $\dim \mathbf{T}(w) = n$ (Lemma 3.20), hence $\dim \mathbf{T}^k(w^k) = n$ for all $k \in \mathcal{K}$. Thus, each w^k is strongly bilevel feasible individually.

($\Leftarrow$) Suppose that w is weakly bilevel feasible, we need to show that there exists some bilevel feasible decomposition (w^k) such that some w^k are weakly bilevel feasible individually. By Lemma 3.20, $(w, g(w))$ is not an extreme point of $\operatorname{epi}(g)$, hence it can be written as a strict convex combination of two distinct points $(\hat{w}, \hat{g})$ and $(\check{w}, \check{g})$ of $\operatorname{epi}(g)$:

$$w = \lambda\hat{w} + (1-\lambda)\check{w}, \tag{3.21}$$

$$g(w) = \lambda\hat{g} + (1-\lambda)\check{g}, \tag{3.22}$$

for some $0 < \lambda < 1$. Since $g(w)$ is convex, $g(w) \leq \lambda g(\hat{w}) + (1-\lambda) g(\check{w})$; and because $(\hat{w}, \hat{g})$ and $(\check{w}, \check{g})$ are in $\operatorname{epi}(g)$, $g(\hat{w}) \leq \hat{g}$ and $g(\check{w}) \leq \check{g}$. It follows that $g(\hat{w}) = \hat{g}$, $g(\check{w}) = \check{g}$, and $\hat{w} \neq \check{w}$. Let $(\hat{w}^k)$ and $(\check{w}^k)$ be some bilevel feasible decompositions of $\hat{w}$ and $\check{w}$, respectively. Define $w^k = \lambda\hat{w}^k + (1-\lambda)\check{w}^k$ for each $k \in \mathcal{K}$. Equation (3.21) implies that (w^k) is a decomposition of w, while Equation (3.22) implies:

$$\begin{aligned} g(w) &= \lambda g(\hat{w}) + (1-\lambda) g(\check{w}) = \sum_{k\in\mathcal{K}} \left[\lambda g^k(\hat{w}^k) + (1-\lambda) g^k(\check{w}^k)\right] \\ &\geq \sum_{k\in\mathcal{K}} g^k(w^k). && \text{(since } g^k \text{ convex)} \end{aligned}$$

Because $\sum_{k\in\mathcal{K}} g^k(w^k) \geq g(w)$ (by Equation (3.16)), (w^k) is bilevel feasible. Moreover, $\lambda g^k(\hat{w}^k) + (1-\lambda) g^k(\check{w}^k) = g^k(w^k)$ for all $k \in \mathcal{K}$. Since $\hat{w} \neq \check{w}$, there exists $k \in \mathcal{K}$ such that $\hat{w}^k \neq \check{w}^k$. Thus, $(w^k, g^k(w^k))$ is a strict convex combination of two distinct points $(\hat{w}^k, g^k(\hat{w}^k))$ and $(\check{w}^k, g^k(\check{w}^k))$ of $\operatorname{epi}(g^k)$. Therefore, $(w^k, g^k(w^k))$ is not an extreme point of $\operatorname{epi}(g^k)$ and w^k is weakly bilevel feasible. $\square$

Lemma 3.29. *If $w \geq 0$ is strongly bilevel feasible, then w has exactly one bilevel feasible decomposition.*

Proof. Suppose that w is strongly bilevel feasible and it has more than one bilevel feasible decompositions. Lemma 3.27 implies that there is an infinite number of bilevel feasible decompositions (w^k) of w, hence an infinite number of compositions (w^k) such that each w^k is strongly bilevel feasible individually (Proposition 3.28). However, each epi(g^k) has a finite number of extreme points, by Lemma 3.20, the number of strongly bilevel feasible w^k is finite, so is the number of their compositions (w^k) which is a contradiction. □

Taking Proposition 3.28 and Lemma 3.29 together, we have:

Theorem 3.30. *A reduced reaction $w \geq 0$ is strongly bilevel feasible if and only if both of the following conditions are satisfied:*

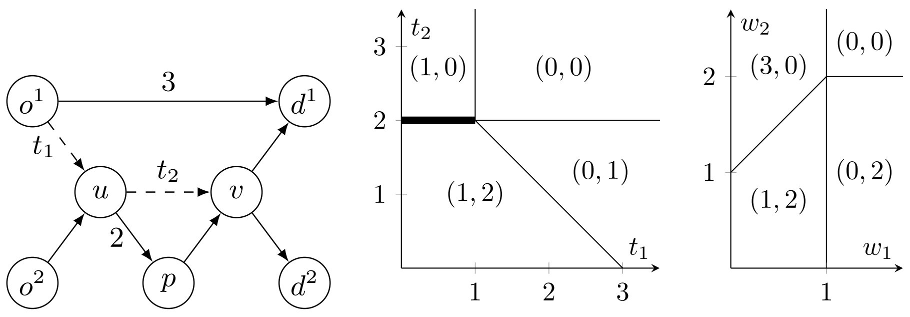
(i) There exists exactly one bilevel feasible decomposition (w^k) of w;

(ii) In the unique decomposition (w^k), each w^k is strongly bilevel feasible individually.

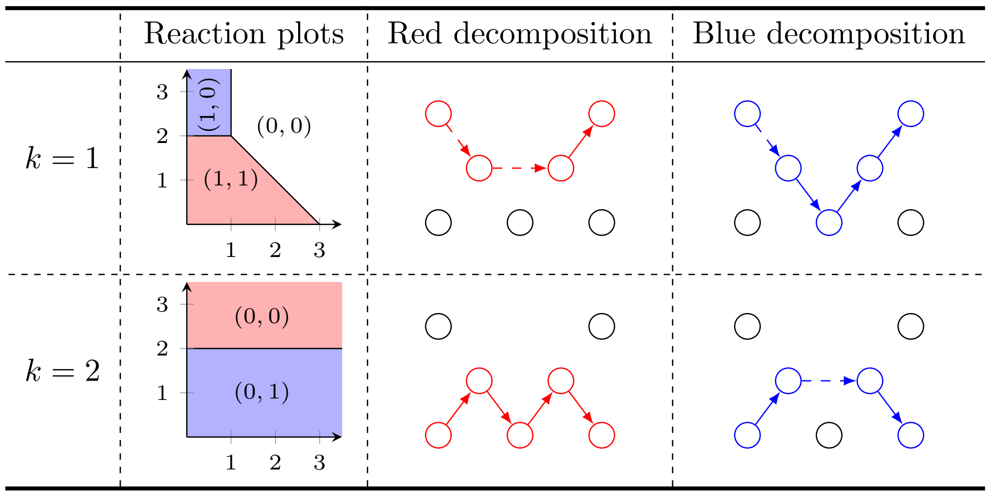
According to Theorem 3.30, if w is weakly bilevel feasible and (w^k) is some bilevel feasible decomposition of w, it might occur that each w^k is strongly bilevel feasible individually. In this case, however, w must have another bilevel feasible decomposition. Such case usually happens when many commodities share the same portion of the graph, hence they indirectly affect each other. We provide two examples to demonstrate this phenomenon.

Example 3.31. *Consider the multi-commodity NPP with the graph in Figure 3.10a. Commodity 1 can use both tolled arcs in series while commodity 2 can only use the second tolled arc. Figures 3.10b and 3.10d illustrate the reaction plots of the whole problem and of individual commodities. In Figure 3.10b, notice that there are two edges that overlap each other (highlighted as a thick line). This is due to the structure $\{u, v, p\}$ being shared by both commodities. The label jumps from* (1,0) *above the edge to* (1,2) *below the edge, skipping the label* (1,1). *The action plot (Figure 3.10c) confirms that $w =$ (1,1) is weakly bilevel feasible, whose action set $\mathbf{T}(w)$ is the thick line in Figure 3.10b. We can decompose $w =$ (1,1) in two (over infinitely many) different ways: (i) The red decomposition: $w^1 = (1, 1)$ and $w^2 = (0, 0)$; (ii) The blue decomposition: $w^1 = (1, 0)$ and $w^2 = (0, 1)$. Figure 3.10d shows that in both decompositions, each w^k is strongly bilevel feasible individually. However, since there are two bilevel feasible decompositions, by Theorem 3.30, $w = (1, 1)$ fails to be strongly bilevel feasible.*

The paths corresponding to these decompositions are illustrated in the last two columns of Figure 3.10d (unused arcs are hidden). In both decompositions, both commodities pass through nodes u and v but they travel on different subpaths between u and v: one takes the tolled arc $u - v$, while the other takes the toll-free segment $u - p - v$. However, a subpath of the shortest path is the shortest subpath. Thus, both commodities should use the same subpath between u and v: either both take the tolled arc, or both take the toll-free segment. Dewez et al. [23] use this observation to add an inequality ruling this case out. In our framework



(a) Graph ***(b)*** Overall reaction plot ***(c)*** Action plot

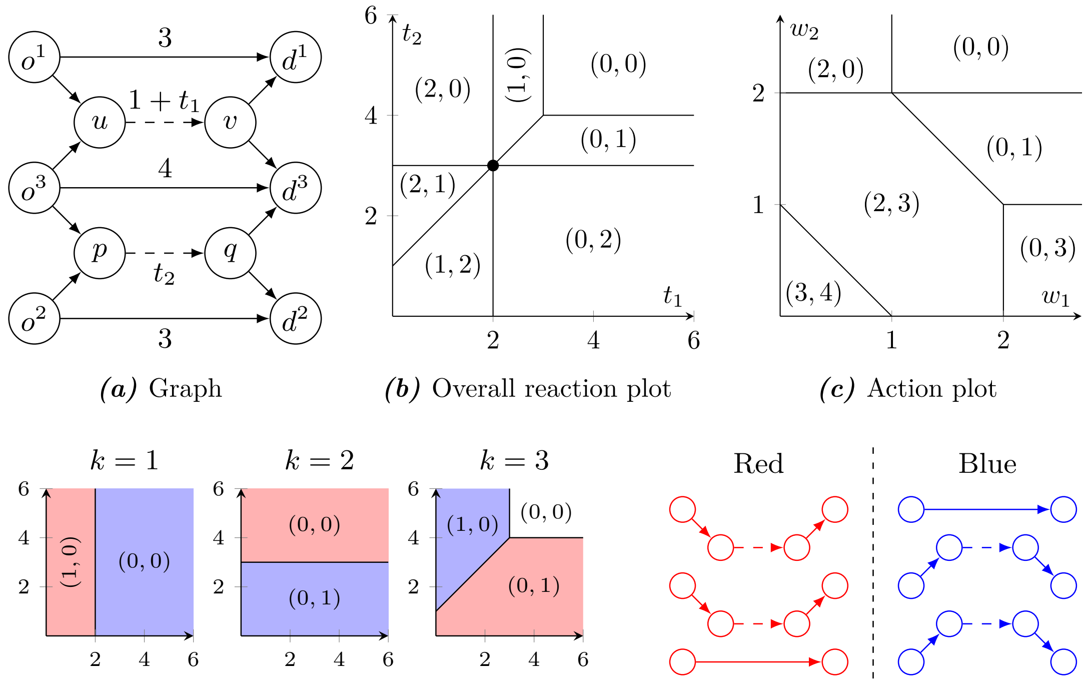


(d) Individual reaction plots and decompositions of $w = (1, 1)$

Figure 3.10 – Illustrations for Example 3.31.

developed so far, we state that $w = (1, 1)$ *is weakly bilevel feasible, and by Theorem 3.22, we should not consider* $w = (1, 1)$ *as a solution of the NPP in favor of other strongly bilevel feasible values of* w *such as* $w = (1, 2)$ *(both take the tolled arc) and* $w = (1, 0)$ *(both take the toll-free segment).*

Example 3.32. *Strong bilevel feasibility is not limited to the case described in Example 3.31, where two commodities passing through a pair of nodes must share the same subpath between those nodes. We can craft an even more intricate example involving three commodities where an integral weakly bilevel feasible* w *only appears when all three are considered. Consider the NPP with graph in Figure 3.11a. Commodities 1 and 2 can only use tolled arcs 1 and 2 respectively, while commodity 3 can use both in parallel. The reaction plots of the overall problem and individual commodities are shown in Figures 3.11b and 3.11d. We are interested in* $w = (1, 1)$ *which is weakly bilevel feasible according to the action plot in Figure 3.11c. Its action set* $\mathbf{T}(w)$ *consists of only a single point* $t = (2, 3)$*. Similar to Example 3.31,* $w = (1, 1)$ *has two bilevel feasible decompositions* (w^k)*, in which each* w^k *is strongly bilevel*



(a) Graph ***(b)*** Overall reaction plot ***(c)*** Action plot

(d) Individual reaction plots ***(e)*** Decompositions of $w = (1,1)$

Figure 3.11 – Illustrations for Example 3.32.

feasible individually: (i) The red decomposition: $w^1 = (1,0)$, $w^2 = (0,0)$, and $w^3 = (0,1)$; (ii) The blue decomposition: $w^1 = (0,0)$, $w^2 = (0,1)$, and $w^3 = (1,0)$. The paths of the decompositions are plotted in Figure 3.11e. In contrast to Example 3.31, no pair of nodes is shared between any pair of commodities, which proves that strong bilevel feasibility is a more general property than the one described in Example 3.31 (shared pair of nodes). Moreover, if any commodity is excluded, this case will disappear, which shows that strong bilevel feasibility involves multiple commodities simultaneously and cannot be considered separately.

Examples 3.31 and 3.32 illustrate the case where a weakly bilevel feasible w has a decomposition (w^k) such that each w^k is strongly bilevel feasible individually, which implies that the decomposition (w^k) is not unique. It may also happen that a weakly bilevel feasible w has a unique bilevel feasible decomposition (w^k). In this case, there must be some w^k that is weakly bilevel feasible individually. For instance, $w = (1,1)$ in Example 3.4 has a unique decomposition $w^1 = (0.5, 0.5)$ and $w^2 = (0.5, 0.5)$, making it weakly bilevel feasible. In Section 3.4.1, we will explain how to use Theorem 3.30 to test strong bilevel feasibility of a given w.

We return to the discussion at the start of this section. There are at most $2^{|\mathcal{A}_1||\mathcal{K}|}$ compositions (w^k), among which only some are bilevel feasible. However, bilevel feasibility alone is not sufficient to reduce this number to $(|\mathcal{K}|+1)^{|\mathcal{A}_1|}$, since for a given w, there can be more than one bilevel feasible decomposition. Theorem 3.30 tells us that such w is weakly bilevel feasible and by Theorem 3.22, we can eliminate all decompositions associated with w. As a result, there are at most $(|\mathcal{K}|+1)^{|\mathcal{A}_1|}$ compositions (w^k) that we need to enumerate, since there are at most $(|\mathcal{K}|+1)^{|\mathcal{A}_1|}$ values of w, and each strongly bilevel feasible w has only one bilevel feasible decomposition (w^k). Therefore, strong bilevel feasibility is a complete description of the asymmetry in the complexity of the NPP.

3.4. Numerical Experiments

In this section, we demonstrate that the asymmetry in the complexity of the NPP can be exploited to solve instances with a high number of commodities more efficiently. More specifically, we compare the performance of an MILP solver under two different MILPs: a control program similar to Program (3.10), and the same program with additional cuts called *strong bilevel feasibility cuts*. The programs and the procedure to generate the cuts are explained in details in Section 3.4.1. Our experimental results are shown in Section 3.4.2.

3.4.1. Strong Bilevel Feasibility Cut

The NPP can be formulated in many different ways as systematized in Bui et al. [14]. In our experiments, we chose the (PASTD) formulation in [14], which is recalled here for the sake of completeness:

$$\max_{t,z,y} \sum_{k\in\mathcal{K}} \sum_{p^k\in\mathcal{P}^k} \eta^k t^\top \hat{x}^{p^k} z^k_{p^k} \tag{3.23a}$$

$$\text{s.t.} \sum_{p^k\in\mathcal{P}^k} z^k_{p^k} = 1, \qquad k\in\mathcal{K}, \tag{3.23b}$$

$$A^\top y^k \le c + t, \qquad k\in\mathcal{K}, \tag{3.23c}$$

$$\sum_{p^k\in\mathcal{P}^k} (c+t)^\top \hat{x}^{p^k} z^k_{p^k} = (b^k)^\top y^k, \qquad k\in\mathcal{K}, \tag{3.23d}$$

$$t \ge 0, \tag{3.23e}$$

$$z^k_{p^k} \ge 0, \qquad k\in\mathcal{K}, p^k\in\mathcal{P}^k. \tag{3.23f}$$

This program is similar to Program (3.10) except that the arc representation x^k_a is replaced by the path representation $z^k_{p^k}$. In Program (3.23), $\mathcal{P}^k$ is the set of bilevel feasible paths, $z^k_{p^k}$ is a binary variable which indicates if the path $p^k \in \mathcal{P}^k$ is chosen, while $\hat{x}^{p^k}$ is a constant vector, whose element $\hat{x}^{p^k}_a = 1$ if arc $a \in \mathcal{A}$ is in path p^k, 0 otherwise. The program also includes non-unit demand (or volume) η^k for each commodity. For the details on the linearization of the bilinear terms $t^\top \hat{x}^{p^k} z^k_{p^k}$ and the enumeration of paths to produce $\mathcal{P}^k$, refer to Bui et al. [14].

The Program (3.23) is the control program, on which we will add strong bilevel feasibility cuts. The main idea of the cuts is to eliminate compositions of paths $(\hat{x}^{p^k})$ such that $(\hat{x}^{p^k})$ is bilevel infeasible or $w = \sum_{k\in\mathcal{K}} \hat{x}^{p^k}_{\mathcal{A}_1}$ is weakly bilevel feasible, which is undesirable in either case according to Theorem 3.22. To do so, we need a test for strong bilevel feasibility. To form $\mathcal{P}^k$, we enumerate bilevel feasible paths $\hat{x}^{p^k}$ which, in the NPP, are already extreme points of $\mathcal{X}^k$. Thus, for all $k \in \mathcal{K}$, $p^k \in \mathcal{P}^k$, $w^k = \hat{x}^{p^k}_{\mathcal{A}_1}$ are, in fact, strongly bilevel feasible individually. By Theorem 3.30, the aggregated reaction $w = \sum_{k\in\mathcal{K}} w^k$ will be strongly bilevel feasible if and only if w has no other bilevel feasible decomposition besides (w^k). Given a composition of paths $(\hat{x}^{p^k})$, first we calculate $w = \sum_{k\in\mathcal{K}} \hat{x}^{p^k}_{\mathcal{A}_1}$, then we solve the following program:

$$
\begin{aligned}
\max_{x} \quad & \sum_{k\in\mathcal{K}} \sum_{a\in\mathcal{A}_1} (1 - \hat{x}^{p^k}_a) x^k_a && && \text{(3.24a)}\\
\text{s.t.} \quad & \sum_{k\in\mathcal{K}} x^k_{\mathcal{A}_1} \le w, && && \text{(3.24b)}\\
& Ax^k = b^k, && k \in \mathcal{K}, && \text{(3.24c)}\\
& \sum_{k\in\mathcal{K}} c^\top x^k \le \sum_{k\in\mathcal{K}} c^\top \hat{x}^{p^k}, && && \text{(3.24d)}\\
& x^k \ge 0, && k \in \mathcal{K}. && \text{(3.24e)}
\end{aligned}
$$

If Program (3.24) has an objective value strictly greater than 0, then there are three cases:

— Constraint (3.24d) is strict, then $\sum_{k\in\mathcal{K}} c^\top \hat{x}^{p^k}$ is not the smallest sum of base costs, hence the composition $(\hat{x}^{p^k})$ is bilevel infeasible (Definition 3.23).

— Constraint (3.24d) is an equality but constraint (3.24b) is strict for some $a \in \mathcal{A}_1$, then $t_a = 0$ for all $t \in \mathbf{T}(w)$ (complementary slackness). This means $\dim \mathbf{T}(w) < n$ implying that w is weakly bilevel feasible.

— Both constraints (3.24b) and (3.24d) are equalities, then both $(\hat{x}^{p^k}_{\mathcal{A}_1})$ and $(x^k_{\mathcal{A}_1})$ are bilevel feasible decompositions of w. The objective function (3.24a) implies that $(\hat{x}^{p^k}_{\mathcal{A}_1})$ and $(x^k_{\mathcal{A}_1})$ are distinct. Thus, w is weakly bilevel feasible.

Therefore, if Program (3.24) has a non-zero objective value, we can add a cut to eliminate the composition $(\hat{x}^{p^k})$.

Removing one composition at a time is inefficient and results in a large number of cuts. Besides, the total number of compositions $(\hat{x}^{p^k})$ that we need to test is exponential. Thus, we limit the test to only pairs of paths. Next, we try to group the pairs of paths that will be eliminated into blocks so we can produce more efficient cuts. The process is described in Algorithm 3.2. Line 2 calculates the closeness score d_{ij} for each pair of commodities, defined as:

$$d_{ij} = \frac{|\mathcal{A}^{k_i} \cap \mathcal{A}^{k_j}|}{\log^2(|\mathcal{P}^{k_i}||\mathcal{P}^{k_j}|)} \tag{3.25}$$

where $\mathcal{A}^{k_i} \subseteq \mathcal{A}$ is the set of arcs that is used by at least one path in $\mathcal{P}^{k_i}$ (same for $\mathcal{A}^{k_j}$). The idea is that the more arcs two commodities share, the more bilevel infeasible $(\hat{x}^p, \hat{x}^q)$ there will be. The denominator in Equation (3.25) prioritizes pairs with low number of paths, since there are less compositions $(\hat{x}^p, \hat{x}^q)$ to test, hence these pairs are faster to process. Line 3 limits the generation of cuts to only N pairs. This parameter N is used to adjust the amount of cuts added to the model. Lower values of N mean fewer cuts and less time spent on generating cuts (and more time spent on actually solving the program). Lines 5-7 conduct the strong bilevel feasibility test between all pairs of paths in the given pair of commodities to build the matrix h_{pq}. Line 8 groups all the compositions $(\hat{x}^p, \hat{x}^q)$ that will be eliminated, *i.e.* $h_{pq} = 1$, into blocks. These blocks are the bicliques of a bipartite graph whose nodes correspond to paths of each commodity $(\mathcal{P}^{k_i}, \mathcal{P}^{k_j})$, and edges connect the pairs of paths that will be eliminated. Thus, within a biclique $(\hat{\mathcal{P}}, \hat{\mathcal{Q}}), \hat{\mathcal{P}} \subseteq \mathcal{P}^{k_i}, \hat{\mathcal{Q}} \subseteq \mathcal{P}^{k_j}$, all $(p, q) \in \hat{\mathcal{P}} \times \hat{\mathcal{Q}}$ can be removed. Therefore, in line 9, for each biclique $(\hat{\mathcal{P}}, \hat{\mathcal{Q}})$, we add an inequality constraint in the form of:

$$\sum_{p \in \hat{\mathcal{P}}} z_p^{k_i} + \sum_{q \in \hat{\mathcal{Q}}} z_q^{k_j} \leq 1. \tag{3.26}$$

Ideally, we want to add as few cuts as possible, so in line 8, our aim is to find the cover with the fewest number of bicliques. Unfortunately, the minimum biclique cover problem is NP-hard [28], so we use a heuristic. Note that the process in Algorithm 3.2 is just one of the possible ways of generating some strong bilevel feasibility cuts, and serves only as a proof of concept. In a more general implementation, one can modify the closeness score, test strong bilevel feasibility for more than two paths at a time, or ignore the cut for some bicliques.

3.4.2. Setup and Results

We tested the performance of the cuts by varying the parameter N in Algorithm 3.2 within the set $\{0, 10, 20, 50, 100, 200, 500, 1000\}$. The case with no cuts ($N = 0$) is the control case. If the cuts are effective, then we should see better results when N increases. All cases were tested on 350 randomly generated instances, whose graphs are $L \times L$ grid with L decreasing from 12 to 6 (50 instances for each L). Decreasing L makes the graph smaller, hence lowers the difficulty. To compensate for this, we increase the number of commodities $|\mathcal{K}|$ so that the product $|\mathcal{A}_1||\mathcal{K}|$ stays constant (which is the number of binary variables of the MILP, as explained in Section 3.2.3). The particular values of L and $|\mathcal{K}|$ are listed in Table 3.2. The fourth column in Table 3.2 shows the reduction ratios of $|\mathcal{A}|$ compared to the first case ($L = 12$). As a result, $|\mathcal{K}|$ in the last five columns must increase by the same

Algorithm 3.2 Generation of strong bilevel feasibility cuts

```
Input: The NPP instance, parameter $N$.
Output: Program (3.23) with strong bilevel feasibility cuts.
1: For each $k \in \mathcal{K}$, enumerate the set of bilevel feasible paths $\mathcal{P}^k$
2: For each $(k_i, k_j) \in \mathcal{K}^2$, calculate the closeness score $d_{ij}$
3: Choose $N$ pairs in $\mathcal{K}^2$ with the highest $d_{ij}$ to form $\mathcal{K}_N^2$
4: for all $(k_i, k_j) \in \mathcal{K}_N^2$ do
5:     for all $(p, q) \in \mathcal{P}^{k_i} \times \mathcal{P}^{k_j}$ do
6:         Run Program (3.24) on composition $(\hat{x}^p, \hat{x}^q)$
7:         $h_{pq} \leftarrow 1$ if the objective value is non-zero, 0 otherwise
8:     Find a solution of the biclique cover problem between $\mathcal{P}^{k_i}$ and $\mathcal{P}^{k_j}$ with $h_{pq}$ being
       the adjacency matrix
9:     Add cuts of the form in Equation (3.26) based on the biclique cover
```

Table 3.2 – The number of commodities $|\mathcal{K}|$ in relation to the grid size L.

L	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{A}\rvert$	Ratio	$\lvert\mathcal{K}\rvert$				
12	144	528	1.00	30	35	40	45	50
11	121	440	1.20	36	42	48	54	60
10	100	360	1.47	44	51	59	66	73
9	81	288	1.83	55	64	73	82	92
8	64	224	2.36	71	82	94	106	118
7	49	168	3.14	94	110	126	141	157
6	36	120	4.40	132	154	176	198	220

ratios. Ten instances are generated for each cell in the last five columns. There are five columns and seven rows, which results in 350 instances in total. Other parameters such as the distribution of the costs, the proportion of tolled arcs, etc., can be found in Bui et al. [14].

We conducted 2800 runs in total (8 values of N times 350 instances). At the start of each run, commodity-wise preprocessing (described in [14]) is applied. Then, strong bilevel feasibility cuts are generated and added to Program (3.23) based on N. The preprocessing and cut generation are implemented in Julia. After all cuts are added, we use Gurobi 1.6.1 [31] to solve the final MILP. Every step is done in a single thread (AMD EPYC 7532 2.4GHz CPU, 8GB of RAM). The time limit for each run is one hour, including the time for preprocessing, cut generation, and solution. All runs were done on the Narval cluster provided by Compute Canada.

The runs are grouped by N and L, with 50 runs per group (10 instances for 5 different values of $|\mathcal{K}|$ given L). We evaluate each group using three metrics: the number of instances solved to optimality within the time limit, the average time, and the average optimality gap. For the first metric, higher is better, while it is the reverse for the other two. The results are shown in Table 3.3. Figure 3.12 plots the metrics for $N = 0$, 100, and 1000. According to the results, increasing N improves the performance significantly when L is low. Specifically, at $L = 6$, the case $N = 1000$ can solve 15/50 more instances compared to the control case $N = 0$. On the contrary, the cuts have a detrimental effect on larger graphs, *i.e.* when L is high. Thus, the cuts are more effective in the cases with high $|\mathcal{K}|$ and low $|\mathcal{A}_1|$. This demonstrates that there exists an asymmetry in the complexity of the NPP and it is possible to take advantage of this asymmetry to accelerate the solution of the NPP with a small graph and a high number of commodities.

3.5. Conclusion

In this paper, we proved that the number of commodities $|\mathcal{K}|$ and the number of tolled arcs $|\mathcal{A}_1|$ scale the NPP differently. Particularly, if $|\mathcal{A}_1|$ is fixed, the problem can be solved in polynomial time with respect to $|\mathcal{K}|$. We explained the logic behind this asymmetry with the use of convex conjugate, through which we derived the conjugate model, bilevel feasibility, and strong bilevel feasibility. To the best of our knowledge, this approach is innovative compared to other applications of the convex conjugate in bilevel programming such as Fenchel-Lagrange duality [1]. Indeed, these tools can be extended to other pricing problems as long as the assumption that f is a polyhedral function is kept (as utilized in Theorem 3.17). Future directions include further algorithmic exploration of the generation of strong bilevel feasibility cuts and extensions to the case where the follower solves an MILP.

Acknowledgments. This work was funded by FRQ-IVADO Research Chair in Data Science for Combinatorial Game Theory, and the NSERC grant 2019-04557.

This research was enabled in part by support provided by Calcul Québec (`https://www.calculquebec.ca`) and the Digital Research Alliance of Canada (`https://alliancecan.ca`).

Table 3.3 – Results of the experiment.

Metrics	N	L						
		12	11	10	9	8	7	6
Number of solved instances	0	20	12	15	11	13	14	19
	10	19	11	13	11	14	15	19
	20	19	11	15	11	14	14	21
	50	19	13	13	11	13	16	20
	100	17	12	14	12	16	16	22
	200	17	12	14	10	14	21	25
	500	17	11	12	9	16	27	31
	1000	17	11	11	7	15	25	34
Average time (minutes)	0	42	50	49	51	49	50	44
	10	42	50	49	52	49	50	44
	20	43	51	50	51	49	50	45
	50	44	51	50	51	49	50	43
	100	47	52	50	51	48	49	43
	200	49	53	52	52	48	48	40
	500	51	55	54	54	48	42	36
	1000	51	55	56	55	51	43	31
Average gap (%)	0	4.5	4.7	4.7	5.0	5.2	4.2	3.6
	10	4.5	4.9	4.6	5.3	5.1	4.2	3.6
	20	4.8	4.8	4.6	4.9	5.1	4.3	3.4
	50	4.6	4.9	4.7	5.1	5.0	4.2	3.5
	100	4.8	5.3	5.0	5.4	4.9	4.0	3.3
	200	5.2	5.7	5.5	5.9	5.0	3.7	3.1
	500	5.7	6.6	6.2	6.4	5.0	3.1	2.4
	1000	5.9	7.6	8.3	7.7	6.3	3.3	2.0

3.A. Extension to the non-unit demand case

Let η^k be the demand of commodity $k \in \mathcal{K}$. The bilevel formulation of the non-unit demand case is:

$$\max_{t,x^k} \Big\{ \sum_{k \in K} \eta^k t^\top x^k \mid t \in \mathcal{T}, x^k \in \mathbf{R}^k(t) \ \ \forall k \in \mathcal{K} \Big\},$$

where $\mathbf{R}^k(t)$ is the reaction set of follower $k \in \mathcal{K}$, defined in Equation (3.4).

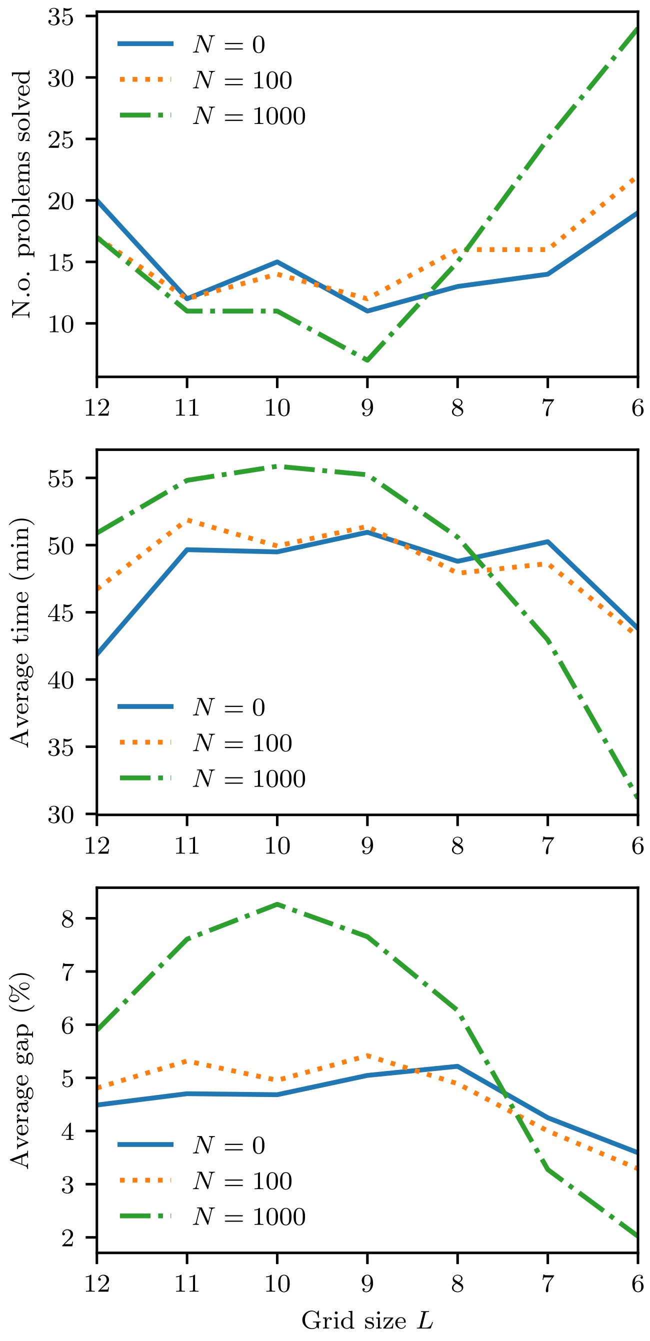


Figure 3.12 – Plots of the results for selected values of N.

To derive the conjugate model for the non-unit demand case, we aggregate the followers' problems with weights η^k. In particular, the conjugate bilevel formulation of the non-unit demand case is still:

$$\max_{w,t}\{w^\top t \mid w \geq 0, t \in \mathbf{T}(w)\}$$

where $\mathbf{T}(w)$ is the set of t that are optimal to the modified conjugate follower formulation:

$$\max_{t,y^k}\Big\{\sum_{k\in K}\left(\eta^k (b^k)^\top y^k\right) - w^\top t \mid A^\top y^k - t \leq c \ \ \forall k \in \mathcal{K}, t \geq 0\Big\}.$$

In this case, we have $w = \sum_{k\in K}\eta^k x^k_{\mathcal{A}_1}$ which may not be integral. However, Theorem 3.1 still applies since the reaction plot is invariant to η^k, hence there are still at most $(|\mathcal{K}|+1)^{|\mathcal{A}_1|}$ regions in the composed reaction plot. In fact, with fixed $|\mathcal{A}_1|$, one can solve the non-unit demand case in polynomial time as follows:

(1) Assume $\eta^k = 1$ and enumerate $\tilde{w}$ in $\{0, 1, \ldots, |\mathcal{K}|\}^{|\mathcal{A}_1|}$;

(2) Solve for t given $\tilde{w}$ as in the unit demand case;

(3) Solve for x^k given t and compute the final revenue $\sum_{k\in K}\eta^k t^\top x^k$.

,

Chapter 4 Third article

Solving Combinatorial Pricing Problems using Embedded Dynamic Programming Models

by

Quang Minh Bui[1], Margarida Carvalho[1], and José Neto[2]

([1]) CIRRELT and Département d'informatique et de recherche opérationnelle
Université de Montréal
([2]) Télécom SudParis
Institut Polytechnique de Paris

This article will be submitted to a peer-reviewed journal..

My contributions and the roles of the coauthors:

— Margarida and José suggested several papers on the topic, including the solution of bilevel programs with decision diagrams.

— I expanded the above concept with the use of selection diagrams and the dynamic generation of constraints.

— The experiments were designed, implemented, and run by me.

— The article (including all algorithms and examples) was written by me and it was revised by Margarida and José.

Abstract. The combinatorial pricing problem (CPP) is a bilevel problem in which the leader maximizes their revenue by imposing tolls on certain items that they can control. Based on the tolls set by the leader, the follower selects a subset of items corresponding to an optimal solution of a combinatorial optimization problem. To accomplish the leader's goal, the tolls need to be sufficiently low to discourage the follower from choosing the items offered by the competitors. In this paper, we derive a single-level reformulation for the CPP by rewriting the follower's problem as a longest path problem using a dynamic programming model, and then taking its dual and applying strong duality. We proceed to solve the reformulation in a dynamic fashion with a cutting plane method. We apply this methodology to two distinct dynamic programming models, namely, a novel formulation designated as selection diagram and the well-known decision diagram. We also produce numerical results to evaluate their performances across three different specializations of the CPP. Our results showcase the advantage of the two proposed reformulations over the natural value function approach, expanding the tools to solve combinatorial bilevel programs.

Keywords: Combinatorial pricing problem, Stackelberg pricing game, Selection diagram, Decision diagram

4.1. Introduction

Consider a bilevel problem with two decision makers where one agent, the *leader*, sets the *tolls* (*i.e.* markups) of certain items, then the other agent, the *follower*, selects a subset of items according to another optimization problem (also known as the *follower's problem*). The follower has a choice between the items that the leader controls (called *tolled items*), and other items offered by the leader's competitors, whose prices, we assume, are fixed. The goal of the leader is to maximize the sum of tolls of the selected tolled items. Although the leader naturally wants to set the tolls high, such tolls cannot be excessive since the leader still needs the follower to choose their items in order to gain a profit. Thus, the objectives of the two agents are neither adversarial nor cooperative. In this paper, we investigate the case where the follower's problem is a combinatorial optimization problem which can be expressed as a binary linear program. We refer to the overall bilevel problem as *combinatorial pricing problem* (CPP). In the literature, the CPP is also called the Stackelberg pricing game.

Motivation. The CPP is inspired by the network pricing problem (NPP) introduced by Labbé et al. [40]. The NPP is a specific case of the CPP where the follower's problem is a shortest path problem. In the CPP, the shortest path problem is replaced by other combinatorial problems such as the minimum spanning tree problem [18], the knapsack problem [7, 50], the stable set problem [16], and the bipartite vertex cover problem [8]. This class of problems is usually proven to be NP-hard [18, 51] or Σ_2^p-hard [16].

Except for the NPP, most papers in the literature focus on the problems computational complexity classification and approximation schemes. Regarding the exact solution methods, general bilevel solvers like `MibS` [54] or the one by Fischetti et al. [26] are not applicable due to the leader's variables (the tolls in this case) being continuous and appearing in the follower's constraints. A popular method to solve the NPP is to find a dual of the follower's problem and then use the Karush-Kuhn-Tucker conditions to convert the bilevel problem to a single-level reformulation [6, 14, 24, 33]. This method, referred to as *dualize-and-combine*, is not trivial to generalize to the CPP because a dual of a binary linear program is not generally defined and even if it exists, it is often computationally untractable. A relatively novel technique to formulate a dual is described in Lozano et al. [43], in which the authors use decision diagrams [3] to derive single-level reformulations for interdiction problems.

Contributions. Our first contribution is the application of the technique in Lozano et al. [43] to the CPP. We convert the follower's problem into an equivalent longest path problem corresponding to a dynamic programming model. The longest path problem has a linear programming formulation, hence we can dualize this linear program and combine it with the original primal follower's program to create a valid single-level reformulation for the CPP. Besides decision diagrams, we also experiment with a new dynamic programming model called *selection diagrams*.

Even with the reformulation well-defined, the computational untractability issue still remains due to the very likely large size of the reformulation. Thus, we propose a dynamic constraint generation scheme similar to the cutting plane method. We provide numerical results to compare the performance of these formulations in the context of 3 specializations of the CPP, in which the follower's problems are the knapsack problem, the maximum stable set problem, and the minimum set cover problem, respectively. These results support a promising direction to scale the solving of the CPP.

Paper Organization. In Section 4.2, we describe the bilevel formulation of the CPP and its *value function reformulation*, which serves as an introduction to the dualize-and-combine methodology. We define two different dynamic programming models: selection diagram and decision diagram in Section 4.3, as well as the dynamic constraint generation algorithms corresponding to each model. Section 4.4 presents the experiment setups and numerical results. Finally, Section 2.6 concludes the paper.

4.2. Combinatorial Pricing Problem

4.2.1. Problem Description

Consider a combinatorial optimization problem with decision variables $x \in \mathcal{X} \subseteq \{0,1\}^{\mathcal{I}}$ indexed by the set of *items* $\mathcal{I}$. Each item $i \in \mathcal{I}$ has an associated *base value* $v_i \in \mathbb{R}$. The set

$\mathcal{I}$ is partitioned into the set of *tolled items* $\mathcal{I}_1$ and the set of *toll-free items* $\mathcal{I}_2$. The leader wishes to impose a toll $t_i \geq 0$ for each tolled item $i \in \mathcal{I}_1$. For the toll-free items $i \in \mathcal{I}_2$, we set $t_i = 0$. Let $\mathcal{T} = \mathbb{R}_+^{\mathcal{I}_1} \times \{0\}^{\mathcal{I}_2}$ be the set of feasible tolls. The overall CPP is formulated as a bilevel program:

$$\max_{t,x}\{t^\top x \mid t \in \mathcal{T},\, x \in \mathbf{R}(t)\}$$

where $\mathbf{R}(t) \subseteq \mathcal{X}$ is the set of optimal solutions of the *follower's problem* given the toll t, defined by:

$$\mathbf{R}(t) = \arg\max_x\{(v-t)^\top x \mid x \in \mathcal{X}\}. \tag{4.1}$$

Note that if $\mathbf{R}(t)$ is not a singleton, *i.e.* there are multiple optimal solutions of the follower's problem, then the follower will choose a solution x that yields the most revenue $t^\top x$ for the leader. This is commonly referred to as the *optimistic assumption.*

Example 4.1. *The knapsack pricing problem (KPP) is a case of the CPP where the underlying optimization problem is a knapsack problem. Let $w \in \mathbb{R}_+^{\mathcal{I}}$ and $C > 0$ be the knapsack weights and capacity, respectively. Then, the follower's problem of the KPP becomes:*

$$\mathbf{R}(t) = \arg\max_x\{(v-t)^\top x \mid w^\top x \leq C,\, x \in \{0,1\}^{\mathcal{I}}\}.$$

We make several remarks regarding the KPP:

Remark 4.2. *The leader has no incentive to set $t_i > v_i$. Indeed, in the perspective of the follower, $v_i - t_i < 0$ is the same as $v_i - t_i = 0$, and neither case gives the leader any revenue, so the leader may as well set $t_i = v_i$. Thus, we can assume that $t_i \leq v_i$ for all $i \in \mathcal{I}$.*

Remark 4.3. *There always exists an optimal solution x^* of the KPP that is maximal. This follows Remark 4.2, since including any new item produces a profit for both the follower and the leader. This property is referred to as monotonicity [27].*

4.2.2. Value Function Formulation

One method to solve the CPP is to convert the bilevel program into a single-level reformulation. Such formulation requires a dual representation of the follower's problem (4.1). The follower's problems that we are interested in are binary programs, hence the dual representation is not trivial to derive. Despite that, next we show how such dual representation could be determined and used within a cutting plane method.

Define a new binary variable $z_{\hat{x}} \in \{0,1\}$ that indicates if a particular solution $\hat{x} \in \mathcal{X}$ is selected or not. We reformulate Program (4.1) as follows:

$$\max_z\Big\{\sum_{\hat{x}\in\mathcal{X}}((v-t)^\top\hat{x})z_{\hat{x}} \;\Big|\; \sum_{\hat{x}\in\mathcal{X}} z_{\hat{x}} = 1,\, z \in \{0,1\}^{\mathcal{X}}\Big\}. \tag{4.2}$$

Program (4.2) is totally unimodular, hence we can relax $z \in \{0,1\}^{\mathcal{X}}$ to $z \geq 0$. The dual of its linear relaxation is:

$$\min_{L}\{L \mid L \geq (v-t)^{\top}\hat{x},\, \hat{x} \in \mathcal{X}\}. \tag{4.3}$$

Combining Program (4.1) and Program (4.3) with the strong duality condition, we derive the single-level reformulation:

$$\begin{aligned}
\max_{t,x,L} \quad & t^{\top}x && && \text{(4.4a)}\\
\text{s.t.} \quad & x \in \mathcal{X}, && && \text{(4.4b)}\\
& L \geq (v-t)^{\top}\hat{x}, && \hat{x} \in \mathcal{X}, && \text{(4.4c)}\\
& (v-t)^{\top}x = L, && && \text{(4.4d)}\\
& t \in \mathcal{T}. && && \text{(4.4e)}
\end{aligned}$$

Observe that Constraints (4.4c) and (4.4d) imply:

$$(v-t)^{\top}x \geq (v-t)^{\top}\hat{x}, \qquad \hat{x} \in \mathcal{X}. \tag{4.5}$$

This is usually referred as the *(optimal) value function constraint.* Thus, we refer to Program (4.4) as the *value function formulation.*

Example 4.4. *Consider a KPP instance of four items $\mathcal{I} = \{1,2,3,4\}$. The first three items are tolled,* i.e. $\mathcal{I}_1 = \{1,2,3\}$ *and* $\mathcal{I}_2 = \{4\}$*. The base values are* $v = (1,1,1,1)$*. The knapsack weights and capacity are* $w = (1,1,1,2)$ *and* $C = 3$ *respectively. Thus, we have*

$$\mathcal{X} = \{x \in \{0,1\}^4 \mid x_1 + x_2 + x_3 + 2x_4 \leq 3\}.$$

By Remark 4.3, we only need to keep track of the maximal solutions. There are four maximal solutions in $\mathcal{X}$: (1,1,1,0), (1,0,0,1), (0,1,0,1), *and* (0,0,1,1)*. It is more convenient to refer to them as the sets they represent:* {1,2,3}, {1,4}, {2,4}, *and* {3,4}*. The value* *function formulation corresponding to this KPP instance is:*

$$\begin{aligned}
\max_{t,x,L} \quad & t_1x_1 + t_2x_2 + t_3x_3 \\
s.t. \quad & x_1 + x_2 + x_3 + 2x_4 \leq 3, \\
& L \geq 3 - t_1 - t_2 - t_3, \\
& L \geq 2 - t_1, \\
& L \geq 2 - t_2, \\
& L \geq 2 - t_3, \\
& (1 - t_1)x_1 + (1 - t_2)x_2 + (1 - t_3)x_3 + x_4 = L, \\
& x \in \{0, 1\}^4, \\
& t \geq 0.
\end{aligned}$$

To solve Program (4.4) as an MILP, we need to linearize the bilinear term $t^\top x$. This can be done using the McCormick envelope [46] by replacing t_ix_i for each $i \in \mathcal{I}_1$ with s_i, accompanied by the following constraints:

$$0 \leq s_i \leq M_ix_i, \qquad\qquad 0 \leq t_i - s_i \leq M_i(1 - x_i),$$

where M_i is a sufficiently large bound of t_ix_i. For instance, a valid bound for the KPP is $M_i = v_i$ due to Remark 4.2.

Typically, the size of $\mathcal{X}$ is exponential, so in practice, we generate Constraint (4.4c) dynamically in a cutting plane fashion as described in Algorithm 4.1.

4.3. Embedded Dynamic Programming Model

The value function formulation in Program (4.4) is simple and general. However, the structure of the follower's problem is oblivious to the dual representation. We aim to introduce some structural information back to the dual in the form of a dynamic programming (DP) model. It is well-known that any DP model with a finite number of states and actions can be represented as a directed multigraph [4]. Thus, for our convenience, we define all DP models by their graph representations. Suppose that the follower's problem possesses a DP model whose definition includes:

— A set of *nodes* $\mathcal{V}$ (corresponding to the *states*);

— A set of *arcs* $\mathcal{A}$ (corresponding to the *actions*);

— Two functions $u, w : \mathcal{A} \to \mathcal{V}$ that return the source node $u(a)$ and the target node $w(a)$ of arc a, respectively;

— An *item function* $I : \mathcal{A} \to 2^{\mathcal{I}}$ which is later used to define the length of the arcs;

Algorithm 4.1 Solution of Program (4.4) with dynamically generated constraints

```
Input: The CPP instance.
Output: The optimal solution $(t^*, x^*)$ of the CPP instance.
 1: Create the master problem from Program (4.4) without Constraint (4.4c).
 2: loop
 3:     Solve the master problem and let $(\tilde{t}, \tilde{x})$ be the current optimal solution.
 4:     Solve Program (4.1) with $t = \tilde{t}$ for $\hat{x}$.      ▷ i.e. choose any $\hat{x} \in \mathbf{R}(\tilde{t})$
 5:     if $(v - \tilde{t})^\top \tilde{x} < (v - \tilde{t})^\top \hat{x}$ then
 6:         ▷ $(\tilde{t},\tilde{x})$ is not bilevel feasible. Add new constraint.      ◁
 7:         Add the constraint $L \geq (v - t)^\top \hat{x}$ to the master program.
 8:     else
 9:         ▷ $(\tilde{t},\tilde{x})$ is bilevel feasible. Stop.      ◁
10:         $(t^*, x^*) \leftarrow (\tilde{t}, \tilde{x})$
11:         break
12: return $(t^*, x^*)$
```

— An *initial node* $p \in \mathcal{V}$ and a *terminal node* $q \in \mathcal{V}$.

The directed multigraph $(\mathcal{V}, \mathcal{A}, u, w)$ must be acyclic. We define the *length* $F(a; t)$ of arc $a \in \mathcal{A}$ given toll $t \in \mathcal{T}$ as the sum of the tolled values of all the items in $I(a)$:

$$F(a; t) = \sum_{i \in I(a)} (v_i - t_i).$$

The objective of the DP model is to find the longest path from p to q. Henceforth, when we refer to *paths*, we mean paths from p to q. This optimization problem can be expressed by the following formulation (which is the dual of a typical linear formulation for the longest path problem):

$$\begin{aligned} \min_{y} \quad & y_p && && (4.6a) \\ \text{s.t.} \quad & y_{u(a)} \geq F(a; t) + y_{w(a)}, && a \in \mathcal{A}, && (4.6b) \\ & y_q = 0. && && (4.6c) \end{aligned}$$

Because the DP model is equivalent to the follower's problem (*i.e.* they return the same optimal objective value), we can use Program (4.6) as a dual representation in a similar way to Program (4.3). The result is a new single-level reformulation:

$$
\begin{aligned}
\max_{t,x,y} \quad & t^\top x && && \text{(4.7a)}\\
\text{s.t.} \quad & x \in \mathcal{X}, && && \text{(4.7b)}\\
& y_{u(a)} \geq F(a;t) + y_{w(a)}, && a \in \mathcal{A}, && \text{(4.7c)}\\
& y_q = 0, && && \text{(4.7d)}\\
& (v-t)^\top x = y_p, && && \text{(4.7e)}\\
& t \in \mathcal{T}. && && \text{(4.7f)}
\end{aligned}
$$

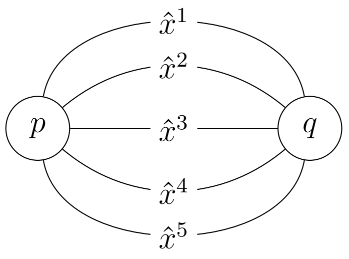


Figure 4.1 – Value function formulation as a DP model.

Example 4.5. *The value function formulation Program (4.4) can be thought as a special case of Program (4.7). Indeed, consider a DP model consisting of only two nodes p and q. Each solution $\hat{x} \in \mathcal{X}$ corresponds to an arc $a_{\hat{x}} \in \mathcal{A}$. The item function is defined to be the set of items selected in $\hat{x}$:*

$$I(a_{\hat{x}}) = \{i \in \mathcal{I} \mid \hat{x}_i = 1\}.$$

Then, the variable L in Program (4.4) is identical to y_p in Program (4.7). The DP graph of the value function formulation is shown in Figure 4.1. In all graphs illustrated in this paper, the arcs are always directed from left to right.

In this section, we explore the application of Program (4.7) to two types of DP models: *selection diagrams* (Section 4.3.1) and *decision diagrams* (Section 4.3.2). The former defines the actions based on the items to be selected, while the latter defines the actions as decisions to include or exclude a specific item. For each type of model, we also describe the procedure to solve Program (4.7) dynamically as in Algorithm 4.1 alongside any other technical improvement.

4.3.1. Selection Diagram

The first family of dynamic programming models that we investigate is called *selection diagram*. In a selection diagram, at each step, the follower either selects an item to be included into the current partial solution, or decides to stop and returns the solution. In particular, let $\mathcal{I}(\hat{x}) = \{i \in \mathcal{I} \mid \hat{x}_i = 1\}$ be the set of items selected in $\hat{x} \in \mathcal{X}$. The set of nodes in a selection diagram is the family of subsets of $\mathcal{I}$ that are contained in some feasible solution in $\mathcal{X}$, adjoined with a special terminal node q:

$$\mathcal{V} = \{J \subseteq \mathcal{I} \mid J \subseteq \mathcal{I}(\hat{x}) \text{ for some } \hat{x} \in \mathcal{X}\} \cup \{q\}.$$

The initial node in this case is the node $\varnothing$.

For every pair of nonterminal nodes J, $K \in \mathcal{V} \setminus \{q\}$ such that $J \subsetneq K$ and $|K| - |J| = 1$, we add an arc a_{JK} from J to K such that $I(a_{JK}) = K \setminus J$. Evidently, $I(a_{JK})$ is a singleton. Moreover, given a nonterminal node J, if $J = \mathcal{I}(\hat{x})$ for some $\hat{x} \in \mathcal{X}$, then we connect J directly to q with a *terminal arc* $\bar{a}_J$ with $I(\bar{a}_J) = \varnothing$.

With this construction, the nodes on a path from $\varnothing$ to q form a sequence of nested subsets of $\mathcal{I}$:

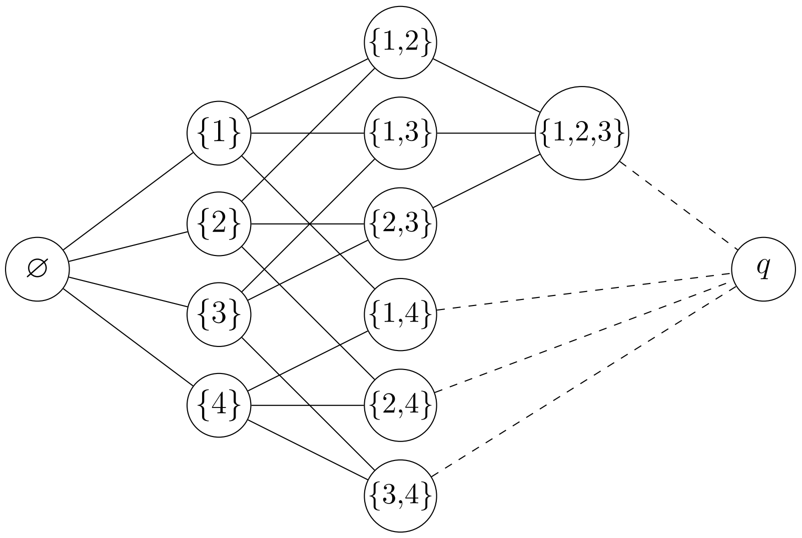
$$\varnothing \ , \ \{i_1\} \ , \ \{i_1, i_2\} \ , \ \ldots \ , \ \{i_1, \ldots, i_n\} \ , \ q$$

such that $\{i_1, \ldots, i_n\} = \mathcal{I}(\hat{x})$ for some $\hat{x} \in \mathcal{X}$. The length of such a path is exactly $(v - t)^\top \hat{x}$. Thus, finding the longest path in a selection diagram is equivalent to finding the solution $\hat{x} \in \mathcal{X}$ that yields the highest objective value for the follower.

Example 4.6. *The selection diagram for the KPP instance in Example 4.4 is illustrated in Figure 4.2. Once again, we only keep track of four maximal solutions:* $\{1,2,3\}$*,* $\{1,4\}$*,* $\{2,4\}$*,* $\{3,4\}$*. Note that for a given solution $\hat{x} \in \mathcal{X}$, there can be multiple paths from $\varnothing$ to q. Applying Program (4.7) to Figure 4.2, Constraints (4.7c) and (4.7d) become:*

$$\begin{array}{llll}
y_\varnothing \geq 1 - t_1 + y_1, & y_\varnothing \geq 1 - t_2 + y_2, & y_\varnothing \geq 1 - t_3 + y_3, & y_\varnothing \geq 1 + y_4, \\
y_1 \geq 1 - t_2 + y_{12}, & y_2 \geq 1 - t_1 + y_{12}, & y_3 \geq 1 - t_1 + y_{13}, & y_4 \geq 1 - t_1 + y_{14}, \\
y_1 \geq 1 - t_3 + y_{13}, & y_2 \geq 1 - t_3 + y_{23}, & y_3 \geq 1 - t_2 + y_{23}, & y_4 \geq 1 - t_2 + y_{24}, \\
y_1 \geq 1 + y_{14}, & y_2 \geq 1 + y_{24}, & y_3 \geq 1 + y_{34}, & y_4 \geq 1 - t_3 + y_{34}, \\
y_{12} \geq 1 - t_3 + y_{123}, & y_{13} \geq 1 - t_2 + y_{123}, & y_{23} \geq 1 - t_1 + y_{123}, & \\
y_{14} \geq 0, & y_{24} \geq 0, & y_{34} \geq 0, & y_{123} \geq 0.
\end{array}$$

4.3.1.1. Dynamic Generation. The number of subsets of $\mathcal{I}$ is exponential with respect to $|\mathcal{I}|$, so in practice, enumerating the full set of Constraints (4.7c) and (4.7d), as in Example 4.6, is untractable. Thus, a method to generate the variables and constraints dynamically is required. Furthermore, some popular MILP solvers do not support column generation, so the dynamic generation of the variables y_J is usually not feasible. In this section, we describe an algorithm to solve Program (4.7) using a selection diagram which is updated dynamically only with new constraints (which are often referred to as *lazy constraints*).



Note: *Dashed arcs represent the terminal arcs.*

Figure 4.2 – Selection diagram of Example 4.6.

The first step is to choose a set of nodes $\hat{\mathcal{V}}$ that will be used throughout the procedure. This set of nodes is final, since we do not opt for column generation. Evidently, the initial node $\varnothing$ and the terminal node q must be in $\hat{\mathcal{V}}$. For the remaining nodes, we limit our options to only singletons and pairs, *i.e.* we only consider the nodes J where $|J| \in \{1, 2\}$. Even so, some pairs may not belong to a feasible solution, and the number of all pairs is often too large, hence not every pair should be present in the selection diagram. We propose a sampling algorithm described in Algorithm 4.2. The algorithm iteratively samples a solution $\hat{x} \in \mathcal{X}$, then it randomly selects a pair within $\mathcal{I}(\hat{x})$. After reaching the desired number of pairs, it builds the connections within the first two layers. The end result is an initial selection diagram to be used in Program (4.7).

The single-level reformulation (Program (4.7)) derived from the initial selection diagram is a relaxation of the one derived from the full selection diagram. Indeed, we did not connect any node to the terminal node q. As a result, the value of $y_\varnothing$ is not "grounded" and the dual representation has no effect at the beginning. Over the course of the procedure, we introduce new paths by connecting nonterminal nodes J to q using a *modified terminal arc* $\hat{a}_J$. As long as the paths that we add to the diagram correspond to feasible solutions in $\mathcal{X}$, the reformulation remains valid. Such solutions are obtained by solving the follower's problem similarly to Algorithm 4.1. The whole process is described in Algorithm 4.3.

Example 4.7. *Consider the KPP instance introduced in Example 4.4. At line 4 in Algorithm 4.2, instead of sampling a random solution, we once again exploit monotonicity (Remark 4.3) and sample only maximal solutions in $\mathcal{X}$. Suppose that we sample the two*

Algorithm 4.2 Generation of the initial selection diagram

Input: The CPP instance, the desired number of pairs N.

Output: The set of nodes $\hat{\mathcal{V}}$ and the set of initial arcs $\hat{\mathcal{A}}$.

```
 1: ▷ Initial nodes ◁
 2: $\hat{\mathcal{V}}_0 \leftarrow \{\varnothing\}$, $\hat{\mathcal{V}}_1 \leftarrow \varnothing$, $\hat{\mathcal{V}}_2 \leftarrow \varnothing$
 3: for $n = 1, \dots, N$ do
 4:     Sample $\hat{x} \in \mathcal{X}$
 5:     Sample a random pair $\{i, j\}$ within $\mathcal{I}(\hat{x})$
 6:     $\hat{\mathcal{V}}_1 \leftarrow \hat{\mathcal{V}}_1 \cup \{i, j\}$        ▷ Set of singletons
 7:     $\hat{\mathcal{V}}_2 \leftarrow \hat{\mathcal{V}}_2 \cup \{\{i, j\}\}$        ▷ Set of pairs
 8: $\hat{\mathcal{V}} \leftarrow \hat{\mathcal{V}}_0 \cup \hat{\mathcal{V}}_1 \cup \hat{\mathcal{V}}_2 \cup \{q\}$
 9: ▷ Initial arcs ◁
10: $\hat{\mathcal{A}} \leftarrow \varnothing$
11: for $k = 1,2$ do
12:     for all $J \in \hat{\mathcal{V}}_{k-1}$, $K \in \hat{\mathcal{V}}_k$ do
13:         if $J \subseteq K$ then
14:             Create an arc $a_{JK}$ from $J$ to $K$
15:             $I(a_{JK}) \leftarrow K \setminus J$
16:             $\hat{\mathcal{A}} \leftarrow \hat{\mathcal{A}} \cup \{a_{JK}\}$
17: return $(\hat{\mathcal{V}}, \hat{\mathcal{A}})$
```

pairs {1,2} *and* {2,4} *in line 5. The initial selection diagram returned by Algorithm 4.2 is drawn in Figure 4.3 (consider only the solid arcs).*

The dashed arcs in Figure 4.3 represent the modified terminal arcs added by Algorithm 4.3 corresponding to the four maximal solutions. The two solutions {1,2,3} *and* {2,4} *contain some pairs in the initial diagram, hence we can connect those pairs to q (which is the best case). There is no pair contained in the other two solutions* {1,4} *and* {3,4}*, so we fall back to singletons. Note that some solutions like* {1,4} *can be introduced in multiple ways (from either state* {1} *or* {4}*). In that case, we choose a random node by iterating $\hat{\mathcal{V}}_k$ in a random order (line 9 in Algorithm 4.3). In the worst case, we can add an arc from $\varnothing$ directly to q, which is equivalent to a value function constraint. Thus, we can always introduce any feasible solution into the diagram.*

4.3.2. Decision Diagram

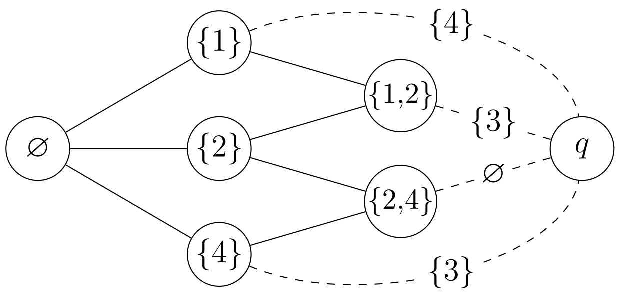
A family of dynamic programming models that is commonly used in the literature is *decision diagrams* [3, 43]. In a decision diagram, each step represents the decision to include

Algorithm 4.3 Solution of Program (4.7) using selection diagram with dynamically generated constraints

```
Input: The CPP instance, the initial selection diagram $(\hat{\mathcal{V}}, \hat{\mathcal{A}})$.
Output: The optimal solution $(t^*, x^*)$ of the CPP instance.
 1: Create the master problem from Program (4.7) using the diagram $(\hat{\mathcal{V}}, \hat{\mathcal{A}})$.
 2: loop
 3:     Solve the master problem and let $(\tilde{t}, \tilde{x})$ be the current optimal solution.
 4:     Solve Program (4.1) with $t = \tilde{t}$ for $\hat{x}$.          ▷ i.e. choose any $\hat{x} \in \mathbf{R}(\tilde{t})$
 5:     if $(v - \tilde{t})^\top \tilde{x} < (v - \tilde{t})^\top \hat{x}$ then
 6:         ▷ $(\tilde{t},\tilde{x})$ is not bilevel feasible. Add new constraint.          ◁
 7:         added ← false
 8:         for $k = 2,1,0$ do
 9:             for all $J \in \hat{\mathcal{V}}_k$ do          ▷ Iterate in a random order
10:                 if $J \subseteq \mathcal{I}(\hat{x})$ then
11:                     Create an arc $\hat{a}_J$ from $J$ to $q$
12:                     $I(\hat{a}_J) \leftarrow \mathcal{I}(\hat{x}) \setminus J$
13:                     $\hat{\mathcal{A}} \leftarrow \hat{\mathcal{A}} \cup \{\hat{a}_J\}$
14:                     Add $y_J \geq \sum_{i \in I(\hat{a}_J)} (v_i - t_i)$ to the master program.
15:                     added ← true
16:                     break
17:             if added then
18:                 break          ▷ Break the for loop, go back to the outer loop
19:     else
20:         ▷ $(\tilde{t},\tilde{x})$ is bilevel feasible. Stop.          ◁
21:         $(t^*, x^*) \leftarrow (\tilde{t}, \tilde{x})$
22:         break
23: return $(t^*, x^*)$
```

or exclude a specific item. Let $n = |\mathcal{I}|$ and let $(i_1, i_2, \dots, i_n)$ be an ordering of the items in $\mathcal{I}$. A *proper node* (neither p nor q) in a decision diagram is a tuple (k, s_k) of a step $k \in \{1, \dots, n-1\}$ and a state s_k. The definition of the states s_k is problem-specific. We partition $\mathcal{V}$ into $n+1$ layers: $\mathcal{V}_0$ which contains only p; $\mathcal{V}_k$ which contains all nodes (k, s_k) for $k = 1, \dots, n-1$; and $\mathcal{V}_n$ which contains only q.

For $k = 0, \dots, n-1$, a node $u_k \in \mathcal{V}_k$ has at most two outgoing arcs $a^0_{u_k}$ and $a^1_{u_k}$, corresponding to the decisions $x_{i_{k+1}} = 0$ and $x_{i_{k+1}} = 1$. The targets of these arcs are in the



Note: Solid arcs are the initial arcs, while dashed arcs represent the modified terminal arcs added during Algorithm 4.3. The labels of the dashed arcs are $I(\hat{a}_J)$.

Figure 4.3 – Dynamically generated selection diagram of Example 4.7.

next layer $\mathcal{V}_{k+1}$. The item function is defined as follows:

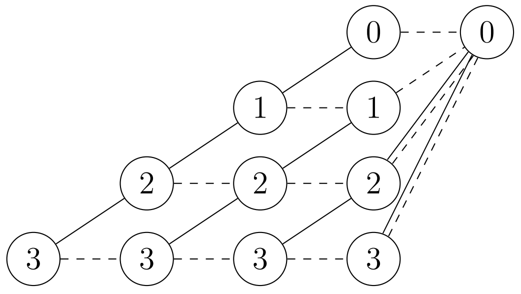
$$I(a^0_{u_k}) = \varnothing, \qquad\qquad I(a^1_{u_k}) = \{i_{k+1}\}.$$

A path in a decision diagram represents a series of decisions $(x_{i_1}, \ldots, x_{i_n})$, fixing the choice of one item at every step. At the end of the path, all entries in x have been fixed and the value of x should correspond to a feasible solution $\hat{x} \in \mathcal{X}$ given a suitable description of the nodes. In this paper, we provide the descriptions of the decision diagrams for three specific problems: the knapsack problem (Example 4.8), the maximum stable set problem (Section 4.3.2.3), and the minimum set cover problem (Section 4.3.2.4). The descriptions of the decision diagrams for several other problems can be found in [3].

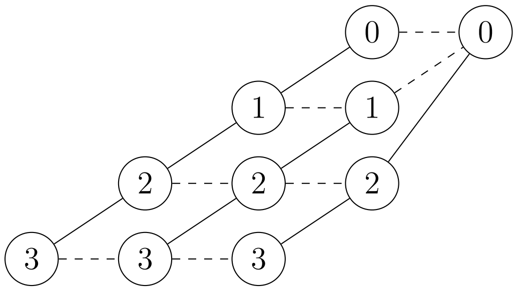
Example 4.8. *In the decision diagram of the knapsack problem, we define the state s_k at step k as the remaining capacity after $x_{i_1}, \ldots, x_{i_k}$ have been decided. Furthermore, we express the initial node p as $(0, C)$ and the terminal node q as $(n, 0)$. The arc $a^0_{u_k}$ is always available and it sends (k, s_k) to $(k+1, s_k)$. The arc $a^1_{u_k}$ only exists if $s_k \geq w_{i_{k+1}}$,* i.e. *there is enough remaining capacity to fit the item i_{k+1}. In that case, it connects (k, s_k) to $(k+1, s_k - w_{i_{k+1}})$. An exception is that all arcs from the second-to-last layer $\mathcal{V}_{n-1}$ end at the terminal node instead.*

The decision diagram for the KPP instance in Example 4.4 is illustrated in Figure 4.4a. We can use Remark 4.3 to simplify the graph. At step k, if the state s_k satisfies $s_k \geq \sum_{j=k+1}^{n} w_{i_j}$, then we must set $x_{i_j} = 1$ for all $j > k$ in order to have a maximal solution. Thus, for each k, we can merge all states s_k such that $s_k \geq \sum_{j=k+1}^{n} w_{i_j}$. Furthermore, if $a^0_{u_k}$ and $a^1_{u_k}$ end at the same node, we can eliminate $a^0_{u_k}$ since including the item i_k is prioritized. The final result is shown in Figure 4.4b. Applying Program (4.7) to Figure 4.4b, *Constraints (4.7c) and (4.7d) become (note that $y_p = y_{0,3}$):*

$$\begin{aligned}
&y_{0,3} \geq y_{1,3}, && y_{0,3} \geq 1 - t_1 + y_{1,2}, && y_{2,1} \geq y_{3,1}, && y_{2,1} \geq 1 - t_3 + y_{3,0},\\
&y_{1,2} \geq y_{2,2}, && y_{1,2} \geq 1 - t_2 + y_{2,1}, && y_{2,2} \geq y_{3,2}, && y_{2,2} \geq 1 - t_3 + y_{3,1},\\
&y_{1,3} \geq y_{2,3}, && y_{1,3} \geq 1 - t_2 + y_{2,2}, && y_{3,0} \geq 0, && y_{2,3} \geq 1 - t_3 + y_{3,2},\\
& && && y_{3,1} \geq 0, && y_{3,2} \geq 1.
\end{aligned}$$



(a) Full



(b) Simplified

Note: The labels of the nodes are the states s_k. The steps of the nodes are implied from left to right as $k = 0,\ldots,4$. Solid arcs and dashed arcs represent the decisions $x_k = 1$ and $x_k = 0$, respectively.

Figure 4.4 – Decision diagrams of Example 4.8.

4.3.2.1. Dynamic Generation. Similar to selection diagrams, the number of nodes in a decision diagram is exponential on $|\mathcal{I}|$. Thus, we also wish to solve Program (4.7) using a dynamically generated decision diagram. Algorithm 4.4 describes a sampling algorithm that returns an initial decision diagram. The size of the initial decision diagram is controlled by a parameter W which is called the *width* of the diagram. As the name suggests, it holds that $|\hat{\mathcal{V}}_k| \leq W$ for all $k = 1,\ldots,n-1$. The algorithm samples exactly W solutions $\hat{x} \in \mathcal{X}$ (line 4), finds a path in the full diagram for each $\hat{x}$ (line 5), then incorporates those paths into the initial diagram (lines 6 and 7). The path-finding step is problem-specific, but usually, it can be done quite efficiently without explicitly enumerating the full diagram.

Updating a decision diagram is more complex than updating a selection diagram. We propose a method to append a path corresponding to a new solution $\hat{x} \in \mathcal{X}$ using as many nodes in $\hat{\mathcal{V}}$ as possible. Let $\mathcal{B}$ be a superset of $\mathcal{A}$ consisting of all *valid transitions* of the decision diagram, which is problem-specific. The longest path in $(\mathcal{V}, \mathcal{B})$ must have the same length as the longest path in $(\mathcal{V}, \mathcal{A})$. Unlike $\mathcal{A}$, for $b \in \mathcal{B}$, $u(b)$ and $w(b)$ need not be in consecutive layers, and $I(b)$ can have more than one item. For instance, $\mathcal{B}$ can include, but is not limited to, the aggregations of path segments in $\mathcal{A}$. The testing of membership in $\mathcal{B}$ should be fast and should not require the enumeration of $\mathcal{B}$.

Algorithm 4.4 Generation of the initial decision diagram

Input: The CPP instance, the desired width W.

Output: The set of nodes $\hat{\mathcal{V}}$ and the set of initial arcs $\hat{\mathcal{A}}$.

```
1: $\hat{\mathcal{V}} \leftarrow \varnothing$                                  ▷ Initial nodes
2: $\hat{\mathcal{A}} \leftarrow \varnothing$                                  ▷ Initial arcs
3: for $\omega = 1, \dots, W$ do
4:     Sample $\hat{x} \in \mathcal{X}$
5:     Find a path $\pi$ corresponding to $\hat{x}$ in the full decision diagram
6:     $\hat{\mathcal{V}} \leftarrow \hat{\mathcal{V}} \cup \mathcal{V}(\pi)$          ▷ $\mathcal{V}(\pi)$ is the set of nodes in $\pi$
7:     $\hat{\mathcal{A}} \leftarrow \hat{\mathcal{A}} \cup \mathcal{A}(\pi)$          ▷ $\mathcal{A}(\pi)$ is the set of arcs in $\pi$
8: return ($\hat{\mathcal{V}}$, $\hat{\mathcal{A}}$)
```

We partition $\hat{\mathcal{V}}$ into layers $\hat{\mathcal{V}}_0, \dots, \hat{\mathcal{V}}_n$ such that $\hat{\mathcal{V}}_k \subseteq \mathcal{V}_k$ for all $k = 0, \dots, n$. Given a solution $\hat{x} \in \mathcal{X}$ that we wish to introduce into the diagram, consider the subset:

$$\hat{\mathcal{B}}(\hat{x}) = \{b \in \mathcal{B} \mid u(b) \in \hat{\mathcal{V}}_j,\, w(b) \in \hat{\mathcal{V}}_k,\, I(b) = \mathcal{I}_{jk}(\hat{x}) \text{ for some } j < k\}$$

where

$$\mathcal{I}_{jk}(\hat{x}) = \mathcal{I}(\hat{x}) \cap \{i_{j+1}, \dots, i_k\}.$$

Enumerating $\hat{\mathcal{B}}(\hat{x})$ takes $\mathcal{O}(|\hat{\mathcal{V}}|^2)$-time assuming that the testing of membership in $\mathcal{B}$ takes constant time (a loop for $u(b) \in \hat{\mathcal{V}}_j$ and an inner loop for $w(b) \in \hat{\mathcal{V}}_k$). By construction, every path in the graph $(\hat{\mathcal{V}}, \hat{\mathcal{B}}(\hat{x}))$ corresponds to $\hat{x}$. We then find the longest path in $(\hat{\mathcal{V}}, \hat{\mathcal{B}}(\hat{x}))$ where the length of all arcs is exactly 1. Finally, we introduce this path as a set of new constraints. The whole process is outlined in Algorithm 4.5.

Example 4.9. *In the KPP, we can derive a path corresponding to a given $\hat{x} \in \mathcal{X}$ in the full diagram (line 5 in Algorithm 4.4) by keeping track of the remaining capacity at step k, defined by the following recursive equations:*

$$s_0 = C,$$
$$s_k = \begin{cases} s_{k-1} - w_{i_k} & \text{if } i_k \in \mathcal{I}(\hat{x}), \\ s_{k-1} & \text{if } i_k \notin \mathcal{I}(\hat{x}), \end{cases} \qquad \text{for } k = 1, \dots, n-1.$$

The nodes on the path are $p, (1, s_1), \dots, (n-1, s_{n-1}), q$. Applying Algorithm 4.4 to the KPP instance in Example 4.4 with two sampled solutions $\{1,2,3\}$ and $\{3,4\}$, we have the initial decision diagram drawn in Figure 4.5 (consider only solid and dashed arcs).

Algorithm 4.5 Solution of Program (4.7) using decision diagram with dynamically generated constraints

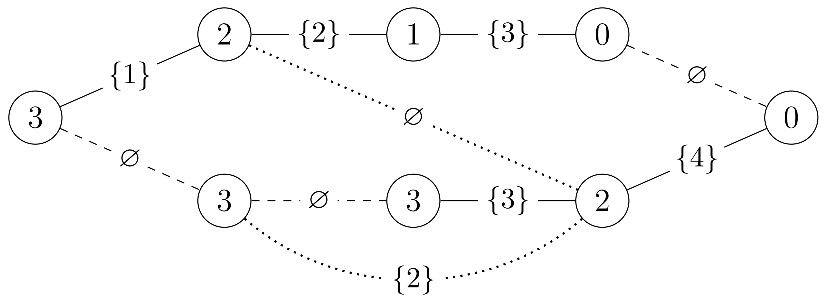
```
Input: The CPP instance, the initial decision diagram $(\hat{\mathcal{V}}, \hat{\mathcal{A}})$.
Output: The optimal solution $(t^*, x^*)$ of the CPP instance.
 1: Create the master problem from Program (4.7) using the diagram $(\hat{\mathcal{V}}, \hat{\mathcal{A}})$.
 2: loop
 3:     Solve the master problem and let $(\tilde{t}, \tilde{x})$ be the current optimal solution.
 4:     Solve Program (4.1) with $t = \tilde{t}$ for $\hat{x}$.        ▷ i.e. choose any $\hat{x} \in \mathbf{R}(\tilde{t})$
 5:     if $(v - \tilde{t})^\top \tilde{x} < (v - \tilde{t})^\top \hat{x}$ then
 6:         ▷ $(\tilde{t},\tilde{x})$ is not bilevel feasible. Add new constraint.        ◁
 7:         Enumerate $\hat{\mathcal{B}}(\hat{x})$.
 8:         Find the longest path $\pi$ in $(\hat{\mathcal{V}}, \hat{\mathcal{B}}(\hat{x}))$ where all arcs have length 1.
 9:         $\hat{\mathcal{A}} \leftarrow \hat{\mathcal{A}} \cup \mathcal{A}(\pi)$        ▷ $\mathcal{A}(\pi)$ is the set of arcs in $\pi$
10:         for $a \in \mathcal{A}(\pi)$ do
11:             Add $y_{u(a)} \geq \sum_{i \in I(a)} (v_i - t_i) + y_{w(a)}$ to the master program.
12:     else
13:         ▷ $(\tilde{t},\tilde{x})$ is bilevel feasible. Stop.        ◁
14:         $(t^*, x^*) \leftarrow (\tilde{t}, \tilde{x})$
15:         break
16: return $(t^*, x^*)$
```

We define the set of valid transitions $\mathcal{B}$ for the KPP to be:

$$\mathcal{B} = \left\{ b \;\middle|\; \begin{array}{ll} u(b) = (j, s_j) \in \mathcal{V}_j, & j = 0, \ldots, n-1; \\ w(b) = (k, s_k) \in \mathcal{V}_k, & k = j+1, \ldots, n; \\ s_j - \sum_{i \in I(b)} w_i \geq s_k & \end{array} \right\}.$$

In other words, a transition b from (j, s_j) to (k, s_k) is valid if and only if all items in $I(b)$ can be fit in a knapsack with capacity $s_j - s_k$. Recall that p and q can be expressed as $(0, C)$ and $(n, 0)$. It is evident that given $u(b)$, $w(b)$, and $I(b)$, this condition can be tested efficiently. If we follow Algorithm 4.5 and add the paths corresponding to $\{1,4\}$ and $\{2,4\}$, we will obtain the dotted arcs in Figure 4.5.

4.3.2.2. Item Grouping. A major disadvantage of decision diagrams compared to selection diagrams is the lack of scalability. For every new item, we need to add a whole new layer to the decision diagram, while the number of layers in a selection diagram stays at 4. Furthermore, many nodes just repeat the states of some nodes in the previous layers (see Figure 4.4a). In this section, we describe a technique to reduce the number of nodes in a



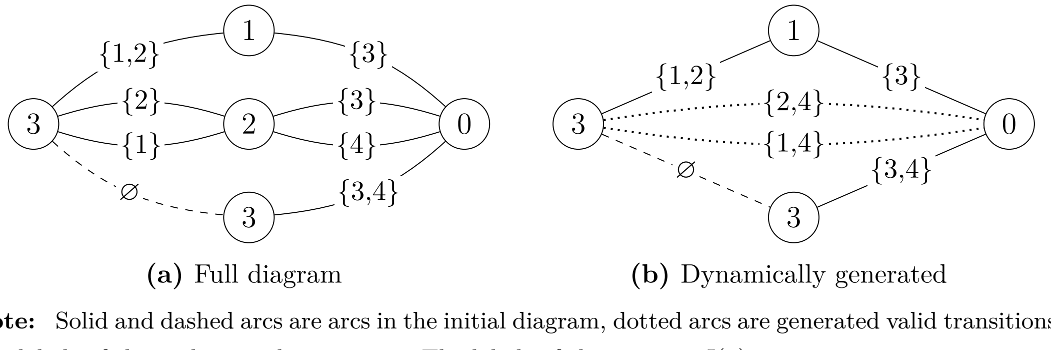
Note: Solid and dashed arcs are the initial arcs obtained from Figure 4.4b, while dotted arcs are valid transitions added during Algorithm 4.5. The labels of the nodes are the states s_k. The labels of the arcs are $I(a)$.

Figure 4.5 – Dynamically generated decision diagram of Example 4.9.

decision diagram by grouping multiple items into one layer, instead of keeping the item-layer ratio at 1:1.

Let $(J_1, \dots, J_m)$ be a list of disjoint subsets of $\mathcal{I}$ such that $J_1 \cup \dots \cup J_m = \mathcal{I}$. We group all items in each *part* J_k together into one layer, so overall, the grouped decision diagram has $m + 1$ layers $\mathcal{V}_0, \mathcal{V}_1, \dots, \mathcal{V}_m$. For $k = 0, \dots, m - 1$, each node $u_k \in \mathcal{V}_k$ can have up to $2^{|J_{k+1}|}$ outgoing arcs towards the next layer $\mathcal{V}_{k+1}$. Each arc $a^K_{u_k}$ corresponds to a subset K of J_{k+1} with the item function simply defined as $I(a^K_{u_k}) = K$. Such an arc represents the decision of fixing $x_i = 1$ for all $i \in K$, and fixing $x_i = 0$ for all $i \in J_{k+1} \setminus K$. A path in the grouped diagram still represents a solution $\hat{x} \in \mathcal{X}$ because $\{J_1, \dots, J_m\}$ forms a partition of $\mathcal{I}$. Hence, all entries in x are fixed and no entry is fixed twice. Algorithms 4.4 and 4.5 are extended accordingly.

Example 4.10. *Consider the decision diagram in Figure 4.4b. We group the first two layers into one group $J_1 = \{1,2\}$ and the last two layers into another group $J_2 = \{3,4\}$. The result is shown in Figure 4.6a. The grouped version of Figure 4.5 is illustrated in Figure 4.6b.*



Note: Solid and dashed arcs are arcs in the initial diagram, dotted arcs are generated valid transitions. The labels of the nodes are the states s_k. The labels of the arcs are $I(a)$.

Figure 4.6 – Grouped decision diagrams of Example 4.10.

In our experiments, we keep the number of layers m fixed and we partition the items randomly into groups of size $\lceil n/m \rceil$. This strategy allows us to maintain a constant number of nodes $|\hat{\mathcal{V}}|$ in the initial decision diagram as n increases. We remark that this technique should be interpreted as a general framework, and one can develop new strategies depending on the situation, including groups with different sizes, subgroups inside a group, separate groups for tolled items and toll-free items, etc.

4.3.2.3. Maximum Stable Set Pricing Problem. We consider the case when the follower's problem is a maximum stable set problem. The overall pricing problem is called the *maximum stable set pricing problem* (MaxSSPP). Let $\mathcal{G} = (\mathcal{I}, \mathcal{E})$ be an undirected simple graph. In this case, the set of items $\mathcal{I}$ is the set of nodes of $\mathcal{G}$. The objective of the follower is to find a *stable set* (*i.e.* a subset of $\mathcal{I}$ in which no pair of nodes are adjacent) that maximizes the sum of tolled values. If $A \in \{0,1\}^{\mathcal{I}\times\mathcal{E}}$ is the node-edge incidence matrix of $\mathcal{G}$, then the follower's problem of the MaxSSPP is formulated as follows:

$$\mathbf{R}(t) = \arg\max_x \{(v-t)^\top x \mid A^\top x \leq \mathbf{1},\, x \in \{0,1\}^{\mathcal{I}}\}$$

where $\mathbf{1}$ is the vector of all ones.

In the decision diagram for the MaxSSPP, the state s_k at step k is the set of all items in $\{i_{k+1}, \ldots, i_n\}$ that are still available, *i.e.* they are not adjacent to any already-selected item. The arc $a^0_{u_k}$ is always available and it sends (k, s_k) to $(k+1, s_k \setminus \{i_{k+1}\})$. The arc $a^1_{u_k}$ only exists if $i_{k+1} \in s_k$ (meaning the item i_{k+1} is available). In that case, it sends (k, s_k) to $(k+1, s_k \setminus N[i_{k+1}])$ where $N[i]$ is the closed neighborhood of i (*i.e.* the set that includes i and all nodes adjacent to i). We express p as $(0, \mathcal{I})$ and q as $(n, \varnothing)$.

In regard to Algorithm 4.4, given $\hat{x} \in \mathcal{X}$, we can generate a path corresponding to $\hat{x}$ by the following recursive system:

$$s_0 = \mathcal{I},$$

$$s_k = \begin{cases} s_{k-1} \setminus N[i_k] & \text{if } i_k \in \mathcal{I}(\hat{x}), \\ s_{k-1} \setminus \{i_k\} & \text{if } i_k \notin \mathcal{I}(\hat{x}), \end{cases} \qquad \text{for } k = 1, \ldots, n.$$

The last piece is the definition of $\mathcal{B}$ used in Algorithm 4.5. A transition b from (j, s_j) to (k, s_k) is valid if and only if $I(b)$ is a stable set, $I(b) \subseteq s_j$, and $s_k \subseteq s_j \setminus N[I(b)]$.

Example 4.11. *Consider a MaxSSPP instance of five items whose graph is shown in Figure 4.7. Assuming that the item ordering is* (1,2,3,4,5)*, the initial decision diagram obtained from Algorithm 4.4 with two sampled solutions* {1,3} *and* {2,5} *is drawn in Figure 4.8 (consider only solid and dashed arcs). The dotted arcs in Figure 4.8 represent the valid transitions generated by Algorithm 4.5 corresponding to the two solutions* {1,4} *and* {3,5}*.*

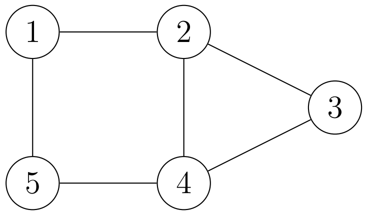
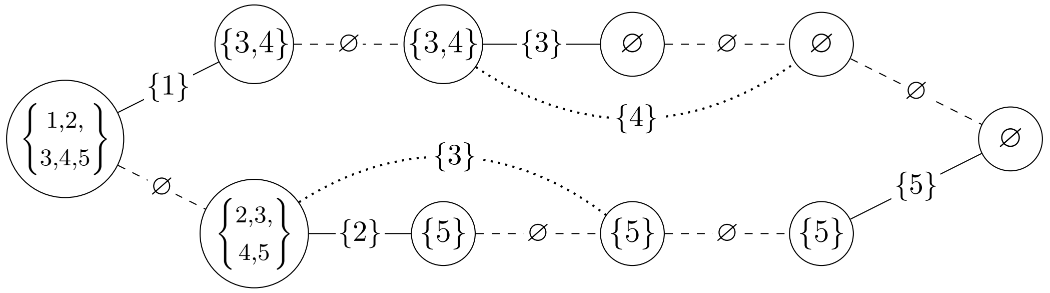


Figure 4.7 – The graph $\mathcal{G}$ of the MaxSSPP instance in Example 4.11.



Note: *Solid and dashed arcs are the initial arcs $a^1_{u_k}$ and $a^0_{u_k}$ respectively, while dotted arcs are generated valid transitions b. The labels of the nodes are the states s_k. The labels of the arcs are $I(a)$.*

Figure 4.8 – Dynamically generated decision diagram of Example 4.11.

4.3.2.4. Minimum Set Cover Pricing Problem. The last pricing problem that we investigate is the *minimum set cover pricing problem* (MinSCPP), in which the follower solves a minimum set cover problem. Let $\mathcal{E}$ be the set of elements that we wish to cover and let $\mathcal{I}$ be a family of subsets of $\mathcal{E}$ that covers $\mathcal{E}$ (*i.e.* the union of all sets in $\mathcal{I}$ is equal to $\mathcal{E}$). For the sake of consistency, we refer to subsets in $\mathcal{I}$ as *items*. The follower aims to find a subset of $\mathcal{I}$ that minimizes the sum of tolled values while still covering $\mathcal{E}$:

$$\mathbf{R}(t) = \arg\min_x \{(v-t)^\top x \mid Ax \geq \mathbf{1},\, x \in \{0,1\}^{\mathcal{I}}\}$$

where $A \in \{0,1\}^{\mathcal{E}\times\mathcal{I}}$ is the incidence matrix between $\mathcal{E}$ and $\mathcal{I}$. We make an assumption that the set of toll-free items $\mathcal{I}_2$ must cover $\mathcal{E}$, otherwise the leader can impose an infinite toll on all tolled items and the pricing problem is unbounded.

The state s_k in the MinSCPP is defined to be the set of uncovered elements of $\mathcal{E}$. In contrast to the first two problems, both arcs $a^0_{u_k}$ and $a^1_{u_k}$ are always available. The former sends (k, s_k) to $(k+1, s_k)$, and the latter sends (k, s_k) to $(k+1, s_k \setminus i_{k+1})$ (recall that i_{k+1} is a subset of $\mathcal{E}$). An exception exists in the next-to-last layer $\mathcal{V}_{n-1}$ to check if the selected items form a cover: if $s_{n-1} \neq \varnothing$ (resp. $s_{n-1} \setminus i_n \neq \varnothing$), then the arc $u^0_{u_{n-1}}$ (resp. $u^1_{u_{n-1}}$) is not available. The nodes p and q are expressed as $(0, \mathcal{E})$ and $(n, \varnothing)$.

Given $\hat{x} \in \mathcal{X}$, a path corresponding to $\hat{x}$ is generated using the recursive system:

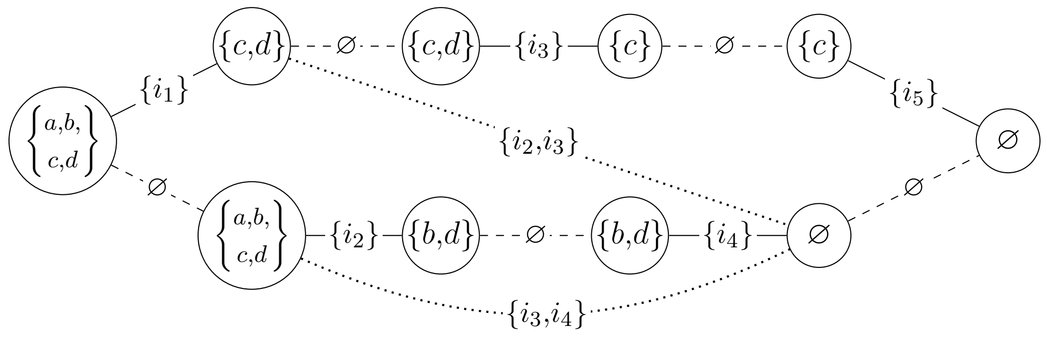
$$s_0 = \mathcal{E},$$

$$s_k = \begin{cases} s_{k-1} \setminus i_k & \text{if } i_k \in \mathcal{I}(\hat{x}), \\ s_{k-1} & \text{if } i_k \notin \mathcal{I}(\hat{x}), \end{cases} \qquad \text{for } k = 1, \ldots, n.$$

Finally, a transition b from (j, s_j) to (k, s_k) is valid if and only if

$$s_j \setminus \Big(\bigcup_{i \in I(b)} i \Big) \subseteq s_k.$$

Example 4.12. *Consider a MinSCPP instance where $\mathcal{E} = \{a,b,c,d\}$ and $\mathcal{I}$ consists of 5 sets $i_1 = \{a,b\}$, $i_2 = \{a,c\}$, $i_3 = \{a,d\}$, $i_4 = \{b,c,d\}$, and $i_5 = \{c\}$. We order the items according to (i_1,i_2,i_3,i_4,i_5). If we sample the solutions $\{i_1,i_3,i_5\}$ and $\{i_2,i_4\}$ for Algorithm 4.4 and then generate valid transitions for $\{i_1,i_2,i_3\}$ and $\{i_3,i_4\}$ in Algorithm 4.5, we will obtain the decision diagram illustrated in Figure 4.9.*



Note: *Solid and dashed arcs are the initial arcs $a^1_{u_k}$ and $a^0_{u_k}$ respectively, while dotted arcs are generated valid transitions b. The labels of the nodes are the states s_k. The labels of the arcs are $I(a)$.*

Figure 4.9 – Dynamically generated decision diagram of Example 4.12.

4.4. Experiments

We conducted numerical experiments to compare the performance of the three dynamic programming models: the value function reformulation (VF), the selection diagram (SD), and the decision diagram (DD). All three models were tested on a family of randomly generated instances belonging to three pricing problems: the knapsack pricing problem (KPP), the maximum stable set pricing problem (MaxSSPP), and the minimum set cover pricing problem (MinSCPP).

To evaluate the gradual impact of the selection diagram and the decision diagram compared to the control case (which is the value function reformulation), we vary the parameters

N in Algorithm 4.2 and W in Algorithm 4.4. Note that if $N = 0$ and $W = 0$, then the programs corresponding to both diagrams become the value function reformulation. To reduce the variance when comparing the results with different values of N and W, we use the same random seed for all sampling procedures (lines 4, 5 in Algorithm 4.2, line 9 in Algorithm 4.5, and line 4 in Algorithm 4.4). For the decision diagram, we also apply item grouping (Section 4.3.2.2) so that the number of layers is at most 20, except for the runs on the KPP where we group every two items together.

Every run was executed on a single thread (AMD EPYC 7532 2.4GHz CPU, 8GB of RAM [1]) with the time limit of one hour. Mixed-integer programs are solved using Gurobi 9.5 [31] while the remaining code (formulation and constraint generation) are written in Julia 1.8.5. The dynamically generated constraints in Algorithms 4.1, 4.3 and 4.5 are implemented as lazy constraints in Gurobi. We reuse the bilevel feasible solutions $(\tilde{t}, \hat{x})$ obtained during constraint generation as feasible solutions of the overall pricing problem. We supplement the solver with these solutions via a custom heuristic.

The result of each pricing problem is presented in Sections 4.4.1 to 4.4.3 respectively. Finally, we provide a summary and our observations in Section 4.4.4.

4.4.1. Knapsack Pricing Problem

We generated random KPP instances that satisfy the following properties:

— The number of items $|\mathcal{I}|$ ranges from 40 to 60.

— The weights w_i are uniformly sampled in the range [1,100].

— The densities v_i/w_i are sampled uniformly in the range $[0.75, 1.25]$. The values v_i are then calculated by multiplying with the random weights.

— The proportion of tolled items $|\mathcal{I}_1|/|\mathcal{I}|$ is equal to the ratio between the capacity and the sum of weights, *i.e.* $C/\sum_{i\in\mathcal{I}} w_i$. We denote this proportion as r. Desirable values for r are from 0.5 to 0.6. The difficulty of the generated instances decreases sharply when r is outside of this range.

— The values v_i of the tolled items are multiplied by 2.0 afterward to encourage the use of tolled items. This technique is adopted from [9].

The parameters listed above are tuned using an evolutionary algorithm to maximize the difficulty of the instances. In total, 330 KPP instances were generated according to 33 different configurations (10 instances per configuration). These 33 configurations are the combinations of 11 values of $|\mathcal{I}| \in \{40, 42, \dots, 58, 60\}$, and three values of $r \in \{0.50, 0.55, 0.60\}$.

1. The experiment was run on the Narval computing cluster provided by the Digital Research Alliance of Canada.

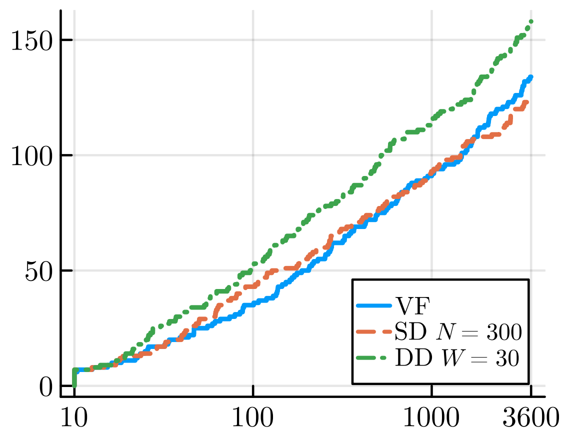
Table 4.1: Numerical results for the KPP.

DP Model	N/W	Num. of instances solved	Time[a] (s)		Gap[b] (%)
			Total	Callback	
VF		**134**	**1301**	**6**	**6.5**
SD	100	118	1316	10	7.6
SD	200	121	1263	10	7.7
SD	**300**	**124**	**1263**	**10**	**7.6**
SD	500	120	1333	9	7.7
SD	700	117	1379	10	7.8
SD	1000	113	1402	10	8.1
SD	1500	118	1465	10	8.0
SD	2000	116	1431	10	7.9
SD	2500	113	1452	10	8.0
SD	3000	114	1487	11	8.2
DD	10	147	973	5	5.7
DD	20	157	994	5	5.5
DD	**30**	**158**	**980**	**6**	**5.4**
DD	50	150	1082	10	5.6
DD	70	144	1155	14	5.6
DD	100	135	1327	19	6.0
DD	150	124	1540	30	6.6
DD	200	113	1735	42	7.1
DD	250	106	1923	53	7.5
DD	300	100	2070	64	7.9

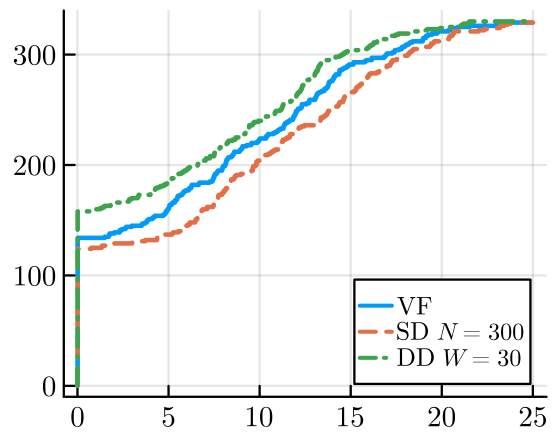
[a] Geometric average.

[b] Arithmetic average.

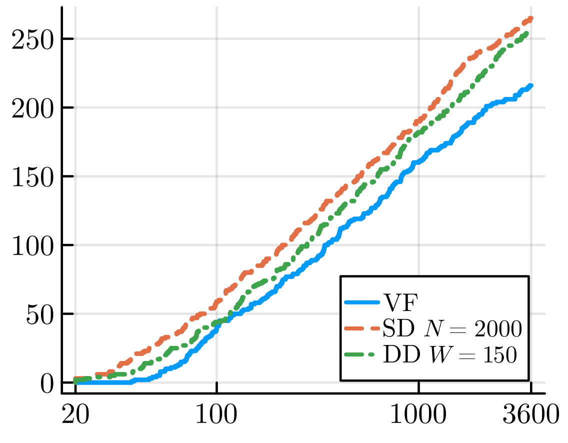
The best configuration of each model is highlighted in bold.



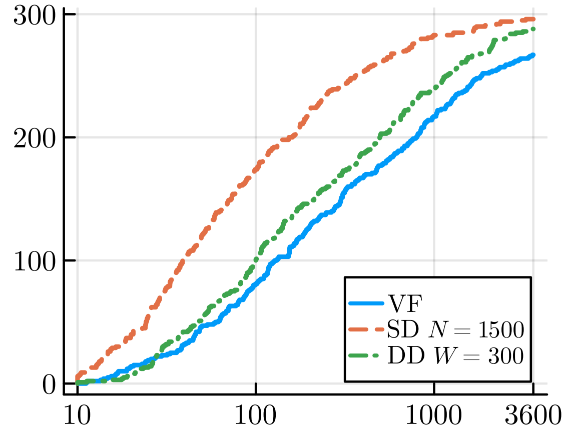
(a) KPP - Time (s)



(b) KPP - Optimality gap (%)



(c) MaxSSPP - Time (s)



(d) MinSCPP - Time (s)

Note: The time is depicted in logarithmic scale, while a linear scale is used for the gap.

Figure 4.10 – Cumulative number of instances (y-axis) with respect to solution time and optimality gap (x-axis).

Table 4.1 shows the number of instances solved, the average time, and the average optimality gap of the three DP models with various values of N and W. The results of the best configurations of each model are plotted in Figures 4.10a and 4.10b. According to Table 4.1, the selection diagram does not improve the performance of the solution compared to the value function model, while the decision diagram does improve it slightly but only at low W (under 100). The knapsack problem is a relatively simple problem, containing only one constraint, so the time to solve the subproblem (included in the callback time) is negligible. The callback time increases when we increase the width W of the decision diagram, reflecting the fact that Algorithm 4.5 scales quadratically with respect to $|\hat{\mathcal{V}}|$.

4.4.2. Maximum Stable Set Pricing Problem

Random MaxSSPP instances were generated on top of random graphs based on the Erdös-Rényi model. The parameters for the generation are as follows:

— The number of items $|\mathcal{I}|$ ranges from 120 to 180.

— The graph density, denoted as d, is from 0.12 to 0.24.

— The proportion of tolled items $|\mathcal{I}_1|/|\mathcal{I}|$ is 0.4.

— The values v_i are uniformly sampled in the range $[50, 150]$, then those of the tolled items are multiplied by 1.3.

We generated 280 instances in total with respect to 28 configurations: 7 values of $|\mathcal{I}| \in \{120,130,\ldots,180\}$ combined with 4 values of $d \in \{0.12, 0.16, 0.20, 0.24\}$.

Table 4.2 and Figure 4.10c show the results of the experiments on the MaxSSPP. The decision diagram still produces a slight improvement and its performance is less sensitive to W. In contrast to the KPP, the selection diagram at high N surpasses the decision diagram. At $N = 2000$, the selection diagram model can solve 50 more instances and is more than 1.5 times faster compared to the value function model. Unlike the KPP, the callback time (which includes the time to solve the subproblem) dominates the total solution time. Thus, both diagrams benefit from the decreases in the number of callback calls and the time per callback call.

4.4.3. Minimum Set Cover Pricing Problem

The generation of MinSCPP instances is more complex compared to the other two problems. First, every element $e \in \mathcal{E}$ is given its own value w_e sampled uniformly in $[50, 85]$. Then, we synthesize the sets $i \in \mathcal{I}$ by including each element in $\mathcal{E}$ with probability 0.23. Special care is required to ensure $\mathcal{I}$ covers $\mathcal{E}$. The value v_i is calculated as the sum of w_e for all $e \in i$, then it is perturbed by a factor sampled within $[0.9, 1.1]$. Next, we find a subcover of $\mathcal{I}$ to form the set of toll-free items $\mathcal{I}_2$ (recall that $\mathcal{I}_2$ must cover $\mathcal{E}$ otherwise the leader can impose an infinite toll). The proportion of tolled items is 0.3. Finally, the values of the tolled items are multiplied by 2.3.

We experimented with six values of $|\mathcal{I}| \in \{70,80,\ldots,120\}$ and five values of $|\mathcal{I}|/|\mathcal{E}| \in \{0.8, 1.0, 1.2, 1.4, 1.6\}$, combined to create 30 configurations, hence 300 random MinSCPP instances.

The results for the MinSCPP are displayed in Table 4.3 and Figure 4.10d. Similar to the MaxSSPP, the callback time plays a major role in the total solution time. The selection diagram greatly outperforms the other two models. At $N = 1500$, it can solve almost all instances and is 3.5 times faster than the value function model. The main contribution is

Table 4.2: Numerical results for the MaxSSPP.

DP Model	N/W	Num. of instances solved	Time[a] (s) Total	Time[a] (s) Callback	Gap[b] (%)	Num. of callback calls[a]	Time per callback call[a] (s)
VF		**216**	**654**	**650**	**15.0**	**202**	**3.22**
SD	100	230	655	650	10.0	201	3.24
SD	200	228	611	606	10.8	194	3.12
SD	300	234	610	604	10.6	202	3.00
SD	500	252	500	493	4.9	191	2.58
SD	700	250	457	449	3.8	185	2.42
SD	1000	257	415	407	3.7	181	2.25
SD	1500	254	393	384	4.3	174	2.20
SD	**2000**	**265**	**389**	**379**	**2.0**	**172**	**2.21**
SD	2500	261	381	371	2.6	168	2.20
SD	3000	259	387	375	2.9	167	2.25
DD	10	251	543	537	4.9	192	2.80
DD	20	247	496	489	4.4	184	2.65
DD	30	250	493	486	4.2	188	2.58
DD	50	252	502	493	4.0	189	2.61
DD	70	255	483	474	4.2	181	2.62
DD	100	253	484	473	4.1	186	2.54
DD	**150**	**257**	**487**	**473**	**3.4**	**181**	**2.61**
DD	200	254	499	481	5.1	176	2.73
DD	250	258	509	487	3.7	175	2.78
DD	300	254	478	456	4.9	169	2.70

[a] Geometric average.

[b] Arithmetic average.

The best configuration of each model is highlighted in bold.

Table 4.3: Numerical results for the MinSCPP.

DP Model	N/W	Num. of instances solved	Time[a] (s) Total	Time[a] (s) Callback	Gap[b] (%)	Num. of callback calls[a]	Time per callback call[a] (s)
VF		**267**	**313**	**302**	**10.1**	**305**	**0.99**
SD	100	282	259	248	7.3	282	0.88
SD	200	283	251	240	6.5	273	0.88
SD	300	283	210	199	5.8	238	0.84
SD	500	283	130	120	3.8	145	0.83
SD	700	292	94	84	1.0	107	0.79
SD	1000	294	88	79	0.4	99	0.80
SD	**1500**	**296**	**89**	**80**	**0.4**	**100**	**0.80**
SD	2000	293	99	89	0.3	110	0.80
SD	2500	292	110	99	0.5	120	0.82
SD	3000	295	129	116	0.3	141	0.82
DD	50	282	244	231	7.1	264	0.88
DD	100	283	232	216	5.3	255	0.85
DD	150	280	227	208	5.7	244	0.85
DD	200	284	228	205	5.4	241	0.85
DD	250	284	221	196	5.0	226	0.87
DD	**300**	**288**	**230**	**203**	**3.5**	**224**	**0.91**

[a] Geometric average.

[b] Arithmetic average.

The best configuration of each model is highlighted in bold.

the drop in the number of callback calls by three times. It also reduces the time spent per callback call by 20%.

4.4.4. Discussions

The experiments on three different problems give us a well-rounded view on the effectiveness of the diagram models. The decision diagram is robust and can provide a small boost in any circumstance compared to the value function reformulation. The selection diagram, however, only performs well in the MaxSSPP and the MinSCPP. The central mechanism of the improvement is due to the reductions in the number of callback calls and the time spent per call. In particular, most of the solution time is spent to solve the subproblem in the MaxSSPP and MinSCPP, while for the KPP, that time is spent mainly on branch-and-bound.

We remark the dissimilarities in the structures of the three problems. In the KPP, any pair of items can appear in a feasible solution (as long as C is large enough). This is not the case for the MaxSSPP, whose main constraint is the exclusion of certain pairs of items. In the MinSCPP, although any pair can also appear in a feasible solution, the minimality of a cover discourages the selection of pair of items that have many overlaps in $\mathcal{E}$. Furthermore, a maximal solution of the KPP contains around 50% of all items. This number is smaller in the other two problems: 20% for the MaxSSPP and 10% for the MinSCPP. The disparity in the distribution of pairs in feasible solutions may explain the unevenness in the performance of the selection diagram across the three problems.

4.5. Conclusion

In this paper, we derived a single-level reformulation of the CPP by taking the dual of a linear program corresponding to a dynamic programming model of the follower's problem, and combining it with the original primal problem using strong duality. We investigated two dynamic programming models: selection diagram and decision diagram, and provided the algorithms to solve them dynamically. Then, we tested these models in three specializations of the CPP: KPP, MaxSSPP, and MinSCPP. We observed that the decision diagram model can consistently lead to performance improvements over the value function reformulation, while the selection diagram formulation only brings (significant) speedups for two specialization of the CPP.

Future research directions include the exploration of other dynamic programming models and the application of this technique to other problems in bilevel optimization.

Acknowledgments. This work was funded by FRQ-IVADO Research Chair in Data Science for Combinatorial Game Theory, and the NSERC grant 2019-04557.

This research was enabled in part by support provided by Calcul Québec (`https://www.calculquebec.ca`) and the Digital Research Alliance of Canada (`https://alliancecan.ca`).

Conclusion

In this thesis, we presented a number of exact solution methods designed to solve the NPP and the CPP, ranging from the formulations and preprocessing algorithms, to the treatment of the multi-commodity variant. The main modeling technique relies on the KKT conditions even in the general case where no trivial dual exists. Meanwhile, preprocessing is done with the help of path enumeration and (strong) bilevel feasibility testing/pruning. Although these methods are novel in theory, they are proven experimentally to be effective in practice.

We enumerate several future research directions spurred from the results of the thesis:

— **Preprocessing based on partial enumeration**: the path-based preprocessing in Section 2.3.2 requires the full set of paths, which may not be available if the NPP scales up in size. Instead of enumerating the full set, we should focus the enumeration effort on eliminating some specific arcs. Then, in the case that the set of all paths is too large, the time spent on enumeration is not wasted.

— **Existence of the complexity asymmetry in other problems**: there may exist other problems similar to the multi-commodity NPP, in which the changes in the followers' reactions follow a certain order, implying an asymmetry in complexity. The identification of these asymmetries, as we saw in Chapter 3, can lead to preprocessing procedures that reduce the solution search space.

— **Extension of the dimensional result in Theorem 3.17**: the theory in Section 3.3.2 is not limited to the NPP and can be applied to any convex polyhedron function.

— **Experiment with different kinds of dynamic programming models**: the formulation in Chapter 4 provides a great level of flexibility in terms of choosing the DP model. Thus, one can design some DP model focusing on a specific property of a given problem, or combine different kinds of diagrams together.

— **Application to general bilinear programs**: the CPP is a special case of general bilinear programs where bilinear terms only exist for each pair of t_i and x_i. Some concepts such as the conjugate model may be transferable to general bilinear programs. Another idea is to rewrite a bilinear program in a form close to the CPP by decoupling the bilinear terms.

Bibliography

[1] A. Aboussoror, S. Adly, and F. E. Saissi. An extended fenchel–lagrange duality approach and optimality conditions for strong bilevel programming problems. *SIAM Journal on Optimization*, 27(2):1230–1255, 2017. doi: 10.1137/16M1080896.

[2] Marc Barthélemy. Spatial networks. *Physics Reports*, 499(1):1–101, 2011. ISSN 0370-1573. doi: 10.1016/j.physrep.2010.11.002.

[3] David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and J. N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016. doi: 10.1287/ijoc.2015.0648.

[4] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 4. Athena scientific, 2012.

[5] Gabriel Bitran and René Caldentey. An overview of pricing models for revenue management. *Manufacturing & Service Operations Management*, 5(3):203–229, 2003. doi: 10.1287/msom.5.3.203.16031.

[6] Mustapha Bouhtou, Stan Hoesel, Anton Kraaij, and Jean-Luc Lutton. Tariff optimization in networks. *INFORMS Journal on Computing*, 19, 08 2007. doi: 10.1287/ijoc.1060.0177.

[7] Patrick Briest, Luciano Gualà, Martin Hoefer, and Carmine Ventre. On stackelberg pricing with computationally bounded customers. *Networks*, 60(1):31–44, 2012. doi: 10.1002/net.20457.

[8] Patrick Briest, Martin Hoefer, and Piotr Krysta. Stackelberg network pricing games. *Algorithmica*, 62(3-4):733–753, 2012. doi: 10.1007/s00453-010-9480-3.

[9] Luce Brotcorne, Martine Labbé, Patrice Marcotte, and Gilles Savard. A bilevel model and solution algorithm for a freight tariff-setting problem. *Transportation Science*, 34 (3):289–302, 2000. doi: 10.1287/trsc.34.3.289.12299.

[10] Luce Brotcorne, Martine Labbé, Patrice Marcotte, and Gilles Savard. A bilevel model for toll optimization on a multicommodity transportation network. *Transportation Science*, 35(4):345–358, 2001. doi: 10.1287/trsc.35.4.345.10433.

[11] Luce Brotcorne, Martine Labbé, Patrice Marcotte, and Gilles Savard. Joint design and pricing on a network. *Operations Research*, 56(5):1104–1115, 2008. doi: 10.1287/opre.1080.0617.

[12] Luce Brotcorne, Fabien Cirinei, Patrice Marcotte, and Gilles Savard. An exact algorithm for the network pricing problem. *Discrete Optimization*, 8(2):246–258, 2011. doi: 10.1016/j.disopt.2010.09.003.

[13] Luce Brotcorne, Fabien Cirinei, Patrice Marcotte, and Gilles Savard. A tabu search algorithm for the network pricing problem. *Computers & Operations Research*, 39(11): 2603–2611, 2012. doi: 10.1016/j.cor.2012.01.005.

[14] Quang Minh Bui, Bernard Gendron, and Margarida Carvalho. A catalog of formulations for the network pricing problem. *INFORMS Journal on Computing*, 34(5):2658–2674, 2022. doi: 10.1287/ijoc.2022.1198.

[15] Samuel Burer. On the copositive representation of binary and continuous nonconvex quadratic programs. *Mathematical Programming*, 120(2):479–495, 2009. doi: 10.1007/s10107-008-0223-z.

[16] Toni Böhnlein, Oliver Schaudt, and Joachim Schauer. Stackelberg packing games. *Theoretical Computer Science*, 943:16–35, 2023. ISSN 0304-3975. doi: 10.1016/j.tcs.2022.12.006.

[17] Alberto Caprara, Margarida Carvalho, Andrea Lodi, and Gerhard J Woeginger. A study on the computational complexity of the bilevel knapsack problem. *SIAM Journal on Optimization*, 24(2):823–838, 2014. doi: 10.1137/130906593.

[18] Jean Cardinal, Erik D Demaine, Samuel Fiorini, Gwenaël Joret, Stefan Langerman, Ilan Newman, and Oren Weimann. The stackelberg minimum spanning tree game. *Algorithmica*, 59:129–144, 2011. doi: 10.1007/s00453-009-9299-y.

[19] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer Programming*. Springer International Publishing, Cham, 2014. ISBN 978-3-319-11008-0. doi: 10.1007/978-3-319-11008-0.

[20] Richard W Cottle, Jong-Shi Pang, and Richard E Stone. *The linear complementarity problem*. SIAM, 2009.

[21] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951.

[22] Scott DeNegre. *Interdiction and discrete bilevel linear programming*. PhD thesis, Lehigh University, 2011.

[23] Sophie Dewez, Martine Labbé, Patrice Marcotte, and Gilles Savard. New formulations and valid inequalities for a bilevel pricing problem. *Operations Research Letters*, 36(2): 141–149, 2008. ISSN 0167-6377. doi: 10.1016/j.orl.2007.03.005.

[24] Mohamed Didi-Biha, Patrice Marcotte, and Gilles Savard. *Path-based formulations of a bilevel toll setting problem*, pages 29–50. Springer US, Boston, MA, 2006. ISBN 978-0-387-34221-4. doi: 10.1007/0-387-34221-4_2.

[25] Wedad Elmaghraby and Pınar Keskinocak. Dynamic pricing in the presence of inventory considerations: Research overview, current practices, and future directions. *Management Science*, 49(10):1287–1309, 2003. doi: 10.1287/mnsc.49.10.1287.17315.

[26] Matteo Fischetti, Ivana Ljubić, Michele Monaci, and Markus Sinnl. A new general-purpose algorithm for mixed-integer bilevel linear programs. *Operations Research*, 65 (6):1615–1637, 2017. doi: 10.1287/opre.2017.1650.

[27] Matteo Fischetti, Ivana Ljubić, Michele Monaci, and Markus Sinnl. Interdiction games and monotonicity, with application to knapsack problems. *INFORMS Journal on Computing*, 31(2):390–410, 2019. doi: 10.1287/ijoc.2018.0831.

[28] Herbert Fleischner, Egbert Mujuni, Daniël Paulusma, and Stefan Szeider. Covering graphs with few complete bipartite subgraphs. *Theoretical Computer Science*, 410(21): 2045–2053, 2009. ISSN 0304-3975. doi: 10.1016/j.tcs.2008.12.059.

[29] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN 0716710455.

[30] François Gilbert, Patrice Marcotte, and Gilles Savard. A numerical study of the logit network pricing problem. *Transportation Science*, 49(3):706–719, 2015. doi: 10.1287/trsc.2014.0560.

[31] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL `https://www.gurobi.com`.

[32] Ward Hanson and Kipp Martin. Optimizing multinomial logit profit functions. *Management Science*, 42(7):992–1003, 1996. doi: 10.1287/mnsc.42.7.992.

[33] G Heilporn, M Labbé, P Marcotte, and G Savard. New formulations and valid inequalities for the toll setting problem. *IFAC Proceedings Volumes*, 39(3):431–436, 2006. doi: 10.3182/20060517-3-FR-2903.00228.

[34] Géraldine Heilporn, Martine Labbé, Patrice Marcotte, and Gilles Savard. A polyhedral study of the network pricing problem with connected toll arcs. *Networks: An International Journal*, 55(3):234–246, 2010. doi: 10.1002/net.2036.

[35] Robert G Jeroslow. The polynomial hierarchy and a simple model for competitive analysis. *Mathematical programming*, 32(2):146–164, 1985. doi: 10.1007/BF01586088.

[36] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 302–311, New York, NY, USA, 1984. Association for Computing Machinery. ISBN 0897911334. doi: 10.1145/800057.808695.

[37] Thomas Kleinert, Martine Labbé, Ivana Ljubić, and Martin Schmidt. A survey on mixed-integer programming techniques in bilevel optimization. *EURO Journal on Computational Optimization*, 9:100007, 2021. ISSN 2192-4406. doi: 10.1016/j.ejco.2021.100007.

[38] H. W. Kuhn and A. W Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492. University of California Press, 1951.

[39] Martine Labbé and Patrice Marcotte. *Bilevel Network Design*, pages 255–281. Springer International Publishing, Cham, 2021. ISBN 978-3-030-64018-7. doi: 10.1007/978-3-030-64018-7_9.

[40] Martine Labbé, Patrice Marcotte, and Gilles Savard. A bilevel model of taxation and its application to optimal highway pricing. *Management Science*, 44(12-part-1):1608–1622, 1998. doi: 10.1287/mnsc.44.12.1608.

[41] Eugene L Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):401–405, 1972. doi: 10.1287/mnsc.18.7.401.

[42] Andrea Lodi, Ted K Ralphs, and Gerhard J Woeginger. Bilevel programming and the separation problem. *Mathematical Programming*, 146(1-2):437–458, 2014.

[43] Leonardo Lozano, David Bergman, and Andre A Cire. Constrained shortest-path reformulations for discrete bilevel and robust optimization. *arXiv preprint arXiv:2206.12962*, 2022.

[44] David G. Luenberger. *Linear and Nonlinear Programming.* Springer International Publishing, 2021. ISBN 978-3-030-85450-8. doi: 10.1007/978-3-030-85450-8.

[45] E. Jerome McCarthy. *Basic Marketing: A Managerial Approach.* Richard D. Irwin, Inc., 1960.

[46] Garth P McCormick. Computability of global solutions to factorable nonconvex programs: Part i—convex underestimating problems. *Mathematical programming*, 10(1): 147–175, 1976. doi: 10.1007/BF01580665.

[47] Maximilian Merkert, Galina Orlinskaya, and Dieter Weninger. An exact projection-based algorithm for bilevel mixed-integer problems with nonlinearities. *Journal of Global Optimization*, 84(3):607–650, 2022. doi: 10.1007/s10898-022-01172-w.

[48] Robert R Meyer. On the existence of optimal solutions to integer and mixed-integer programming problems. *Mathematical Programming*, 7:223–235, 1974. doi: 10.1007/BF01585518.

[49] James T. Moore and Jonathan F. Bard. The mixed integer linear bilevel programming problem. *Operations Research*, 38(5):911–921, 1990. doi: 10.1287/opre.38.5.911.

[50] Ulrich Pferschy, Gaia Nicosia, Andrea Pacifici, and Joachim Schauer. On the stackelberg knapsack game. *European Journal of Operational Research*, 291(1):18–31, 2021. ISSN 0377-2217. doi: 10.1016/j.ejor.2020.09.007.

[51] Sébastien Roch, Gilles Savard, and Patrice Marcotte. An approximation algorithm for stackelberg network pricing. *Networks: An International Journal*, 46(1):57–67, 2005. doi: 10.1002/net.20074.

[52] R.T. Rockafellar. *Convex Analysis*. Princeton Landmarks in Mathematics and Physics. Princeton University Press, Princeton, N. J., 1970. ISBN 9780691015866.

[53] Heinrich von Stackelberg. *Marktform und gleichgewicht*. Verlag von Julius Springer, 1934.

[54] Sahar Tahernejad, Ted K Ralphs, and Scott T DeNegre. A branch-and-cut algorithm for mixed integer bilevel linear optimization problems and its implementation. *Mathematical Programming Computation*, pages 1–40, 2020. doi: 10.1007/s12532-020-00183-6.

[55] CPM van Hoesel, Anton F van der Kraaij, Carlo Mannino, Gianpaolo Oriolo, and Mustapha Bouhtou. *Polynomial cases of the tarification problem*. METEOR, Maastricht University School of Business and Economics, 2003.

[56] Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.

[57] Jin Y Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971. doi: 10.1287/mnsc.17.11.712.