EXPLOITING STRUCTURE IN MIXED INTEGER PROGRAMMING: BRANCHING, CUTTING PLANES, AND COMPLEXITY

A Dissertation
Presented to
The Academic Faculty

By

Yu Yang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
H. Milton Stewart School of Industrial & Systems Engineering

Georgia Institute of Technology

August 2020

EXPLOITING STRUCTURE IN MIXED INTEGER PROGRAMMING: BRANCHING, CUTTING PLANES, AND COMPLEXITY

Approved by:

Dr. Martin Savelsbergh, Advisor
H. Milton Stewart School of Industrial & Systems Engineering
Georgia Institute of Technology

Dr. Natashia Boland, Co-advisor
H. Milton Stewart School of Industrial & Systems Engineering
Georgia Institute of Technology

Dr. Alan Erera
H. Milton Stewart School of Industrial & Systems Engineering
Georgia Institute of Technology

Dr. Santanu Dey
H. Milton Stewart School of Industrial & Systems Engineering
Georgia Institute of Technology

Dr. Bistra Dilkina
Department of Computer Science, Viterbi School of Engineering
University of Southern California

Date Approved: July 14, 2020

Learning without thought is labor lost; thought without learning is perilous.

Confucius

To my beloved parents Aihua Yang and Aizhen Huang.

ACKNOWLEDGEMENTS

It has been a wonderful four-year study at Georgia Tech. In this fruitful journey, I came to know so many knowledgeable and respectable people, without whom, the completion of this dissertation could not have been possible. Firstly, I would like to express my deepest gratitude to my advisors Professor Martin Savelsbergh and Professor Natashia Boland, who have been extremely nice to offer me countless guidance and encouragement in research. Working with them over the past few years has been a real privilege, which helps to make up my mind to pursue an academic career and become a serious researcher like them. I am sincerely grateful to Professor Alan Erera and Professor Bistra Dilkina for numerous meaningful discussions throughout our collaboration. Many thanks go to Professor Santanu Dey for making being an instructor of the course ISyE 3133-Engineering Optimization an amazing experience and serving as my dissertation committee member. I would also like to give special thanks to Professor George Lan for providing me tremendous help and guidance during the first two years of my Ph.D. study. I would like to thank Professor Martin Savelsbergh, Professor Alan Erera, Professor Santanu Dey and Professor George Lan again for their support for my academic job search.

I am profoundly indebted to my parents for their unconditional love and support, which help me stay determined and go through those hard times. My mother has always been optimistic towards life and constantly encouraged me to see things from a positive perspective. My father is my most consistent source of motivation and perseverance.

Moreover, I am grateful to my officemates Amin Gholami and Zhiqiang Zhou for having many interesting conversations together; my collaborator Yassine Ridouane for those productive discussions; my previous roommates Yuang Chen, Tianyi Liu and Zhe Zhou for the help in life; my fellow Ph.D. students: Mostafa Reisi Gahrooei, Di Wu, Yi Zhou, Can Zhang and Rui Gao for providing various useful suggestions on job search, Yufeng Cao, Junqi Hu and Digvijay Boob for sharing information on job search, Guanyi Wang and

Zhehui Chen for having Chinese food together from time to time, just to name a few.

In addition, I would like to thank my friends Kaiwen You, Zhicai Zhang, Yuxin Xiao, Yajing Cui, Shuaili Liu, and Simeng Zhang for their longlasting friendship. Last but certainly not least, thanks to my wife for not showing up in the first 26 years of my life, with whom this dissertation may not have been finished.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

As a powerful mathematical modeling framework, mixed integer programming has seen many industrial applications in areas such as airline industry, logistics, online advertising, cloud computing, etc. Consequently, research on the efficient solution of mixed integer programs (MIPs) has received significant attention (despite the fact that many of the problems modeled as MIPs are known to be NP-hard). Tremendous algorithmic advances have been achieved and state-of-the-art solvers, such as CPLEX and Gurobi, can now solve previously unsolvable instances in just seconds. However, many instances, especially if the underlying problem has a complex structure and the instances are large, can still take a long time to solve. With unprecedented changes in industry, e.g., due to the rise of the sharing economy, this is almost inevitable, and motivates our research. In our research, we take on the challenge of solving MIPs faster through novel branching schemes and novel cutting planes. Via computational complexity analysis, we also justify the usage of MIPs for solving challenging problems and inspire the design of primal heuristics.

The first part of the dissertation focuses on novel branching schemes. We explore the benefits of multi-variable branching schemes in achieving node efficiency, i.e., producing small sized branch and bound search trees. Furthermore, we show that machine learning (ML) techniques can significantly accelerate the selection of sets of variables to branch on and thus turn multi-variable branching schemes into computationally efficient methods.

We use the 0-1 knapsack problem to illustrative the concepts and benefits of multi-variable branching schemes. We present examples where multi-variable branching has advantages over single-variable branching, and partially characterize situations in which this happens. For a special class of 0-1 knapsack instances from [28], we show an LP based branch-and-bound algorithm employing an appropriately chosen multi-variable branching scheme explores either three or seven nodes while it's proved in [28] that a single-variable branching counterpart must explore exponentially many nodes to arrive at an optimal so-

lution. Furthermore, we investigate the performance of various multi-variable branching schemes for 0-1 knapsack instances computationally and demonstrate their potential.

Given that strong branching (SB) has been shown empirically to yield smaller search trees than other branching schemes, we introduce a multi-variable branching analogy, which we refer to as *generalized strong branching* (GSB). Unfortunately, even though GSB outperforms SB, a straightforward implementation is prohibitively time-consuming. Therefore, we apply extreme gradient boosting to train a model that predicts the ranking of sets of candidate variables. To make a branching decision, it now suffices to collect (computationally cheap) features as input to the trained model to efficiently identify a set of variables to branch on. As a result, we achieve a significant time reduction in making the branching decisions and the learned method is able to solve instances of three well-known classes of optimization problems faster than the default branching strategy of commercial solver CPLEX.

The second part of the dissertation addresses two problems arising in service networks. The first problem of interest is the *service network design problem with in-tree constraints* (SNDPITC). Although in-tree constraints are common in practice, they are rarely considered in the academic literature. We compare the size and strength of three existing Steiner in-tree formulations and introduce three integer programming (IP) formulations for the SNDPITC based on these. We compare the size and strength of linear programming (LP) relaxations of these formulations and provide simple strengthening inequalities. Then we focus on the most compact of the three, the flow-based formulation, and derive six new classes of valid inequalities. We discuss separation challenges and present ideas for separation heuristics. Finally, as a proof-of-concept, we conduct a few numerical experiments to show that violated inequalities can be identified and that they can improve the dual bound.

The other problem of interest is the *equipment balancing* problem for a package express carrier operating multiple equipment types in its service network. The weekly schedule of movements used to transport packages through the service network leads to changes in

equipment inventory at the facility level. We seek to reduce this change by substituting the equipment types initially assigned to the movements. To the best of our knowledge, the underlying optimization problems, i.e., minimizing network-wide equipment imbalance and minimizing the number of substitutions required to achieve minimum network-wide equipment imbalance have not been studied and thus their computational complexity was unknown. We carefully analyze several simplified cases and prove the possibly discouraging, but not surprising, result that they are *NP-hard* in general.

CHAPTER 1

INTRODUCTION AND BACKGROUND

This dissertation is about the use and solution of mixed integer program (MIPs). Therefore, in this chapter, we define what a MIP is and introduce branch and cut, the standard framework for solving a MIP (the framework used in most commercial and open-source solvers). Then we review the branching strategies currently employed in the branch and cut framework, as novel branching strategies for MIPs is the focus of the first part of the dissertation. Next, we introduce two problems arising in service network operations, as effectively using MIPs to solve these problems is the focus of the second part of the dissertation. For branching strategies, our innovation is branching on multiple variables rather than on a single variable, which has been the current standard. For service network operations, we consider a design problem with in-tree constraints and analyze formulations and develop new classes of valid inequalities, and we consider an operational problem with multiple equipment types and provide a computational complexity analysis. We end this chapter with a summary of the contributions of the dissertation.

A MIP is an optimization problem of the form:

$$\begin{aligned} \text{min} \quad & c^T x \\ \text{subject to} \quad & Ax = b, \\ & l \leq x \leq u, \\ & x_j \in \mathbb{Z}, \ \ \forall j \in I, \\ & x_j \in \mathbb{R}, \ \ \forall j \in \bar{I}, \end{aligned}$$

where $N = \{1, \ldots, n\}$, $I \subseteq N$, $\bar{I} = N \setminus I$, $c \in \mathbb{R}^n$, A is a $m \times n$ matrix, $b \in \mathbb{R}^m$, and the decision variables are divided into two sets; the variables in set I are required to take

integer values whereas the variables in the other set $\bar{I}$ can take any continuous values.

Despite the linearity of objective function and constraints, solving a MIP efficiently can be difficult and it has been a long-standing challenge faced by researchers and practitioners. Fortunately, the linear programming relaxation, i.e. allowing all variables to take continuous values, is much more tractable thanks to the simplex method by George Dantzig and it provides a dual bound that helps us decide how far we are from the optimality and when we should stop. As a result, solving linear programs (LPs) lies at the heart of the solution of MIPs.

1.1 Cutting Plane and Branch and Bound Algorithms

Let $P = \{x : Ax = b, l \leq x \leq u, x_j \in \mathbb{Z}, \forall j \in I\}$ be the MIP's feasible region and Conv(P) be its convex hull. Let $\bar{P} = \{x : Ax \leq b, l \leq x \leq u\}$ denote the feasible region of the corresponding LP relaxation where the integrality requirement is dropped. Obviously, Conv$(P) \subseteq \bar{P}$ as a result of the convexity of $\bar{P}$. In view of the fact that the optimal solution $\bar{z}^*$ returned by simplex method is an vertex of $\bar{P}$, it lies outside of Conv(P) if it's not feasible to the original MIP. As shown in Figure 1.1, it can be "cut off" by a linear inequality that is satisfied by all $z \in P$, a so-called valid inequality, but that is violated by $\bar{z}^*$. Such a valid inequality is referred to as a *cutting plane* and the process of finding a violated cutting plane, i.e., a cutting plane that separates the optimal LP solution $\bar{z}^*$ from Conv(P), is called separation. A pure cutting plane method iteratively refines the feasible region by adding valid inequalities obtained via some separation procedure. In the 1950s, Gomory proposed a cutting plane algorithm that converges to an optimal solution in a finite number of steps ([48]). The valid inequalities used, the so-called Gomory cuts, can be efficiently separated. Unfortunately, in the worst case, an exponential number of Gomory's cuts have to be added to obtain an optimal solution. For many other types of cutting planes, even the separation itself is NP-hard, let alone solving the MIP. Therefore, the pure cutting plane method is not viewed as a viable approach to solving MIPs for a long time. It was not

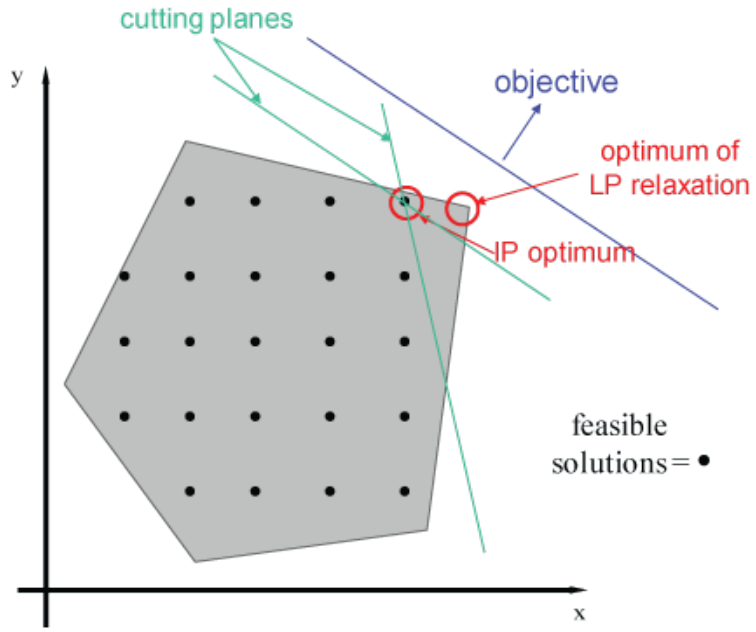


Figure 1.1: An illustration of cutting planes for MIP.

until the 1980s, when cutting planes were incorporated in a branch and bound algorithm for solving instances of the traveling salesman problem (see [31]), that the full potential of cutting planes became clear.

Branch and bound was first proposed in 1960 by Land and Doig in their celebrated work [56] and has grown into a standard algorithmic framework for solving discrete optimization problems. Using bounds on the optimal solution value to avoid searching parts of the solution space that cannot contain an optimal solution, branch and bound can be far more effective than exhaustive search. LP based branch and bound is at the heart of all commercial, and most open-source, solvers, where dual bounds at the nodes in the search tree are obtained by solving the LP relaxation of the MIP at the nodes. A high-level description of the LP based branch and bound algorithm is included in Algorithm 1.

Algorithm 1: LP based based branch and bound algorithm.

Initialization: $\mathcal{L} = \{\text{the original MIP}\}$ and $z_{best} = -\infty$.
while $\mathcal{L} \neq \emptyset$ **do**
  Select and remove a MIP P_i from $\mathcal{L}$.
  Solve the LP relaxation of P_i.
  if *the LP relaxation is feasible* **then**
    Retrieve the LP solution x_i^{LP} and its objective value z_i^{LP}.
    if $z_i^{LP} > z_{best}$ **then**
      if x_i^{LP} *is integer feasible* **then**
        $x_{best} \leftarrow x_i^{LP}$ and $z_{best} \leftarrow z_i^{LP}$
        Remove MIPs P_j from $\mathcal{L}$ with $z_j^{UB} \leq z_{best}$.
      else
        Apply a branching strategy to create two new mixed integer programs $P_{i,1}$ and $P_{i,2}$, set $z_{i,1}^{UP} = z_{i,2}^{UP} = z_i^{LP}$, and add $P_{i,1}$ and $P_{i,2}$ to $\mathcal{L}$.

Similar to the pure cutting plane method, branch and bound is an exact approach that solves MIPs with provable optimality. Nevertheless, it may not be efficient by itself when LP relaxations yield very loose dual bounds, which is very common for MIPs arising in real applications.

Incorporating cutting planes in an LP-based branch and bound algorithm has been shown to be extremely effective for solving many integer programs, especially 0-1 integer programs (see [30] for pioneering work). The resulting algorithm has received its own name: branch and cut. More specifically, when the LP relaxation is solved at the root, cutting planes are added to cut off the solution in case of infeasibility and thus improve the dual bound. In the hope of getting a tighter global dual bound, this process can be repeated several times at the root node before the actual branch and bound procedure starts. Afterwords, cutting planes are periodically added to nodes in the search tree to improve local dual bounds.

1.2 Branching Strategies

This section is devoted to branching strategies, which is the main focus of the first part of this dissertation. Branching is a critical component of the branch and cut framework. We start this section by providing background on existing branching strategies and by motivating our approach to enhancing branching strategies. Then we review literature on single-variable branching schemes, which is the standard in MIP solvers, on multi-variable branching schemes, and on the application of machine learning techniques to assist making branching decisions.

1.2.1 Background and Motivation

As dual bounds at the nodes in the search tree are obtained by solving the LP relaxation of the MIP at the nodes, the branching scheme employed has a significant impact on the effectiveness of the algorithm. A computational study in [6], using CPLEX 12.5, shows that a naive branching scheme, i.e., branch on the most infeasible variable, increases computation time by a factor of 4.5 and number of nodes explored by a factor of 6.5 times compared to the default branching scheme.

The typical branching scheme identifies a variable x_i with fractional value f in the solution to the LP relaxation and creates two branches (i.e., two new nodes in the search tree) one in which the constraint $x_i \geq \lceil f \rceil$ is imposed and one in which the constraint $x_i \leq \lfloor f \rfloor$ is imposed, i.e., branching is based on a *single-variable* dichotomy. As a result, most research on branching schemes (in the context of LP based branch and bound) has focused on the identification of the variable to branch on, e.g., pseudo-cost branching [19], strong branching [11], and reliability branching [5]. Branching based on sets of variables has only been considered in very special cases, e.g., *special ordered set* branching (see [16] and [17]). We note that in [62], they list multi-variable branching schemes as one of the under-explored areas of research that deserves further investigation.

1.2.2 Single-Variable Branching

Over the years, much effort has been devoted to finding schemes for selecting a (single) variable to branch on that result in small search trees. The "desirability" of branching on a particular variable in a single-variable branching scheme has historically been defined by the value $\max\{z^-, z^+\}$, in case of a maximization problem, where z^- and z^+ represent the optimal value of the LP relaxation on the "down" branch and the "up" branch of that variable, respectively, and where a smaller value of $\max\{z^-, z^+\}$ is more desirable.

The simplest scheme is to select the variable x_i with fractional part $f - \lfloor f \rfloor$ closest to 0.5. Let z be the value of the LP solution. The intuition (or hope) is that enforcing $x_i \leq \lfloor f \rfloor$ and enforcing $x_i \geq \lceil f \rceil$ result in noticeable changes to the value of the LP solution, and, therefore, we will have $\max\{z^-, z^+\}$ noticeably less than z, which improves the dual bound. The computational requirements of this naive method are minimal, but, unfortunately, studies have shown that it is not much better than randomly selecting a variable (see [4]).

Rather than hoping that $\max\{z^-, z^+\}$ is noticeably less than z, pseudo-cost branching (PB) (see [19]) observes and records actual changes to the values of the LP relaxations and maintains for each variable x_i an average change and uses these values to choose a variable to branch on. The major weakness of PB is that initially little or no information is available and initial branching decisions, at the top of the search, are critically important. However, PB is computationally cheap, and computational studies have shown that it performs reasonably well.

Instead of hoping or observing and recording, strong branching (SB) (see [11]) goes ahead and computes the change in dual bound for each variable x_i with a fractional value, i.e., it computes the value z_i^- at the "down" child where $x_i \leq \lfloor f_i \rfloor$ is imposed and the value z_i^+ at the "up" child where $x_i \geq \lceil f_i \rceil$ is imposed. SB then chooses the variable x_i for which $\max\{z_i^-, z_i^+\}$ is the smallest. SB results in small search trees, but, unfortunately, is computationally intensive and the benefits of smaller search trees are often undone by the

effort required to select the variable to branch on.

Hybrid strong / pseudo-cost branching tries to overcome the weaknesses of the two methods by applying SB to nodes high in the search tree (i.e., depth up to d) and switching to PB for nodes low in the search tree (i.e., depth more than d). Since d is typically chosen to be relatively small, fractional variables without a pseudo-cost may still be encountered at depth more than d, in which case SB is used to initialize the pseudo-cost ([45, 58]). These ideas have been taken one step further in reliability branching (RB) (see [5]) which uses SB to initialize variables without a pseudo-cost or to re-initialize variables with an unreliable pseudo-cost. Extensive computational experiments have shown that RB performs the best among all aforementioned methods in terms of solution time. Furthermore, several generalizations and alternatives to SB have been considered. For example, the linear scoring rule [58] generalizes SB: it chooses the variable x_i for which $(1-\mu)\max\{z - z_i^-, z - z_i^+\} + \mu \min\{z - z_i^-, z - z_i^+\}$ is the largest, where $\mu \in [0,1]$ is a parameter ($\mu = 1$ gives SB). An alternative, which performs well in practice, is the product scoring rule [2], in which the variable x_i chosen is the one for which the product $(\max\{z - z_i^-, \epsilon\}) \cdot (\max\{z - z_i^+, \epsilon\})$ is the largest, where ϵ is a small positive tolerance. Other rules are discussed in [14], which also provides theoretical and computational insights into the challenges of designing branching rules.

Instead of deciding candidate branching variables purely from one optimal solution of the current LP relaxation, in [20], they take advantage of the multiple alternative optimal solutions and potentially avoids solving unpromising LPs in strong branching. In [3] reliability branching is combined with criteria from conflict analysis widely used in satisfiability problems (SAT) into hybrid branching which solves standard integer programs faster than reliability branching.

1.2.3 Multi-Variable Branching

Two types of special ordered sets (SOS) were introduced in [16] for modeling nonconvex problems. Special order sets of type 1 (SOS1), in which at most one element from an ordered set can have a non-zero value, can be used to model multiple choice problems, while special ordered sets of type 2 (SOS2), in which up to two consecutive elements from an order set can have non-zero values, can be used when a nonlinear function is approximated by a piecewise linear function. Multi-variable branching can be readily applied to variables appearing in special ordered sets. For example, let x_i for $i \in \{1, \ldots, n\}$ be the variables in a special ordered set of type 1, and let $x_0 = x_1 = \ldots x_{k-1} = 0$, $x_k = x_{k+1} = 0.5$, and $x_{k+2} = \ldots, x_n = 0$, then we can branch by imposing $x_i = 0$ for $i = 1, \ldots, k$ on one branch and imposing $x_i = 0$ for $i = k + 1, \ldots, n$ on the other branch. Similar branching ideas can be developed for special ordered sets of type 2. Beyond the well-known SOS1 and SOS2, Linked Ordered Sets (LOS) was introduced in [17]. We note that these multi-variable branching rules target special structures occurring in certain types of optimization problems.

Basis reduction methods, following the groundbreaking work of [57], also give rise to multi-variable branching, often referred to as *hyperplane branching* in that context. (For an introduction to basis reduction methods, see [63], and for an overview of "non-standard" methods for integer programming, see [1].) Ihe potential of hyperplane branching for decomposable knapsack problems (DKPs) is established in [54]. They show that for certain classes of 0-1 knapsack instances a feasibility version can be generated and demonstrate how the infeasibility of this feasibility version can be proven by branching on hyperplanes. Obtaining these hyperplanes, however, involves basis reduction computations which are time-consuming (in spite of their polynomial complexity for fixed dimension). One of the classes of 0-1 knapsack instances considered by [54] are the Chvátal instances. They show that a feasibility version can be generated for instances with an odd number of variables and that therefore infeasibility of these instances can be proven instantly via branching on

hyperplanes. Unfortunately, a feasibility version cannot be generated for the instances with an even number of nodes, which are more the interesting instances, and, therefore, their difficulty cannot be assessed using the [54] framework. Using a traditional, but multi-variable branching perspective, we show how to choose a set S of variables to branch on that will solve all instances (regardless of whether the number of variables is odd or even) evaluating only a few nodes.

1.2.4 Learning to Branch

The recent success of machine learning has inspired new ways to guide the selection of the branching variable. DASH (Dynamic Approach for Switching Heuristics) is introduced in [34], which chooses a branching scheme from a collection of branching schemes to be applied at a node in the search tree based on the features of the integer program at that node. The learning is offline, i.e., it is based on the performance of the branching schemes on a set of training instances. In [14], it is shown how to learn near-optimal parameters in a convex combination of scoring functions from a sample of instances drawn from a distribution over an instance class and provides complexity guarantees on the number of samples needed.

Almost all single-variable branching schemes, pure pseudo-cost branching being the exception, in one way or another rely on SB concepts. SB has been shown to result in small search trees, but is computationally prohibitive. Thus another line of research tries to mimic the behavior of SB via machine learning (ML) techniques in the hope of achieving search trees of small size without the need to spend huge amounts of computing to select a variable to branch on. The idea is to solve many instances using a SB strategy, collecting information throughout the solution process, e.g., features of candidate branching variables, SB scores, ranking of SB scores at a node, and then train a ML model to predict SB scores or a ranking of candidate variables based on their features. In [7] extremely randomized forests are used to train a ML model offline. Their computational study shows

that the learned model is faster than full SB, but that it is outperformed by RB. An online learning variant enhanced by a reliability mechanism is proposed in [61]. The authors observe improvement over (their implementation of) full SB and RB, but it is difficult to tell from the performance profile how significant the improvement is. In [52], an online ML model trained by SVM^{rank} is proposes to predict a ranking of candidate variable by SB scores rather than SB scores themselves. Their computational study shows that using the ML model has some benefits over using PB, but cannot compete with the state-of-the-art implementations of RB, which is the default branching scheme of CPLEX. Finally, in [44], they use imitation learning (using a graph convolutional neural network) to learn SB and report promising results that the resulting branching scheme outperforms the default branching scheme of SCIP, a state-of-the-art open-source solver, on three classes of optimization problems. For a recent survey and more in-depth discussion of learning and branching for MIP, see [59].

1.3 Service Network Operations

The second part of this dissertation is focused on challenging MIPs arising in service network operations. In this section, we provide background information on the two problems considered in this context, and we motivate our specific research efforts.

1.3.1 Background and Motivation

Two of the major challenges faced by logistics companies, such as Fedex and United Parcel Service, are designing service networks that allow efficient and economic daily operations to satisfy their business needs and ensuring their operations are carried out steadily.

To address the first challenge, we consider the *service network design problem with in-tree constraints* (SNDPITC), which tries to find a minimum-cost transportation plan for shipping multiple less-than-truckload (LTL) commodities from their respective origin to their respective destination. In addition, if any commodities sharing the same destination

should meet at any given point in their respective paths, then they must continue on a common path from that point forward.

As is often the case, those companies use a large and heterogeneous pool of trailers and containers in their service (linehaul) networks. To ensure that the right equipment is available at the right location at the right time is part of the second challenge. This is difficult to achieve, in part, because the flow of packages between facilities in the network is not balanced. Another source of imbalance comes from their planning process, where a phased approach is typically applied to reduce the complexity. As a consequence, the companies are forced to move equipment empty, i.e., reposition equipment, which is expensive and motivates research for cheaper ways to restore balance.

1.3.2 Service Network Design With In-Tree Constraints

LTL companies often, in load planning, require the in-tree restriction, which is used to simplify operations at handling terminals (crossdock facilities and the like): dock workers loading trucks need only look at the destination of a commodity to know the next destination it should be loaded to. This restriction implies not only that each commodity follows a single path, but that the paths for commodities having the same destination induce an in-tree rooted at the destination.

Network design problems for freight transportation have been extensively researched, and reviews of modeling and mathematical programming developments can be found in papers such as [29], [55], [71]. Network design problems are usually modeled as a multi-commodity flow problem (MCFP) with some additional design variables (usually integral) and constraints forcing the flows to comply with the design. The variant in which the flow of a commodity must follow a single path, (it cannot be split to be sent on multiple paths), is known as network design with *unsplittable* or *nonbifurcating* flow, and has also been relatively well studied [see 70], as has the case with *splittable* or *bifurcating* flow [see 42]. MCFP with various transport cost functions, including piecewise linear functions [41] and

applications with nonlinear functions [23, 32, 46], has also investigated a lot. A polyhedral study and branch-and-cut algorithm for network design with splittable and unsplittable flow is presented in [13]. A column generation model and branch-and-price-and-cut algorithm for MCFP with unsplittable flows is presented in [15]. A comparison of branch-and-cut algorithms for MCFP with unsplittable flow is presented in [9]. An exact solution methodology for the multicommodity network flow problem with general step cost functions is provided in [43]. A branch-and-cut algorithm that embeds new classes of valid inequalities for an network design problem with unsplittable flow is presented in [18]. A branch-and-cut algorithm with filtering for the multicommodity capacitated fixed charge network design problem is developed in [27]. A dynamic discretization discovery method is designed for solving the scheduled service network design problem in [49].

In contrast, the SNDPITC is relatively unexplored. In [65], they present a local improvement heuristic which manipulates the tree constraint, as well as primal-dual algorithms that provide upper and lower bounds. Similar versions to one of our formulations including time constraints, which are not considered in our work, are presented in [50] and [38] along with heuristics.

1.3.3 Equipment Balancing

We are primarily interested in reducing imbalance resulting from the phased planning process, which goes sequentially as follows. In a flow planning phase, a forecast of daily origin-destination demand is used to determine origin-destination paths for packages that guarantee the service commitments are met and that create consolidation opportunities (consolidation is the primary mechanism a package express carrier employs to reduce/control its operational costs). In a load planning phase, the package flows are converted into loads, i.e., timed movements of equipment through the network. This phase continues to focus primarily on the flow of packages (now in discrete units - by assigning the flows of packages to equipment types), but equipment repositioning decisions are also

made. In a scheduling phase, driver schedules are created to actually move the loads from their origin to their destination directly or through one or multiple relay points. A driver schedule plan typically covers a period of a week and has to satisfy many requirements, e.g., Hours of Service regulations and union contract rules. As considerations related to the equipment pool are only of secondary importance in the above planning phases, the resulting plan (i.e., the plan of loads to be moved in the coming week and the driver schedules to execute this plan) tends to be imbalanced, in the sense that the inventory of the different equipment types at a facility at the start of the week differs from that at the end of the week, which is referred to as the equipment imbalance introduced by (or associated with) the plan. As this may lead to a surplus or a shortage of equipment in the future, the final phase in the planning process seeks to reduce the equipment imbalance introduced by the plan.

The existing literature on equipment management in the trucking industry focuses on the design of empty repositioning strategies to balance equipment (also referred to as "empty vehicle allocation" or "redistribution"), e.g., [36], [22], [37], [60], [33], and [24]. We are not aware of any literature on approaches based on equipment substitution to deal with equipment imbalance in the trucking industry, and thus the complexity of related optimization problem is unknown.

1.4 Main Contributions

The three main contributions are: (1) improved branching strategies for solving (general) integer programs, (2) improved formulations and new classes of valid inequalities for service network design with in-tree constraints, and (3) computational complexity analysis of service network operations with multiple equipment types. Details are provided below.

We consider a multi-variable branching scheme that is a natural generalization of a single-variable branching scheme. Rather than branching on a single variable with a fractional value in the solution to the LP relaxation, branching is performed on a set of vari-

ables such that the sum of their values in the solution to the LP relaxation is fractional. More specifically, a set S of variables with $\sum_{i \in S} x_i = f$ with f fractional is identified and two branches are created, one in which the constraint $\sum_{i \in S} x_i \geq \lceil f \rceil$ is imposed and one in which the constraint $\sum_{i \in S} x_i \leq \lfloor f \rfloor$ is imposed. Chvatal, using a class of 0-1 knapsack instances, demonstrates that branch and bound algorithms employing a single-variable branching scheme may have to explore exponentially many nodes [28]. One of our contributions is to show that there is a branch and bound algorithm employing a multi-variable branching scheme that can solve any instance in this class by exploring either three or seven nodes. That some instances in this class can be solved easily by a multi-variable branching scheme was known, as it follows from investigations of basis reduction methods for decomposable knapsack problems [54], but we are the first to show that *all* instances in this class can be solved easily by a multi-variable branching scheme. Another contribution is that we present and partially characterize situations in which multi-variable branching schemes lead to better dual bounds, i.e., smaller values of $\max\{z^-, z^+\}$, which should lead to smaller search trees. Finally, we demonstrate the potential benefits of multi-variable branching schemes in an extensive computational study in which we assess their relative performance, where our primary measure for the quality of a branching scheme is the number of nodes explored in the search tree, but, at the same time, we seek branching schemes that are computationally efficient.

We propose *generalized strong branching* (GSB) which uses SB scores to select a set of variables to branch on. If only sets of variables of up to size k are considered, this will be denoted as GSB-k. The computational challenges associated with computing SB scores are even greater than the single-variable counterpart since we have far more choices of candidates. To reduce the computational burden, only a small number of candidate sets are considered. Furthermore, we propose a new weighted product function to compute SB scores for candidate sets that makes GSB-2 able to achieve better node efficiency than SB, i.e. the resulting search trees are smaller, which is non-trivial. Nevertheless, GSB-

2 still remains unpractical in view of its significantly longer computation time compared to existing single-variable branching schemes in use. Therefore, we exploit advances in ML to (efficiently) select a good set of variables to branch on and demonstrate that ML techniques can be used to develop branching strategies that are computationally efficient and mimic SB (single variable) and GSB (pair of variables). Moreover, we highlight the importance and benefit of a systematic approach to feature selection. Finally, we show that branching strategies based on learning can outperform the default branching strategy of state-of-the-art commercial solver CPLEX in terms of both the number of nodes explored in the search tree and the time it takes to explore the search tree through extensive numerical experiments.

We approach the SNDPITC from a pure cutting plane perspective. We first compare the size and strength of three popular Steiner in-tree formulations: the *path-only formulation*, the *path-tree formulation* and the *tree-only formulation*. The size is measured by the number of variables and constraints of each formulation and the strength is compared by the feasible region of their corresponding linear programming (LP) relaxation. We conclude that the path-only formulation is weaker than the path-tree and the tree-only formulations, which are proved to be equivalent. Based on them, we subsequently present three integer programming (IP) formulations for the SNDPITC: a *commodity-based formulation*, a *destination-based formulation*, and a *flow-based formulation*. We also compare their sizes and show that the flow-based formulation is significantly more compact than the other two formulations. In terms of strength, we conclude the destination-based formulation is the strongest amongst the three by showing the feasible region of its LP relaxation is strictly contained in those of the other two formulations. Then we present some simple constraints to further strengthen these three formulations. For real applications, the number of commodities $|K|$ can be prohibitively large and the flow-based formulation may be the only viable model since its size is independent of $|K|$. Thus, we are particularly interested in this formulation and try to iteratively tighten it by cutting planes. To this end, we derive six

new classes of cutting planes: the *wheat-stalk inequalities*, the *wheat-sheaf inequalities*, the *combinatorial rounding inequalities*, the *generalized cut-set inequalities*, the *commodity-merging inequalities*, and the *truck-balancing inequalities*. We show by a non-trivial instance that inequalities of these six classes can help improve the dual bound significantly when all simple strengthening inequalities and cut-set inequalities have already been applied. Despite their potential effectiveness, the separation can be challenging. We discuss the difficulty of separation and present ideas for separation heuristics. Finally, as a proof-of-concept, we implement simple separation heuristics for two classes of valid inequalities and demonstrate that violated inequalities are identified and that their addition improves the dual bound.

We introduce a substitution-based equipment balancing problem and develop a MIP formulation. We consider several simplified versions of the underlying optimization problems, i.e., minimizing imbalance and minimizing the number of substitutions required to achieve minimum imbalance. By determining the computational complexity theoretically, we conclude that when the number of equipment types in the network is two, then the two problems of interest can be polynomially solvable. However, when the number of equipment types is at least three, the solution becomes difficult, actually NP-hard. Although the complexity analysis results in somewhat negative results but it provides insights to the source of difficulty and potentially helps design efficient heuristics.

1.5 Remarks

The research presented in Chapter 4 was initiated by Dr. Ira Wheaton Jr. who carried out the analysis of the formulations and was responsible for the basic variant of the wheat-stalk inequality.

The research presented in Chapter 5 of this dissertation was part of a larger effort on developing decision support technology for equipment management at a large U.S. package express carrier based in Atlanta. The material presented in the dissertation represents the

areas of research for which I was responsible. For an overview of the complete research effort, see [74].

CHAPTER 2

MULTI-VARIABLE BRANCHING: A 0-1 KNAPSACK PROBLEM CASE STUDY

In this chapter, we explore the benefit of multi-variable branching schemes focusing on 0-1 knapsack problems of the form:

$$\begin{aligned} \text{max} \quad & \sum_{i=1}^{n} p_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq b, \\ & x_i \in \{0,1\}, \quad \forall\, i \in [n]. \end{aligned}$$

where $[n] = \{1, \ldots, n\}$ and $b \in \mathbb{Z}_{>0}$, $p_i, w_i \in \mathbb{Z}_{>0}$ and $w_i \leq b$ for all $i \in [n]$. We demonstrate the potential of multi-variable branching on a special class of knapsack instances analyzed in [28] and partially characterize the situations when branching on multiples variables does bring benefits, which is further confirmed by a numerical study.

The remainder of this chapter is organized as follows. In Section 2.1, we consider a class of difficult 0-1 knapsack problems used in [28]. In Section 2.2, we give examples showing that branching on sets of variables can be better than branching on a single variable, and derive conditions for branching on sets of two variables to outperform branching on a single variable. In Section 2.3, we report the results of a number of numerical experiments which further reinforce that branching on sets of variables holds promise. Finally, in Section 2.4, we provide some concluding remarks.

2.1 Chvátal Instances and Multi-Variable Branching

We first study the class of instances characterized by the following formulation:

$$\begin{aligned} \text{max} \quad & \sum_{i=1}^{n} a_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} a_i x_i \leq \left\lfloor \sum_{i=1}^{n} a_i/2 \right\rfloor, \\ & x_i \in \{0,1\}, \quad i \in \{1,\ldots,n\}, \end{aligned}$$

where

- $\sum_{i\in I} a_i \leq \sum_{i=1}^{n} a_i/2$ for all sets $I \subseteq \{1,\ldots,n\}$ with $|I| \leq n/10$,
- every integer greater than one divides fewer than $9n/10$ of the integers a_i,
- $\sum_{i\in I} a_i \neq \sum_{j\in J} a_j$, when $I \neq J$, and
- there is no set I such that $\sum_{i\in I} a_i = \lfloor \sum_{i=1}^{n} a_i/2 \rfloor$.

In [28], it is shown that for this class of 0-1 knapsack instances, a branch and bound algorithm employing a singe-variable branching scheme must explore at least $2^{n/10}$ nodes. In [28], a "specific example" with $a_j = 2^{k+n+1} + 2^{k+j} + 1$ for $j = 1,\ldots,n$ and with $k = \lfloor \log(2n) \rfloor$ is also considered. For this class of 0-1 knapsack instances, it can be shown that a branch and bound algorithm employing a singe-variable branching scheme must explore at least $2^{(n-1)/2}$ nodes. We will show, for this class of instances, that a branch and bound algorithm employing an appropriately chosen multi-variable branching scheme solves instances by evaluating at most 7 nodes. We start by observing some properties of the instance data and showing that each instance has a unique optimal solution.

Lemma 2.1.1. *When n is odd, the sum of the largest $\frac{n-1}{2}$ coefficients in the "specific ex-*

ample" is less than b, while the sum of the smallest $\frac{n+1}{2}$ coefficients exceeds b:

$$\sum_{j=\frac{n+1}{2}+1}^{n} a_j < b \qquad \textit{and} \qquad \sum_{j=1}^{\frac{n+1}{2}} a_j > b. \tag{2.1}$$

Proof. For n odd,

$$b := \left\lfloor \sum_{i=1}^{n} a_i/2 \right\rfloor = \left\lfloor \sum_{j=1}^{n} \left(2^{k+n} + 2^{k+j-1} + 1/2\right) \right\rfloor = (n+1)2^{k+n} - 2^k + \frac{n-1}{2}.$$

Summing the largest $\frac{n-1}{2}$ elements, we have

$$\sum_{j=\frac{n+1}{2}+1}^{n} a_j = \sum_{j=\frac{n+1}{2}+1}^{n} \left(2^{k+n+1} + 2^{k+j} + 1\right) = (n+1)2^{k+n} - 2^{k+\frac{n+1}{2}+1} + \frac{n-1}{2} < b. \tag{2.2}$$

Summing the smallest $\frac{n+1}{2}$ elements, we have

$$\sum_{j=1}^{\frac{n+1}{2}} a_j = \sum_{j=1}^{\frac{n+1}{2}} \left(2^{k+n+1} + 2^{k+j} + 1\right) = (n+1)2^{k+n} + 2^{k+\frac{n+1}{2}+1} - 2^{k+1} + \frac{n+1}{2} > b. \tag{2.3}$$

■

Corollary 2.1.1. *When n is odd, the inequality*

$$\sum_{j=1}^{n} x_j < \frac{n+1}{2}$$

is valid for the LP relaxation of the "specific example".

Proof. This is obvious from the second part of Lemma 2.1.1. ■

Corollary 2.1.2. *When n is odd, the optimal solution to an instance in the "specific example" class is:*

$$x^* = (\overbrace{0, \ldots, 0}^{\frac{n+1}{2}}, \overbrace{1, \ldots, 1}^{\frac{n-1}{2}}).$$

Proof. By Corollary 2.1.1 and integrality, an optimal solution can have at most $\frac{n+1}{2} - 1 = \frac{n-1}{2}$ non-zero variables. By Lemma 2.1.1, packing the last $\frac{n-1}{2}$ items into the knapsack is feasible, and since these have the largest coefficients, will give the best possible value. ■

Proposition 2.1.1. *If n is even, the optimal solution to an instance in the "specific example" class is:*

$$x^* = (\underbrace{0, \ldots, 0}_{\frac{n}{2}-1}, \underbrace{1, \ldots, 1}_{\frac{n}{2}}, 0).$$

Proof. When n is even,

$$b = \left\lfloor \sum_{i=1}^{n} a_i/2 \right\rfloor = \left\lfloor \sum_{j=1}^{n} \left(2^{k+n} + 2^{k+j-1} + 1/2\right) \right\rfloor = (n+1)2^{k+n} - 2^k + \frac{n}{2}.$$

Summing the smallest $\frac{n}{2}$ elements, we have

$$\sum_{j=1}^{\frac{n}{2}} a_j = \sum_{j=1}^{\frac{n}{2}} \left(2^{k+n+1} + 2^{k+j} + 1\right) = n2^{k+n} + 2^{k+\frac{n}{2}+1} - 2^{k+1} + \frac{n}{2} < b. \tag{2.4}$$

Summing the first $\frac{n}{2} + 1$ elements, we have

$$\sum_{j=1}^{\frac{n}{2}+1} a_j = \sum_{j=1}^{\frac{n}{2}+1} \left(2^{k+n+1} + 2^{k+j} + 1\right) = (n+2)2^{k+n} + 2^{k+\frac{n}{2}+2} - 2^{k+1} + \frac{n}{2} + 1 > b. \tag{2.5}$$

Thus, we can pack at most $\frac{n}{2}$ items. The most valuable $\frac{n}{2}$ items have total weight

$$\sum_{j=\frac{n}{2}+1}^{n} a_j = \sum_{j=\frac{n}{2}+1}^{n} \left(2^{k+n+1} + 2^{k+j} + 1\right) = (n+1)2^{k+n} + 2^{k+n} - 2^{k+\frac{n}{2}+1} + \frac{n}{2} > b,$$

and, thus, we cannot simply pack the last $\frac{n}{2}$ items. However, note that

$$\begin{aligned} \left(\sum_{j=1}^{\frac{n}{2}-1} a_j\right) + a_n &= \left(\sum_{j=1}^{\frac{n}{2}-1} \left(2^{k+n+1} + 2^{k+j} + 1\right)\right) + 2^{k+n+1} + 2^{k+n} + 1 \\ &= (n+1)2^{k+n} + 2^{k+\frac{n}{2}} - 2^{k+1} + \frac{n}{2} > b. \end{aligned} \tag{2.6}$$

This shows that if the n-th item is included, we can pack at most $\frac{n}{2}-1$ items with total value no more than

$$\alpha := \sum_{j=\frac{n}{2}+2}^{n} a_j = \sum_{j=\frac{n}{2}+2}^{n} \left(2^{k+n+1}+2^{k+j}+1\right) = n2^{k+n} - 2^{k+\frac{n}{2}+1} + \frac{n}{2} - 1. \tag{2.7}$$

Excluding the n-th item and packing the most valuable $\frac{n}{2}$ items among the remaining $n-1$ items has value

$$\alpha < \sum_{j=\frac{n}{2}}^{n-1} a_j = \sum_{j=\frac{n}{2}}^{n-1} \left(2^{k+n+1}+2^{k+j}+1\right) = (n+1)2^{k+n} - 2^{k+\frac{n}{2}} + \frac{n}{2} < b. \tag{2.8}$$

Thus, we obtain that the optimal solution is $x^* = (\underbrace{0,\ldots,0}_{\frac{n}{2}-1},\underbrace{1,\ldots,1}_{\frac{n}{2}},0)$. ■

Next, we define the multi-variable branching scheme to be used for this class of instances. Let $\hat{x}$ be the solution to LP relaxation at a node, let $X_I = \{i:\ \hat{x}_i \in \{0,1\}, i \in [n]\}$ be the set of variables with an integral value, i.e. 0 or 1, in that solution, $X_f = \{i:\ 0 < \hat{x}_i < 1, i \in [n]\}$ be the set of variables with a fractional value in that solution, and $j = \min\{i:\ i \in X_f\}$ be the index of the first variable with a fractional value. Then we branch on the set $S = X_I \cup \{j\}$.

Theorem 2.1.1. *A branch and bound algorithm employing a multi-variable branching based on the set S as defined above solves an instance in at most 7 nodes when n is even and at most 3 nodes when n is odd.*

Proof. We introduce the following notation. The root node of the search tree is identified by $(0,\cdot)$ and other nodes are identified either by a pair $(k,+)$ or by a pair $(k,-)$ where k represents the depth of the node, $-$ indicates the node was reached from its parent by branching down, and $+$ indicates the node was reached from its parent by branching up. In general, this does not uniquely identify a node in the search tree, but, as we will see shortly, in this case it does. The solution $x^{(0,\cdot)}$ at the root node has at most one fractional variable,

which implies that $S = [n]$.

When n is odd, by Lemma 2.1.1, we have $\frac{n-1}{2} < \sum_{i=1}^{n} x_i^{(0,\cdot)} < \frac{n+1}{2}$. Branching on $S = [n]$ generates two branches defined by $\sum_{i=1}^{n} x_i \leq \frac{n-1}{2}$ and $\sum_{i=1}^{n} x_i \geq \frac{n+1}{2}$. By Lemma 2.1.1, LP$^{(1,+)}$ is infeasible, while LP$^{(1,-)}$ has solution $x^{(1,-)} = (\underbrace{0, \ldots, 0}_{\frac{n+1}{2}}, \underbrace{1, \ldots, 1}_{\frac{n-1}{2}})$, which is the optimal solution to the instance. Thus, in this case, at most 3 nodes are evaluated.

When n is even, by (2.5), $\sum_{i=1}^{n} x_i^{(0,\cdot)} < \frac{n}{2} + 1$. By reasoning in the proof of Proposition 2.1.1, $\frac{n}{2} - 1 < \sum_{i=1}^{n} x_i^{(0,\cdot)}$. To see this, suppose otherwise, i.e., suppose $\frac{n}{2} - 1 \geq \sum_{i=1}^{n} x_i^{(0,\cdot)}$. In this case, the objective value of $x^{(0,\cdot)}$ cannot be better than that obtained by packing the $\frac{n}{2} - 1$ most valuable items, given by α in (2.7). But $\alpha < z^*$, the value of the integer program, which cannot exceed the value of the LP relaxation (which must be b).

We consider several cases for $\sum_{i=1}^{n} x_i^{(0,\cdot)}$ in the open interval $(\frac{n}{2} - 1, \frac{n}{2} + 1)$.

Case 1: $\frac{n}{2} - 1 < \sum_{i=1}^{n} x_i^{(0,\cdot)} < \frac{n}{2}$. Branching on $S = [n]$ generates two branches defined by $\sum_{i=1}^{n} x_i \leq \frac{n}{2} - 1$ and $\sum_{i=1}^{n} x_i \geq \frac{n}{2}$. For LP$^{(1,-)}$, it is easy to see that the solution is $x^{(1,-)} = (\underbrace{0, \ldots, 0}_{\frac{n}{2}+1}, \underbrace{1, \ldots, 1}_{\frac{n}{2}-1})$, which is integer but not optimal. Then we analyze two subcases for LP$^{(1,+)}$. (The search trees are shown in Figure 2.1.)

Case 1(a): If solution $x^{(1,+)}$ has exactly one fractional component, then $S = [n]$ and $\frac{n}{2} < \sum_{i=1}^{n} x_i^{(1,+)} < \frac{n}{2} + 1$. Branching on $S = [n]$ generates two branches defined by $\sum_{i=1}^{n} x_i \leq \frac{n}{2}$ and $\sum_{i=1}^{n} x_i \geq \frac{n}{2} + 1$. Clearly, LP$^{(2,+)}$ is infeasible. Observe that LP$^{(2,-)}$ has both $\sum_{i=1}^{n} x_i \leq \frac{n}{2}$ and $\sum_{i=1}^{n} x_i \geq \frac{n}{2}$, which implies that $\sum_{i=1}^{n} x_i = \frac{n}{2}$, which forces $x^{(2,-)}$ to have exactly two fractional variables. One of them has to be x_n, because if $x_n^{(2,-)} = 0$, it will not give the largest possible value by (2.7) and (2.8). If $x_n^{(2,-)} = 1$, even if all $\frac{n}{2} - 1$ items with smallest weight are included, i.e., a_1 through $a_{\frac{n}{2}-1}$, the weight constraint is still violated in view of (2.6). Therefore, at this node, $S = [n-1]$, which generates two branches defined by $\sum_{i=1}^{n-1} x_i \leq \frac{n}{2} - 1$ and $\sum_{i=1}^{n-1} x_i \geq \frac{n}{2}$. LP$^{(3,-)}$ has both $\sum_{i=1}^{n} x_i = \frac{n}{2}$

and $\sum_{i=1}^{n-1} x_i \leq \frac{n}{2} - 1$, which implies $x_n = 1$ and $\sum_{i=1}^{n-1} x_i = \frac{n}{2} - 1$, which is infeasible. LP$^{(3,+)}$ has solution $x^{(3,+)} = (\underbrace{0, \ldots, 0}_{\frac{n}{2}-1}, \underbrace{1, \ldots, 1}_{\frac{n}{2}}, 0)$ which is optimal to the original integer program. Thus, in this case, at most 7 nodes will be evaluated.

Case 1(b): If solution $x^{(1,+)}$ has two fractional components, then $x_n^{(1,+)}$ must be fractional using a similar argument to the one given for LP$^{(2,-)}$ in Case 1(a). Therefore, at this node, $S = [n-1]$, which generates two branches defined by $\sum_{i=1}^{n-1} x_i \leq \frac{n}{2} - 1$ and $\sum_{i=1}^{n-1} x_i \geq \frac{n}{2}$. LP$^{(2,-)}$ has both $\sum_{i=1}^{n} x_i \geq \frac{n}{2}$ and $\sum_{i=1}^{n-1} x_i \leq \frac{n}{2} - 1$ which implies $x_n = 1$ and $\sum_{i=1}^{n-1} x_i = \frac{n}{2} - 1$, which is infeasible by (2.6). LP$^{(2,+)}$ has both $\sum_{i=1}^{n} x_i \geq \frac{n}{2}$ and $\sum_{i=1}^{n-1} x_i \geq \frac{n}{2}$, one of which is redundant. Therefore, its solution $x^{(2,+)}$ has exactly one fractional variable, and, consequently, $S = [n]$ and $\frac{n}{2} < \sum_{i=1}^{n} x_i^{(2,+)} < \frac{n}{2} + 1$. Branching on $S = [n]$ generates two branches defined by $\sum_{i=1}^{n} x_i \leq \frac{n}{2}$ and $\sum_{i=1}^{n} x_i \geq \frac{n}{2} + 1$. Clearly, LP$^{(3,+)}$ is infeasible. LP$^{(3,-)}$ has $\sum_{i=1}^{n} x_i \geq \frac{n}{2}$, $\sum_{i=1}^{n-1} x_i \geq \frac{n}{2}$ and $\sum_{i=1}^{n} x_i \leq \frac{n}{2}$, which implies $\sum_{i=1}^{n} x_i = \frac{n}{2}$ and $x_n = 0$, which gives solution $x^{(3,-)} = (\underbrace{0, \ldots, 0}_{\frac{n}{2}-1}, \underbrace{1, \ldots, 1}_{\frac{n}{2}}, 0)$, which is the optimal to the instance. Thus, in this case too, at most 7 nodes will be evaluated.

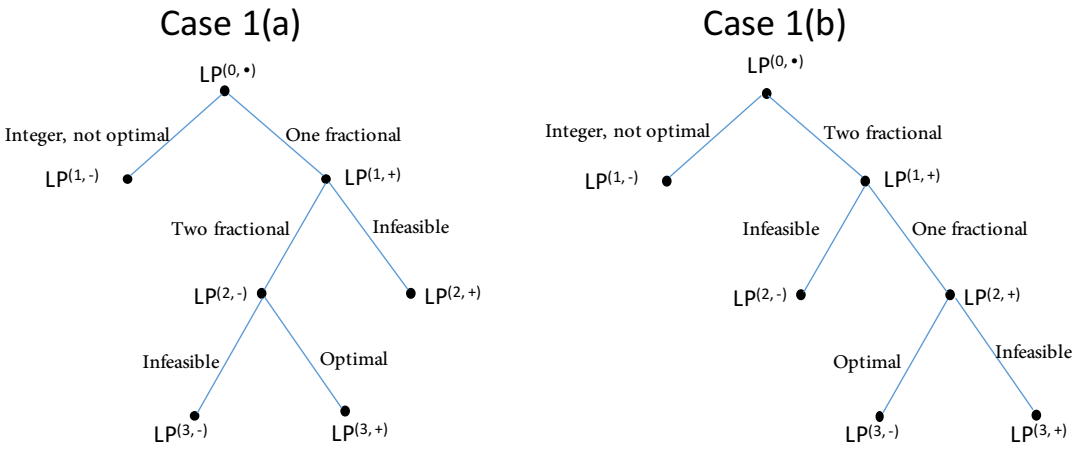


Figure 2.1: Search trees when n is even for Case 1.

Case 2: $\frac{n}{2} < \sum_{i=1}^{n} x_i^{(0,\cdot)} < \frac{n}{2} + 1$**.** Branching on $S = [n]$ generates two branches defined by $\sum_{i=1}^{n} x_i \leq \frac{n}{2}$ and $\sum_{i=1}^{n} x_i \geq \frac{n}{2} + 1$. LP$^{(1,+)}$ is infeasible. Next, we analyze two

subcases for $\text{LP}^{(1,-)}$. (The search trees are shown in Figure 2.2.)

Case 2(a): If $x^{(1,-)}$ has only one fractional variable, then $S = [n]$ and $\frac{n}{2} - 1 < \sum_{i=1}^{n} x_i^{(1,-)} < \frac{n}{2}$ and two branches are generated defined by $\sum_{i=1}^{n} x_i \leq \frac{n}{2} - 1$ and $\sum_{i=1}^{n} x_i \geq \frac{n}{2}$, respectively. $\text{LP}^{(2,-)}$ and $\text{LP}^{(2,+)}$ are the same as $\text{LP}^{(1,-)}$ in case Case 1 and $\text{LP}^{(2,-)}$ in case Case 1(a), which completes the argument for this case.

Case 2(b): If $x^{(1,-)}$ has two fractional variables, by similar argument for $\text{LP}^{(2,-)}$ in Case 1(a), we know $x_n^{(1,-)}$ is fractional. Then $S = [n-1]$ and $\frac{n}{2} - 1 < \sum_{i=1}^{n-1} x_i^{(1,-)} < \frac{n}{2}$ and two branches are generated defined by $\sum_{i=1}^{n-1} x_i \leq \frac{n}{2} - 1$ and $\sum_{i=1}^{n-1} x_i \geq \frac{n}{2}$, respectively. $\text{LP}^{(2,+)}$ has both $\sum_{i=1}^{n} x_i \leq \frac{n}{2}$ and $\sum_{i=1}^{n-1} x_i \geq \frac{n}{2}$, which yields the optimal solution $x^{(2,+)} = (\underbrace{0, \ldots, 0}_{\frac{n}{2}-1}, \underbrace{1, \ldots, 1}_{\frac{n}{2}}, 0)$. $\text{LP}^{(2,-)}$ has both $\sum_{i=1}^{n} x_i \leq \frac{n}{2}$ and $\sum_{i=1}^{n-1} x_i \leq \frac{n}{2} - 1$, one of which is redundant. $x^{(2,-)}$ can only have one fractional variable, and $\sum_{i=1}^{n-1} x_i < \frac{n}{2} - 1$ and $x_n^{(2,-)}$ is fractional. Thus $S = [n]$ and $\frac{n}{2} - 1 < \sum_{i=1}^{n} x_i^{(2,-)} < \frac{n}{2}$ and two branches are generated defined by $\sum_{i=1}^{n} x_i \leq \frac{n}{2} - 1$ and $\sum_{i=1}^{n} x_i \geq \frac{n}{2}$, respectively. $\text{LP}^{(3,-)}$ are the same as $\text{LP}^{(1,-)}$ in Case 1, which has non-optimal integer solution, and $\text{LP}^{(3,+)}$ is infeasible. ■

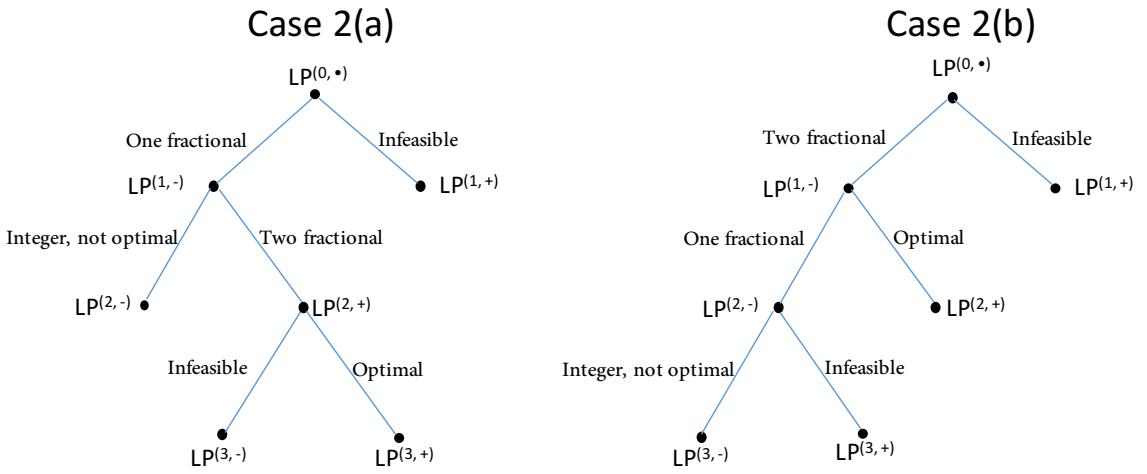


Figure 2.2: Search trees when n is even for Case 2.

2.2 Examples and Analysis

The multi-variable branching scheme used to solve Chvátal's specific example class of instances shows the dramatic improvements that are possible, but it is a specific branching scheme for a specific class of instances. We now present and analyze examples in which branching on a set of variables of size two improves over branching on a set of variables of size one, and branching on a set of variables of size three improves over branching on a set of variables of size two. The insights obtained from the analysis suggest branching schemes that are more general and that may be computationally viable.

The following example illustrates that branching on sets of variables of size two can be better than branching on sets of variables of size one (standard single-variable branching). Consider the 0-1 knapsack problem

$$\begin{aligned} z^* = \ \max \quad & 9x_1 + 4.2x_2 + x_3 \\ \text{subject to} \quad & 3x_1 + 2x_2 + x_3 \leq 5.7, \\ & x_i \in \{0,1\}, \quad i \in [3]. \end{aligned}$$

The solution to the LP relaxation is $x^{LP} = (1, 1, 0.7)$ with value $z^{LP} = 13.9$. Using standard single-variable branching, the solution to LP relaxation at the node on the down branch is $x^- = (1, 1, 0)$ with value $z^- = 13.2$, and the solution to LP relaxation at the node on the up branch is $x^+ = (1, 0.85, 1)$ with value $z^+ = 13.57$. If, instead, we branch on the set of variables $S = \{2, 3\}$, then the solution to LP relaxation at the node on the down branch is $x^- = (1, 1, 0)$ with value $z^- = 13.2$, and the solution to LP relaxation at the node on the up branch is $x^+ = (0.9, 1, 1)$ with value $z^+ = 13.3$. Thus, we obtain a better bound as $\max\{13, 13.3\} = 13.3 < \max\{13, 13.57\} = 13.57$. See Figure 2.3 for an illustration.

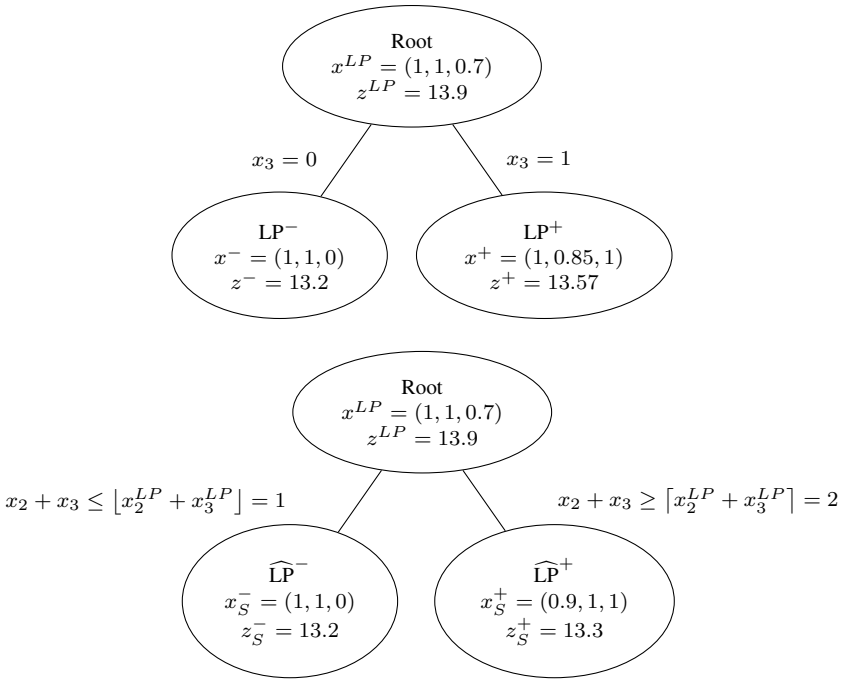


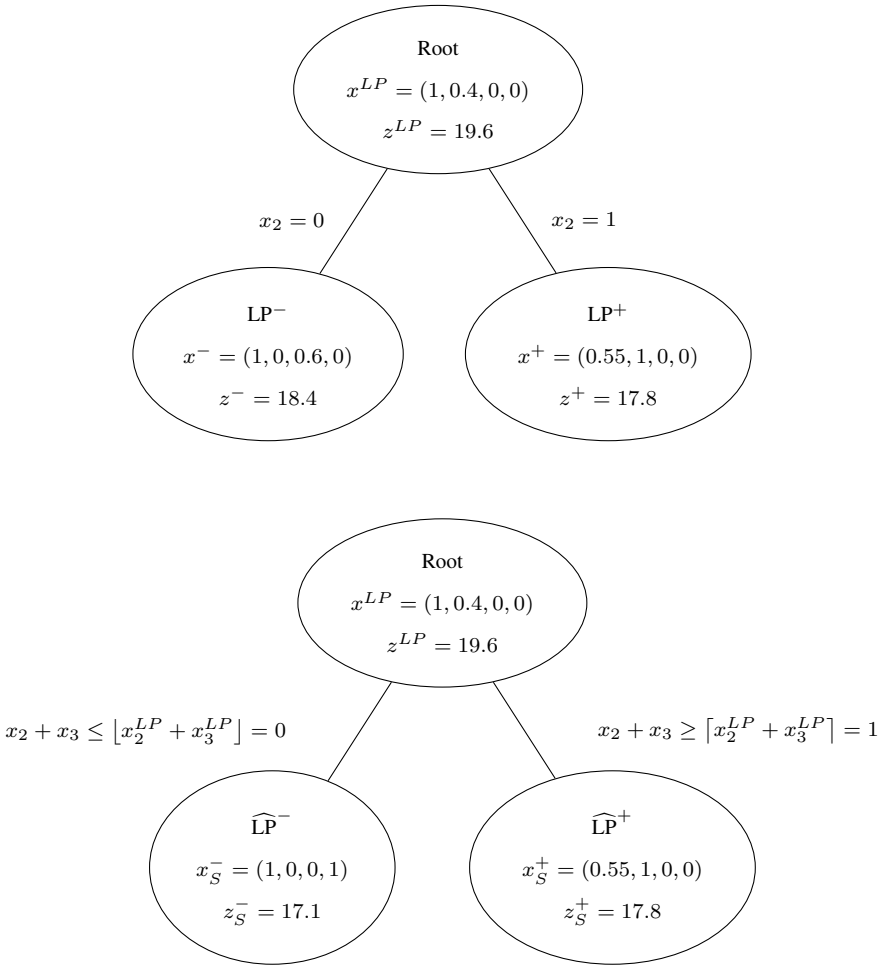
Figure 2.3: Example in which multi-variable branching (with set of branching variables $|S| = 2$) improves over single-variable branching.

Consider the 0-1 knapsack problem

$$\begin{aligned} z^* = \ \text{max} \quad & 16x_1 + 9x_2 + 4x_3 + 1.1x_4 \\ \text{subject to} \quad & 4x_1 + 3x_2 + 2x_3 + x_4 \leq 5.2, \\ & x_i \in \{0, 1\}, \quad i \in [4]. \end{aligned}$$

The solution to the LP relaxation is $x^{LP} = (1, 0.4, 0, 0)$ with value $z^{LP} = 19.6$. Using standard single-variable branching, the solution to the LP relaxation at the node on the down branch is $x^- = (1, 0, 0.6, 0)$ with value $z^- = 18.4$, and the solution to the LP relaxation at the node on the up branch is $x^+ = (0.55, 1, 0, 0)$ with value $z^+ = 17.8$. If, instead, we branch on the set of variables $S = \{2, 3\}$, then the solution to the LP relaxation at the node on the down branch is $x^- = (1, 0, 0, 1)$ with value $z^- = 17.1$, and the solution to LP relaxation at the node on the up branch is $x^+ = (0.55, 1, 0, 0)$ with value $z^+ = 17.8$. Thus, we obtain a better bound: $\max\{17.1, 17.8\} = 17.8 < \max\{18.4, 17.8\} = 18.4$. (Branching on $S = \{1, 2\}$ gives the same bound as single-variable branching: the up branch

is infeasible and the down branch gives bound 18.4.) However, if, instead, we branch on the set of variables $S = \{1, 2, 3\}$, then the solution to the LP relaxation at the node on the down branch is $x^- = (1, 0, 0, 1)$ with value $z^- = 17.1$, but the solution to the LP relaxation at the node on the up branch is $x^+ = (0.2, 0.8, 1, 0)$ with value $z^+ = 14.4$. Thus, we obtain an even better bound as $\max\{17.1, 14.4\} = 17.1 < \max\{17.1, 17.8\} = 17.8$. See Figure 2.4 for an illustration.



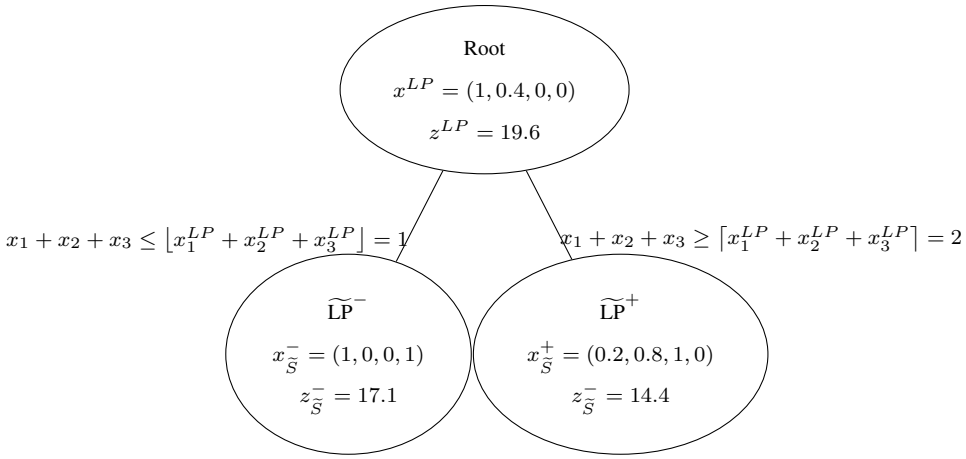


Figure 2.4: Example in which multi-variable branching using a set of branching variables of size three improves over multi-variable branching using a set of branching variables of size two.

Next, we present a necessary, but not sufficient, condition as well as a sufficient, but not necessary, condition for multi-variable branching using a set of variables of size two to outperform single-variable branching. In all that follows, when we say "outperforms" or "is better than" we mean that the dual bound after branching is lower. Without loss of generality, suppose $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \ldots \geq \frac{p_n}{w_n}$ and that the solution to the LP relaxation at the root node is fractional, i.e., the solution has the form

$$x^{LP} = (\underbrace{1, \ldots, 1}_{i-1}, f, \underbrace{0, \ldots, 0}_{n-i}),$$

with $1 < i \leq n$. (Note that we have assumed that $w_i \leq b$ for $i = 1, \ldots, n$.) We consider two cases, $i = n$ and $1 < i < n$. In the former case, we give a necessary condition for branching on a set of two variables to outperform single-variable branching. In the latter case, we provide a sufficient condition.

Proposition 2.2.1. *If $i = n$, then branching on $S = \{k, n\}$ can be better than branching on $S' = \{n\}$ only if the solution to the LP relaxation on the up branch, x^+, is fractional and k is the index of the fractional variable in x^+.*

Proof. Since $i = n$, it must be that $x^{LP} = (1, 1, \ldots, 1, f)$ with $0 < f < 1$. It is easy to see

that branching on S' gives

$$x^- = (\underbrace{1, \ldots, 1}_{n-1}, 0), \quad x^+ = (\underbrace{1, \ldots, 1}_{k-1}, f^+, \underbrace{0, \ldots, 0}_{n-k-1}, 1), \quad 0 \leq f^+ \leq 1,\ 1 \leq k \leq n-1.$$

If f^+ is integer, then we have integer solutions on both the down and the up branch, and $z = \max\{z^-,\ z^+\}$ is the optimal value for the original integer program. Therefore, branching on S cannot be better than branching on S'.

So assume $0 < f^+ < 1$. Consider $S = \{\ell, n\}$ with $\ell < k$ ($k \geq 2$). On the down branch we add $x_\ell + x_n \leq \left\lfloor x_\ell^{LP} + x_n^{LP} \right\rfloor = 1$, and on the up branch we add $x_\ell + x_n \geq \left\lceil x_\ell^{LP} + x_n^{LP} \right\rceil = 2$. Note that x^- is feasible for the down branch, and, thus, $z_S^- \geq z^-$. Similarly, x^+ is feasible for the up branch, and, thus, $z_S^+ \geq z^+$. This implies $z_S = \max\{z_S^-, z_S^+\} \geq \max\{z^-, z^+\} = z$, which means that branching on S cannot be better than branching on S'. Next, consider $S = \{\ell, n\}$ with $k < \ell < n$ ($k \leq n-2$). On the down branch, we add $x_\ell + x_n \leq \left\lfloor x_\ell^{LP} + x_n^{LP} \right\rfloor = 1$, which is satisfied by both x^- and x^+. Again, branching on S cannot be better than branching on S'. ■

Proposition 2.2.2. *Suppose $1 < i < n$ and let $\frac{p_{i+1}}{w_{i+1}} > \frac{p_{i+2}}{w_{i+2}}$ and $w_{i+1} \geq w_i$. Furthermore, let the value of the solution to the LP relaxation on the down branch be greater than the value of the solution to the LP relaxation on the up branch, i.e., $z^- > z^+$, when branching on $S' = \{i\}$. Then branching on $S = \{i, i+1\}$ is better than branching on S' and achieves the best bound that can be achieved by branching on a set of variables of size two.*

Proof. Since $w_{i+1} \geq w_i$, branching on S' results, on the down branch, in

$$x^- = (\underbrace{1, \ldots, 1}_{i-1}, 0, f^-, \underbrace{0, \ldots, 0}_{n-i-2}), \quad 0 < f^- < 1.$$

When we branch on S, we add $x_i + x_{i+1} \leq \left\lfloor x_i^{LP} + x_{i+1}^{LP} \right\rfloor = 0$ on the down branch, and, because $\frac{p_{i+1}}{w_{i+1}} > \frac{p_{i+2}}{w_{i+2}}$, we have that $z_S^- < z^-$.

Let $\bar{b} = b - \sum_{k=1}^{i-1} w_k$. Because $x_i^{LP} = f$ is fractional, $w_i > \bar{b}$, thus, $w_{i+1} > \bar{b}$. We

will show that $x_S^+ = x^+$, and, thus, $z_S^+ = z^+$. When we branch on S, we add $x_i + x_{i+1} \geq \lceil x_i^{LP} + x_{i+1}^{LP} \rceil = 1$, which causes some "waste" of the resource as items i and $(i+1)$ are less desirable than items 1 through $i-1$. By setting $x_i = 1$ and $x_{i+1} = 0$, the branching constraint is satisfied and the waste is minimized. As a consequence, we obtain solution x^+. Thus, $z_S = \max\{z_S^-, z_S^+\} = \max\{z_S^-, z^+\} < z^- = z$.

Then, consider branching on $S'' = \{k, i\} \neq \{i, i+1\}$. When $k < i$, we add $x_k + x_i \leq \lfloor x_k^{LP} + x_i^{LP} \rfloor = 1$ on the down branch, and have $z_{S''}^- \geq z^-$, because x^- satisfies $x_k^- + x_i^- \leq 1$. This implies $z_{S''}^- \geq z^- > z^+$, i.e., branching on S'' cannot be better than branching on S (in fact, it cannot even better than branching on S'). When $k > i+1$, we add $x_i + x_k \leq \lfloor x_i^{LP} + x_k^{LP} \rfloor = 0$ on the down branch, and have $x_i = x_k = 0$. Since $w_{i+1} \geq w_i > \bar{b}$, all of the resource can be consumed by items in $\{1, \ldots, i-1, i+1\}$. When branching on S, on the down branch only items in $\{1, \ldots, i-1, i+2, \ldots, n\}$ can be used, which implies $z_{S''}^- \geq z_S^-$. We have already argued that when we branch on S, on the up branch we will have $x_i^+ = 1$ and $x_{i+1}^+ = 0$. This solution is feasible when branching on S'', as $x_i + x_k \geq \lfloor x_i^{LP} + x_k^{LP} \rfloor = 1$ is added on the up branch. Therefore, we also have $z_{S''}^+ \geq z_S^+$, which implies branching on S'' cannot be better than branching on S. ■

2.3 Computational Study

In this section, we evaluate the performance of four multi-variable branching schemes motivated by the observations in the previous section.

2.3.1 Branching on sets of size two

The first two multi-variable branching schemes involve sets of size two in order to limit the computation time required to identify the set of variables to branch on. In the first scheme, we use strong branching to computer SB scores for all sets of size no larger than two (all sets of size one and two) and then pick the best one to branch on. This scheme, which we refer to as "B1", is an analogue of the standard single variable strong branching

and can be very time-consuming. In the second scheme, which is inspired by the analysis in Section 2.2, we choose $S = \{i, i+1\}$ where i is the largest index such that x_i^* is fractional and $i < n$, and choose $S = \{i, j\}$, when $i = n$ and where j the largest index such that $x_j^* \in \{0, 1\}$. Observe that this multi-variable branching scheme is attractive computationally as determining S does not require the solution of any linear programs. We refer to this scheme as "B2".

2.3.2 Branching on dynamically determined sets

The last two multi-variable branching schemes may involve sets of size larger than two as this may improve the performance (as we have seen in Section 2.2). To control the computation time somewhat, we generate the set S of variables to branch on dynamically. More specifically, we start from $S = \{i\}$ with i the index of the fractional variable that gives the best bound among all fractional variables when branching on these variables. Note that i is the index of the variable that would be chosen if strong branching was used. Next, we evaluate sets $S = \{i, j\}$ for j such that $x_i^{LP} + x_j^{LP}$ is fractional, and, again, choose the set that gives the best bound, say $S = \{i, \hat{j}\}$. If the best bound associated with $\{i, \hat{j}\}$ is better than the best bound associated with $\{i\}$, then the process continues, i.e., we explores all sets of size three, extending $\{i, \hat{j}\}$, to see if there is one among them that results in a better best bound. We continue extending the set S with one more variable as long as it results in an improved bound. We refer to this scheme as "B3". Clearly, the B3 branching scheme is computationally intensive, and, therefore, we consider a variant in which the cardinality of the set is limited to at most K_1 and it is only applied at the top of the search tree, i.e., at nodes of depth less than or equal to K_2. At nodes of depth more than K_2, standard single-variable branching is used, where we branch on the fractional variable with the smallest index. We refer to this scheme as "B4".

2.3.3 Computational experiments

In our computational experiments, we compare the five branching schemes:

B0: Standard single-variable branching;

B1: Standard two-variable branching;

B2: Restricted two-variable branching;

B3: Dynamic multi-variable branching; and

B4: Restricted multi-variable branching.

We implemented these five variable selection schemes in the branch-and-bound framework of CPLEX using their callback functions. All experiments to compare the performance of the five branching schemes use CPLEX 12.8 in single-thread mode with a time limit of one hour for each instance. Since our focus is on the impact of the branching strategy, we turn off cuts, heuristics, and presolve.

For our computational experiments, we use two classes of instances introduced in [64]: uncorrelated data instances and weakly correlated data instances with coefficient range $R = \{10^3, 10^4\}$ and instance size $n = \{100, 200, 500, 1000\}$, i.e., 8 different combinations of R and n for each class. For each combination, we generate 100 instances. The performance metrics of interest are: (1) N: the number of nodes evaluated, (2) t: the solve time.

Many commercial (and open-source) solvers have switched to using the product scoring rule [2], rather than $\max\{z^-, z^+\}$, to assess the desirability of branching on a particular variable. However, our theoretical analysis in the previous sections has been done using the $\max$ scoring rule. Thus, we show all of our results computed by both the $\max$ and product rules. Note a score function is needed only for B1, B3, and B4. Actually, preliminary experiments with these instances revealed no performance improvement (in fact, in many cases a performance deterioration) is yielded by the product form.

In Tables 2.1 and 2.2, we report, for each of the branching schemes, the number of nodes evaluated and the solve time as a fraction of the number of nodes evaluated and the solve time of standard single-variable branching B0, respectively, averaged over all instances. Since nearly all weakly corrected instances with $n = 1000$ exceed the 1 hour time limit when B1 (both max and product functions) or B3 (product function) is used, we leave out those results.

Table 2.1: Ratios of N with respect to B0.

R=10^3	n \ Score	Max$\{z^-, z^+\}$					Product		
		B0	B1	B2	B3	B4	B1	B3	B4
Uncorrelated	100	1.00	0.55	0.80	0.23	0.56	0.64	0.44	0.87
	200	1.00	0.46	0.69	0.16	0.58	0.56	0.38	0.85
	500	1.00	0.36	0.52	0.08	0.67	0.46	0.28	0.96
	1000	1.00	0.34	0.45	0.04	1.04	0.42	0.22	4.67
Weakly correlated	100	1.00	0.75	1.31	0.42	0.69	0.92	1.02	1.34
	200	1.00	0.62	1.08	0.26	0.73	0.74	0.93	1.30
	500	1.00	0.45	0.75	0.17	1.26	0.59	0.81	1.70
	1000	1.00	—	0.63	0.19	4.13	—	—	3.93

R=10^4	n \ Score	Max$\{z^-, z^+\}$					Product		
		B0	B1	B2	B3	B4	B1	B3	B4
Uncorrelated	100	1.00	0.52	0.74	0.20	0.52	0.63	0.43	0.81
	200	1.00	0.47	0.71	0.16	0.53	0.55	0.40	0.86
	500	1.00	0.36	0.50	0.08	0.51	0.45	0.30	0.79
	1000	1.00	0.28	0.32	0.02	0.53	0.33	0.15	0.92
Weakly correlated	100	1.00	0.52	0.74	0.20	0.52	0.63	0.43	0.81
	200	1.00	0.47	0.71	0.16	0.53	0.55	0.40	0.86
	500	1.00	0.36	0.50	0.08	0.51	0.45	0.30	0.79
	1000	1.00	—	0.36	0.04	0.93	—	—	2.19

In Figures 2.5 and 2.6, we show two performance profiles [35] for the five branching schemes using the max score function: the first with respect to N, the number of nodes evaluated, and the second with respect to t, the solve time.

In Figure 2.5 (and also in Table 2.1), we see that all four multi-variable branching schemes achieve better node-efficiency than standard single-variable branching for both classes of instances and for all combinations of R and n. Furthermore, it is also clear

Table 2.2: Ratios of t with respect to B0.

R=10^3	n \ Score	Max$\{z^-, z^+\}$					Product		
		B0	B1	B2	B3	B4	B1	B3	B4
Uncorrelated	100	1.00	121.80	0.84	212.88	96.77	186.21	664.66	161.35
	200	1.00	391.10	0.72	469.27	153.50	630.03	1635.47	252.10
	500	1.00	2004.81	0.50	1332.98	222.53	3250.63	7639.95	365.70
	1000	1.00	8355.19	0.45	3378.24	221.88	12653.67	35272.41	314.00
Weakly correlated	100	1.00	229.33	1.63	542.09	214.71	359.89	2506.82	329.76
	200	1.00	507.03	1.27	970.21	326.34	801.70	6930.39	436.65
	500	1.00	2389.17	0.69	4011.84	303.46	4259.84	32596.95	368.25
	1000	1.00	—	0.60	17242.40	332.22	—	—	305.71

R=10^4	n \ Score	Max$\{z^-, z^+\}$					Product		
		B0	B1	B2	B3	B4	B1	B3	B4
Uncorrelated	100	1.00	124.83	0.82	224.26	97.21	197.89	678.29	156.23
	200	1.00	379.42	0.73	449.02	139.80	598.36	1655.14	251.13
	500	1.00	1946.13	0.48	1271.18	182.79	3115.66	5386.72	310.01
	1000	1.00	7114.82	0.32	3093.03	124.79	10592.71	13749.53	199.88
Weakly correlated	100	1.00	124.83	0.82	224.26	97.21	197.89	678.29	156.23
	200	1.00	379.42	0.73	449.02	139.80	598.36	1655.14	251.13
	500	1.00	1946.13	0.48	1271.18	182.79	3115.66	5386.72	310.01
	1000	1.00	—	0.35	6594.44	150.62	—	—	222.64

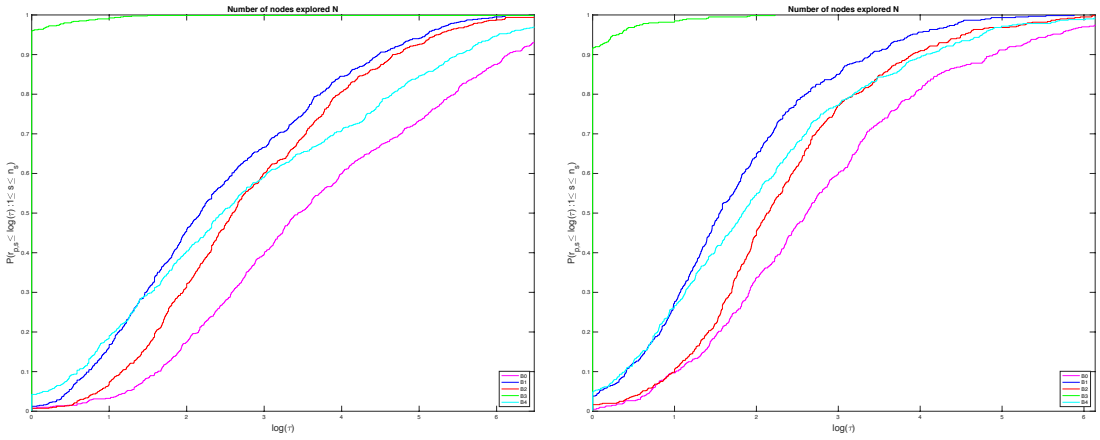


Figure 2.5: Performance profile on number of nodes explored N. Left: uncorrelated instances; Right: weakly correlated instances.

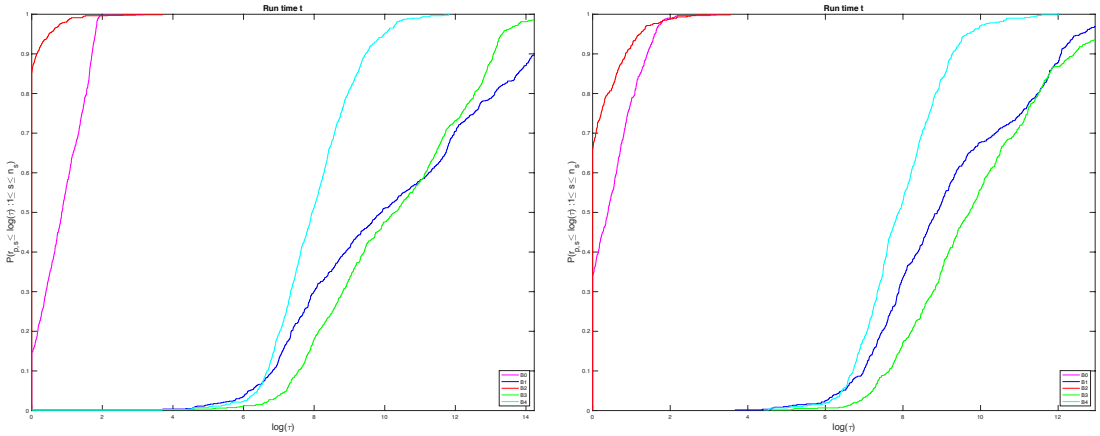


Figure 2.6: Performance profile on run time t Left: uncorrelated instances; Right: weakly correlated instances.

that branching scheme B3 results in the most significant reduction in number of nodes evaluated. However, as Figure 2.6 reveals (and also Table 2.2), this reduction in number of nodes evaluated comes at a high price in terms of solve time. As expected, branching scheme B4 is faster than branching scheme B3 even though it explores many more nodes. Tuning the two restriction parameters of B4 may achieve an even better balance between node-efficiency and time-efficiency. We observe too that only one of the multi-variable branching schemes, B2, does better than the standard single-variable scheme, B0, in terms of solve time; for uncorrelated instances and for large weakly corrected instances (with $n = 500$ and 1000). By comparing both score functions for B1, B3, and B4, we arrive to a different conclusion from what's claimed in [2]. This is interesting and suggests that score function also plays an important role in multi-variable branching.

To further explore the benefits of multi-variable scheme B2, we generated and solved additional instances of significantly larger size: $n = \{5000, 10000\}$ (with $R = 10^4$). As before, for each combination of R and n, we generate 100 instances. The results can be found in Table 2.3.

These results for branching scheme B2 (as well as those in Table 2.1 and 2.2) indicate the promise of multi-variable branching: both the number of nodes and the solve time can be reduced.

Table 2.3: Ratios of N and t with respect to B0 for instances of larger size.

		N		t	
	n	B0	B2	B0	B2
Uncorrelated	5000	1.00	0.32	1.00	0.43
	10000	1.00	0.54	1.00	0.63
Weakly correlated	5000	1.00	0.37	1.00	0.47
	10000	1.00	0.60	1.00	0.69

2.4 Final Remarks

In this chapter, we explored the potential benefits of branching on sets of variables rather than standard single-variable branching. Preliminary computational experiments on randomly generated instances of the 0-1 knapsack problem show promise and reveal the trade off between node-efficiency and time-efficiency. Our current research is focused on reproducing the promise of multi-variable branching on general mixed integer programs and developing efficient heuristics to obtain high-quality sets of variables to branch on.

CHAPTER 3
LEARNING GENERALIZED STRONG BRANCHING FOR SET COVERING, SET PACKING, AND 0-1 KNAPSACK PROBLEMS

The primary challenge associated with the use of a set branching strategy is the efficient selection of an effective branching set. The promising computational results reported in the literature on developing a ML model that effectively mimics SB and is computationally efficient (as it does not require solving LPs to compute SB scores), prompted us to explore similar ideas, but focusing on multi-variable variants of strong branching, which we refer to as GSB-k, where k specifies the cardinality of the sets of variables to branch on. More specifically, we focus on learning GSB-2 for three well-known classes of integer programs (IPs): set covering, set packing, and 0-1 knapsack problems.

Practical implementations of SB only compute SB scores for a small set of candidates. Similarly, our implementation of GSB-2 selects a pair of variables to branch on from among a small set of candidate pairs, where each candidate pair includes the variable identified as the best single variable to branch on. Thus, our approach relies on two learned models: one to select the best single variable to branch on from among a small set of candidate variables, and one to select the best pair of variables to branch on from among a small set of candidate pairs of variables. We employ a systematic approach to identify the features to be used to train the ML model. We start from 25 features that we believe, based on our domain knowledge, provide meaningful information for predicting the ranking of candidates by SB scores. After training data has been gathered, a feature selection procedure is applied to eliminate insignificant features. The result is a small set of significant features (about 5) that can be computed efficiently. More details are found in the sections below.

The remainder of the chapter is organized as follows. In Section 3.1, we introduce GSB-k and our implementation of GSB-2. In Section 3.2.1 and 3.2, we present our methodology

for learning SB and extend this framework to learn GSB-2. Section 3.3 presents the results of an extensive computational study. We finish, in Section 3.4, with final remarks.

3.1 Generalized Strong Branching

GSB refers to a branching strategy in which set branching is employed and in which the set of variables to branch on is determined using the SB score. To obtain the SB score for a candidate set S at the current node in the search tree, two LPs have to be solved: one in which the branching constraint $\sum_{i \in S} x_i \geq \lceil f \rceil$ is added (the up branch) and one in which the branching constraint $\sum_{i \in S} x_i \leq \lfloor f \rfloor$ is added (the down branch). The SB score is computed using a score function $score(\Delta^-, \Delta^+)$, where $\Delta^- = |z - z^-|$, $\Delta^+ = |z - z^+|$, and z, z^+, and z^- are the objective values of the LP relaxations at the current node, the up branch, and the down branch, respectively. The candidate set S with the largest SB score is selected to branch on. If no restrictions are imposed on the candidate sets S, GSB becomes computationally intractable as the number of sets of size k is $\mathcal{O}(n^k)$. To explore the benefits of GSB, we therefore only consider GSB-2, i.e., branching on sets of size up to 2.

3.1.1 Score Function

In [2], it is shown that the score function used can significantly impact the performance of SB. His computational study shows that

$$score(\Delta^-, \Delta^+) = \max\{\Delta^-, \epsilon\} \cdot \max\{\Delta^+, \epsilon\}$$

with $\epsilon = 10^{-6}$, the product form, outperforms

$$score(\Delta^-, \Delta^+) = (1 - \mu) \min\{\Delta^-, \Delta^+\} + \mu \max\{\Delta^-, \Delta^+\},$$

the more traditional sum form, by almost 15%. The reason, most likely, is that the product form better balances the contribution of the two branches than the sum form. When one of

Δ^- and Δ^+ is much larger than the other, the SB score is dominated by the larger one in the sum form, which implies that the SB score does not fully capture the information provided by the two branches. On the other hand, with the product form, even a small value affects the SB score as it pushes down the value of the product.

Even though the product form of the score function outperforms the sum form of the score function, we believe it should be slightly modified when used for set branching. Consider, for example, set covering problems, where all constraints have the form $\sum_{k \in K} x_k \geq 1$. When branching on a single variable, the effect of fixing a variable to zero or fixing a variable to one is somewhat similar because no further variables can be fixed directly. As result, the search tree is likely somewhat balanced. When branching on a pair of variables, this is no longer the case. There is a difference between adding $x_i + x_j \leq 0$, which fixes two variables, and adding $x_i + x_j \geq 1$, which fixes no variables, and less balanced search trees may result. A simple product does not capture these differences. By placing more importance on the branch with more freedom a more effective SB score may be obtained. Therefore, we use a weighted product score function

$$\max\{\underline{z}/z - 1.0,\ \epsilon\}^{\alpha} \cdot \max\{\overline{z}/z - 1.0,\ \epsilon\}^{\beta}, \tag{3.1}$$

where $\underline{z}$ is the LP value of the branch with more freedom, $\overline{z}$ is that of the other branch, $\alpha > 1$, $\beta < 1$, $\alpha + \beta = 2$ and $\epsilon = 10^{-6}$.

The results of computational experiment using 100 randomly generated set covering instances, in which we compare the performance of the weighted product score function with $\alpha = 1.5$ and $\beta = 0.5$ to the direct product score function with $\alpha = \beta = 1$ are shown in Figure 3.1. The weighted product score function does better in 74 out of the 100 instances and on average reduces the number of nodes in the search tree by 18%.

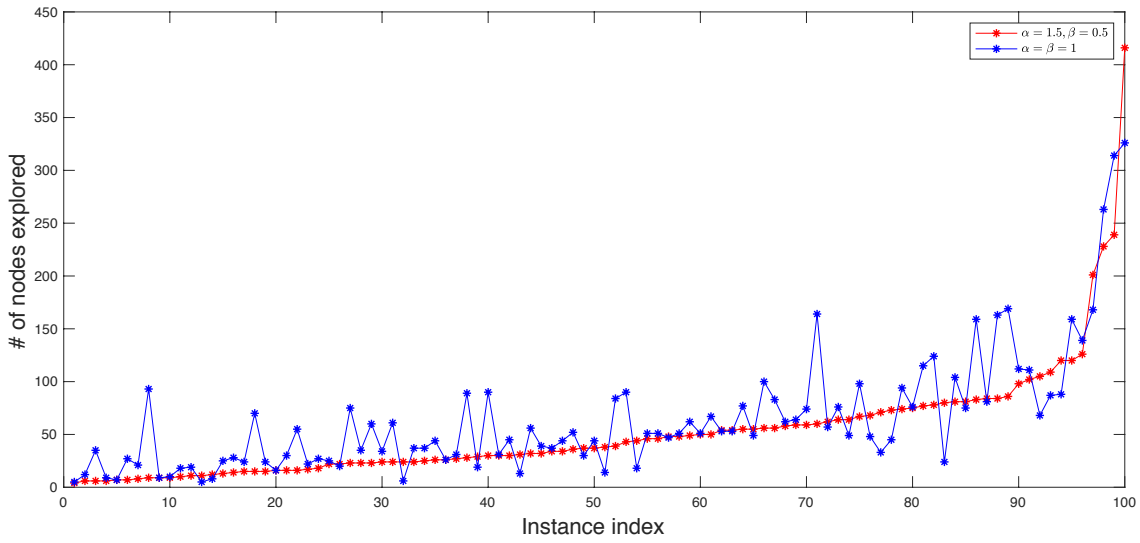


Figure 3.1: A comparison of the weighted product score function and the simple product score function on 100 randomly generated set covering instances.

3.1.2 Variable Selection

When the number of variables, n, is large, considering every possible set of size up to 2 will be computationally prohibitive. A straightforward way to reduce the number of candidates is to randomly select a small number of qualifying variable pairs, i.e., a pair (x_i, x_j) with $x_i^{LP} + x_j^{LP}$ fractional. However, given that there are $\mathcal{O}(n^2)$ potential candidates, the chance that a small subset contains a good pair of variables to branch on is small. Therefore, we adopt the following scheme. Given that the variable x_{i^*} with the highest SB score is highly likely to result in effective branching if used by itself, we include it in all the candidate pairs of variables, i.e., we consider only pairs of the form (x_{i^*}, x_j) with $x_{i^*}^{LP} + x_j^{LP}$ fractional. Preliminary computational experiments revealed that it is also beneficial to consider only variables x_j with $x_j^{LP} > 0$. Finally, let (x_{i^*}, x_{j^*}) be the variable pair with the largest SB score $\hat{s}^*$ and let the SB score of x_{i^*} be s^*. If $\hat{s}^* > s^*$, then we choose to branch on the pair (x_{i^*}, x_{j^*}), otherwise, we choose to branch on x_{i^*}. The selection scheme is summarized in Figure 3.2.

To assess the performance of the proposed selection scheme, where we randomly gen-

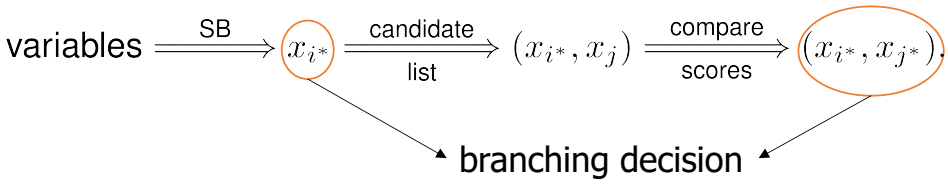


Figure 3.2: Practical implementation of GSB.

erate 50 candidate pairs, we compare its performance with three other selection schemes on 100 randomly generated set covering instances. The other selection schemes proceed as follows. Scheme 1 randomly selects 100 pairs (x_i, x_j) with $x_i^{LP} + x_j^{LP}$ fractional. Scheme 2 selects randomly selects 100 pairs (x_i, x_j) with $x_i^{LP} + x_j^{LP}$ fractional and $x_i^{LP} + x_j^{LP} > 1$. Scheme 3 randomly selects 100 pairs (x_i, x_j) with $x_i^{LP} + x_j^{LP}$ fractional and $x_i^{LP} > 0$ and $x_j^{LP} > 0$.

Table 3.1: Comparison of candidate pair selection schemes.

	GSB	Scheme 1	Scheme 2	Scheme 3
Total # of nodes explored	5286	9687	5570	6216
# of times with smallest search tree	51	6	24	19

We see that the proposed selection scheme explores fewer nodes and most often results in the smallest search tree even though it considers fewer candidate sets; 50 compared to 100 for the other schemes. This confirms that always including the variable with the highest SB score is beneficial.

3.2 Learning Generalized Strong Branching

To decide the candidate sets in GSB, we first identify the variable with the highest SB score. Therefore, learning strong branching is a prerequisite to learning generalize strong branching.

3.2.1 Learning Strong Branching

Methodology

We train a separate model that mimics SB for each problem class and the learning is performed offline. More specifically, we start by using our implementation of SB to solve instances in a training set. At the current node of the search tree, our implementation of SB selects $\ell = \min\{50, v\}$ integer variables with a fractional value in the solution to the LP relaxation, where v is the number of such variables, computes their SB scores, and then selects the one with the largest SB score to branch on. For each variable x_i in the candidate set, we collect a feature vector $\mathbf{f}_i$ and its SB score s_i. Since the ranking by SB scores suffices to select the variable to branch on, predicting the actual SB scores is not necessary. The exact ranking is not necessary either, because candidates with similar SB scores are expected to behave similarly, and, thus, can be ranked similarly. By doing so, we introduce some flexibility and tolerance for errors in the model.

We transform a list of SB scores $\{s_{i_1}, \ldots, s_{i_\ell}\}$ into an r-level ranking by a function $g : \{(1, s_{i_1}), \ldots, (\ell, s_{i_\ell})\} \to \{1, \ldots, r\}$. Without loss of generality, we suppose $s_{i_1} \geq s_{i_2} \geq \ldots \geq s_{i_\ell}$. Since we are particularly interested in the candidates with the highest SB scores, we assign rank 1 to the candidates with SB score greater than ηs_{i_1}, where $\eta \in (0, 1)$ and close to 1, which is adopted by learning to branch in [52]. Since SB score is not a perfect predictor of the benefit of branching on a variable, it is (too) risky to assign rank 1 to just one or two variables, which can happen if there is only one score that is significantly larger than others. To avoid overfitting and thus improve robustness in the training process, we we assign rank 1 to at least the top t variables. More specifically, if the number of such candidates is less than t, we assign rank 1 to the t candidates with highest SB scores. Let $k' = \text{argmax}_k\{s_{i_k} > \eta s_{i_1}\}$ and $t' = \max\{t, k'\}$. We go further than than the binary labeling scheme in [52]. The remaining candidates are ranked from 2 to r by

evenly distributed quantile:

$$g(k, s_{i_k}) = \begin{cases} 1, & 1 \leq k \leq t' \\ p+1, & t'+1 \leq k \leq \ell, \;\; s_{i_{t'}} - pq < s_{i_k} \leq s_{i_{t'}} - (p-1)q, \end{cases}$$

where $q = \frac{s_{i_{t'}} - s_{i_\ell}}{r-1} + \epsilon$, $p \in \{1, 2, \ldots, r-1\}$, $\epsilon = 10^{-15}$.

Our goal is to train a model that takes in features $(\mathbf{f}_{i_1}, \ldots, \mathbf{f}_{i_\ell})$ of the ℓ candidate variables and predicts a ranking $(r_{i_1}, \ldots, r_{i_\ell})$ close to the true ranking g. We use extreme gradient boosting (XGBoost), an optimized distributed gradient boosting library (see [25]), to accomplish this goal. In addition to its high efficacy and efficiency proved in various machine learning competitions[1], XGBoost offers a C API, which is essential for our numerical experiments since we work with CPLEX C API to take advantage of the built-in callback functions. In our training, we choose the learning objective to be *rank:pairwise*, which uses *LambdaMART* ([73]) to obtain a ranking. We also tested the other two objective functions available in XGBoost, i.e., *rank:ndcg* and *rank:map*, but they yielded worse performance in terms of the XGBoost performance metric (ndcg) and the branching scheme performance metrics (runtime and search tree size).

In the training process, we solve a set of instances (the training set TS) to obtain the data for training a ML model; this data set is split into two subsets, a set, denoted by ML-train, on which the ML algorithm is trained and a set, denoted ML-eva, on which the training quality is evaluated by ndcg, and use another set of instances (the evaluation set ES) to assess the performance of the solver when using the learned model to make branching decisions. To achieve the best performance on the evaluation set, we perform a two-stage tuning. The first stage uses the set ML-eva and the metric is ndcg, while the second uses the set ES and the metrics are the runtime and the number of nodes explored, which are what we really care about. We tune the following hyperparameters of XGBoost: *max_depth* for the maximum depth of a tree, *eta* for learning rate, *tree_method* for the tree construction

[1] https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions

algorithm used in XGBoost. After tuning has been completed, the performance of the learned model is finally evaluated on a set of instances (the testing set TEST) that have not been used in any part of the training process.

Features

In [7], they proposed three sets of features for training the model, i.e., 9 static problem features, 6 dynamic problem features and 6 dynamic optimization features (21 features in total). The static problem features represent the static state of the problem and are computed solely from parameters c, A and b once for all. The remaining two sets of features are meant to represent the the solution of the problem at the current B&B node and overall state of the optimization, respectively. The same features are used for training in [8]. Two sets of features of a candidate variable at a node for learning, i.e., 18 static features and 54 dynamic features, are used in [52]. However, little information is provided regarding the effectiveness of these features (or combinations of these features).

As computational efficiency is critical, we start from a smaller set of 25 features that may provide meaningful information to predict the ranking. Some of the features are used in the papers mentioned above, but not necessarily in the same form, such as $f_2, f_3, \ldots, f_7$ present below. Other features are introduced based on our knowledge of the domain, e.g., the observed reduced costs of a variable during the solution process and the number of times a variable appeared in the basis of an LP relaxations (f_{18} and f_{21}).

The 25 features can be grouped in three categories: static features, dynamic "history-free" features, and dynamic "history-dependent" features. Static features do not change during the solution process and can be computed once before the solution process starts. The dynamic features change during the solution process and need to be computed or updated at every node of the search tree that is not fathomed. Dynamic "history-free" features only use information at the current node, whereas dynamic "history-dependent" features embed information from previously evaluated nodes in the search tree as well. As all fea-

tures should be independent of instance size and parameter scale, we normalize feature values by the average feature value for the candidate variables.

For convenience, let z_0, z, z_j^- and z_j^+ be the objective values of the LP relaxations at the root node, at the current node, at the down branch, and at the up branch, if variable x_j is branched on, respectively. Furthermore, let c_j be the coefficient of variable x_j in the objective function, A_{ij} be the coefficient of variable x_j in the i-th constraint, and x^{LP} be the solution to the LP relaxation at the current node. Finally, let m and n be the number of constraints and variables, respectively, and $[m] = \{1, \ldots, m\}$ and $[n] = \{1, \ldots, n\}$. The 25 features associated with variable x_j are:

Static features

- f_1: Number of constraints x_j participates in divided by the total number of constraints.
- f_2: $\frac{c_j - \min_{k\in[n]} c_k}{\max_{k\in[n]} c_k - \min_{k\in[n]} c_k}$.
- f_3: $\frac{f_2}{\text{mean}(\{A_{ij},\ A_{ij}\neq 0,\ i\in[m]\})}$.
- f_4, f_5, f_6: Mean, min, max of $\frac{A_{ij}}{\sum_{k\in[n]} A_{ik}/n}$ over $i \in [m]$.

Dynamic "history-free" features

Node-based

- f_7: Depth of the current node.
- f_8: Current node gap, i.e., $|z - z_0|/|z_0|$.
- f_9: Current node infeasibility, i.e., $\sum_{k\in[n]} \min\{x_k^{LP} - \lfloor x_k^{LP} \rfloor, \lceil x_k^{LP} \rceil - x_k^{LP}\}$ divided by the number of fractional variables in the solution to the LP relaxation.

Variable-based

- f_{10}: $x_j^{LP} - \lfloor x_j^{LP} \rfloor$.

- f_{11}: $\lceil x_j^{LP} \rceil - x_j^{LP}$.
- f_{12}: $x_j^{LP} \cdot f_2$.
- f_{13}, f_{14}, f_{15}: Pseudo costs of x_j (up, down, geometric mean) divided by z.
- f_{16}: Number of times x_j appears in a constraint that is binding divided by the number of binding constraints.

Dynamic "history-dependent" features

Variable-based

- f_{17}: Number of times x_j has been branched on divided by the number of nodes that required branching (so far).
- f_{18}: Number of times x_j appears in the basis divided by number of nodes evaluated (so far).
- f_{19}, f_{20}: Average improvement of the bound on the up and down branches when branching on x_j, i.e., of $|z_j^+ - z|/z$ and $|z_j^- - z|/z$ (so far).
- f_{21}: Average of the reduced cost of x_j divided by z (so far).
- f_{22}, f_{23}, f_{24}: Average pseudo costs (down, up, geometric mean) of x_j divided by z (so far).
- f_{25}: Average of the value of x_j in the solution to the LP relaxation (so far).

Feature Selection

Once a model is trained, XGBoost is able to report an importance score for each feature as shown on the left in Figure 3.3. Our feature selection algorithm uses these importance scores and proceeds as follows. We repeatedly drop features that have the smallest importance score in the current model and train a new model using the remaining features. For

each model, we compute the prediction accuracy, defined as the ratio of the correct ranking predictions and the total number of ranking predictions, where a predicted ranking is deemed correct if an item ranked 1 in the prediction is in the top 5 of the true ranking. We seek to identify a model with a high prediction accuracy and a small number of features. More specifically, we select a model with at least five features and more if a significant drop in accuracy occurs. A plot of the accuracy changes in the feature selection process of learning SB for set covering problems is shown on the right in Figure 3.3 – the model with 5 features is selected.

XGBoost offers five importance scores which can yield different results for our feature selection procedure. We observed that the use of different importance scores usually result in models with similar accuracy even when the features selected are not be same. This suggests that some of the features capture similar information and using different importance scores results in different sets of features that jointly capture the necessary information. Since the resulting accuracy is not sensitive to the type of importance score in use, we decide to use the default importance type 'weight', which is the number of times a feature is used to split the data across all trees, for our feature selection.

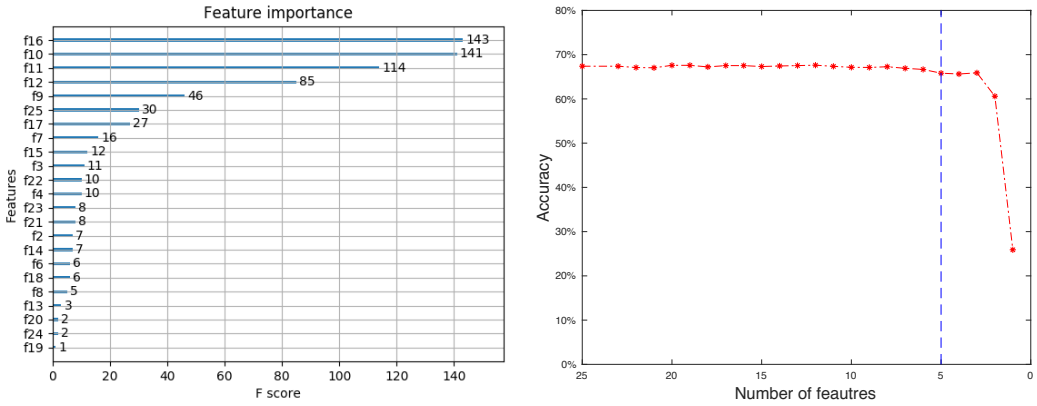


Figure 3.3: Feature selection of learning SB for set covering problems. On the left: importance scores of the original 25 features in the first model trained (features with importance score 0 are not shown). On the right: accuracy changes in the feature selection process.

3.2.2 Learning Generalized Strong Branching

Methodology

Similar to learning SB, we train a model that mimics GSB for each problem class and the learning is performed offline. We start by using GSB (as described in Section 3.1) to solve instances in a training set. For each candidate pair of variables (x_i, x_j), we collect a feature vector $\hat{\mathbf{f}}_{ij}$ and its SB score $\hat{s}_{ij}$. We introduce an artificial pair of variables (x_{i^*}, x_{i^*}) with SB score s_{i^*}, which represents branching on a single variable (i.e., the variable selected by SB). This captures our desire to branch on the variable x_{i^*} if its SB score s_{i^*} is larger than SB score $\hat{s}_{i^*j^*}$ of the the candidate pair of variables (x_{i^*}, x_{j^*}) with the highest score. Again, we train a model LRN-GSB using XGBoost which takes in features $(\hat{\mathbf{f}}_{i^*j_1}, \ldots, \hat{\mathbf{f}}_{i^*j_\ell}, \hat{\mathbf{f}}_{i^*i^*})$ and predicts a ranking $(r_{j_1}, \ldots, r_{j_\ell}, r_{i^*})$. If $r_{i^*} = 1$, then we branch on the variable x_{i^*}, otherwise, we branch on a variable pair (x_{i^*}, x_j) with $r_j = 1$.

Features

Let u be the number of single-variable features resulting from the feature selection process of learning SB in Section 3.2.1. For each candidate pair (x_{i^*}, x_j), we collect the following $(6 + 2u)$ features. The first 6 features are features that seek to capture the interaction of the two variables x_{i^*} and x_j. The next u features are single-variable features of the variable x_j and the last u features are pairwise products of single-variable features of the two variables x_{i^*} and x_j. For the artificial pair (x_{i^*}, x_{i^*}), the first 6 and last u features are set to 0.

Interaction features

- f_1': $\frac{|c_{i^*} - c_j|}{\max_{k\in[n]} c_k - \min_{k\in[n]} c_k}$.
- f_2': Number of times x_{i^*} and x_j appear in the same constraint divided by the number of constraints.
- f_3': Number of times x_{i^*} and x_j appear in the same binding constraint divided by the

number of binding constraints.

- f'_4: $x_{i^*}^{LP} + x_j^{LP} - \lfloor x_{i^*}^{LP} + x_j^{LP} \rfloor$.
- f'_5: Indicator function $\mathbb{1}_{\{x_{i^*}^{LP} + x_j^{LP} > 1\}}$.
- f'_6: Indicator function $\mathbb{1}_{\{i=j\}}$.

Single-variable features

- $f'_7, \ldots, f'_{6+u}$: The u features of the second variable x_j.

Transformed single-variable features

- $f'_{7+u}, \ldots, f'_{6+2u}$: Pairwise product of the u features of the two variables x_{i^*} and x_j.

Feature selection

We go through the same feature selection process as described in Section 3.2.1.

3.3 Computational Study

3.3.1 Settings

We use CPLEX 12.8 for all experiments. To be able to focus on the impact of the branching strategy, we allow cuts to be added only at the root and turn off heuristics. We also turn off root presolve as variable aggregation can significantly alter the structure of an instance. All performance evaluation experiments have a node limit of 10^6 and are conducted using a single thread to ensure a fair comparison. We use $\alpha = 1.5$ and $\beta = 0.5$ in the score function (3.1).

3.3.2 Set Covering Problem

We consider the set covering problem, i.e., integer programs of the following form:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \geq \vec{\mathbf{1}}, \\ & x \in \{0,1\}^n, \end{aligned}$$

where $c \in \mathbb{R}^n$, A is a $m \times n$ (0,1)-matrix. When we branch on the variable pair (x_i, x_j) and $x_i^{LP} + x_j^{LP} \in (0,1)$, then the down branch will have more freedom than the up branch, and the opposite is true when $x_i^{LP} + x_j^{LP} \in (1,2)$. This observation is used when evaluating the score function (3.1).

Instances

We have two classes of instances, OR-Library instances and randomly generated instances. All randomly generated instances have $n = 10m$. The cost c_j of variable x_j for $j = 1, \ldots, n$ is drawn from a discrete uniform distribution on $\{1, 2, \ldots, 100\}$. The number of nonzero elements n_i in row i of A is drawn from a discrete uniform distribution on $\{\frac{2n}{25}+1, \ldots, \frac{3n}{25}-1\}$, and coefficient a_{ij} is set to 1 with probability $\frac{n_i}{n}$. Given that $n = 10m$, we have that the density of A is about 10%.

The details of the instances used in training, evaluation, and testing are shown in Table 3.2, where w indicates the density of the coefficient matrix and q the number of instances in the subset.

Training and feature selection

We use strategy SB to solve instances in training set TS1 to generate data to train LRN-SB. The feature selection process described in Section 3.2.1 results in the following five features being chosen:

1. Current node infeasibility, i.e., $\sum_{k\in[n]} \min\{x_k^{LP} - \lfloor x_k^{LP} \rfloor, \lceil x_k^{LP} \rceil - x_k^{LP}\}$ divided by

Table 3.2: Details of the instances used in training, evaluation, and testing.

TS1	m	n	w	q
Random_a1	300	3000	10%	20
Random_a2	400	4000	10%	20
OR-LIB_a1	200	1000	5%	5
OR-LIB_a2	200	2000	2%	5
OR-LIB_a3	300	3000	5%	5
OR-LIB_a4	300	3000	2%	5

TS2	m	n	w	q
Random_a1	300	3000	10%	20

ES	m	n	w	q
Random_b1	400	4000	10%	10
Random_b2	500	5000	10%	10
Random_b3	600	6000	10%	10
OR-LIB_b1	500	5000	10%	5
OR-LIB_b2	500	5000	20%	5

TEST	m	n	w	q
Random_c1	300	3000	10%	20
Random_c3	400	4000	10%	20
Random_c3	500	5000	10%	20
Random_c4	600	6000	10%	20
OR-LIB_b1	500	5000	10%	5
OR-LIB_b2	500	5000	20%	5

the number of fractional variables in the solution to the LP relaxation.

2. $x_j^{LP} - \lfloor x_j^{LP} \rfloor$.

3. $\lceil x_j^{LP} \rceil - x_j^{LP}$.

4. $\frac{x_j^{LP} \cdot (c_j - \min_{k \in [n]} c_k)}{\max_{k \in [n]} c_k - \min_{k \in [n]} c_k}$.

5. Number of times x_j appears in a constraint that is binding divided by the number of binding constraints.

Next, we use strategy GSB to solve instances in training set TS2, with TS2 a small subset of TS1, to generate to train LRN-GSB. The reason why we use a small subset is that solving instances using strategy GSB is very time-consuming. Feature selection results in the following five features being chosen (Figure 3.4 presents information related to the feature selection process):

1. $x_{i^*}^{LP} + x_j^{LP} - \lfloor x_{i^*}^{LP} + x_j^{LP} \rfloor$.

2. $\frac{x_j^{LP} \cdot (c_j - \min_{k \in [n]} c_k)}{\max_{k \in [n]} c_k - \min_{k \in [n]} c_k}$.

3. Number of times x_j appears in a constraint that is binding divided by the number of binding constraints.

4. $(x_{i^*}^{LP} - \lfloor x_{i^*}^{LP} \rfloor) \cdot (x_j^{LP} - \lfloor x_j^{LP} \rfloor)$.

5. $(\lceil x_{i^*}^{LP} \rceil - x_{i^*}^{LP}) \cdot (\lceil x_j^{LP} \rceil - x_j^{LP})$.

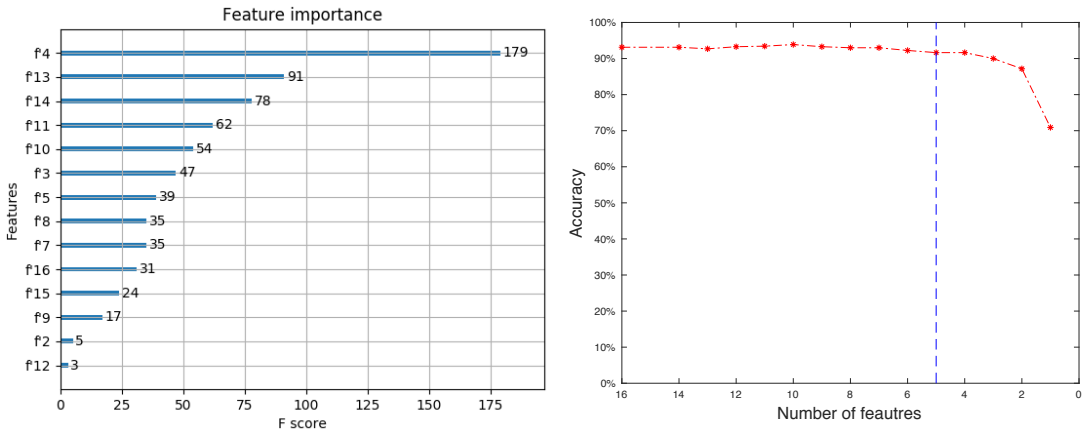


Figure 3.4: Feature selection of learning GSB for set covering problems. On the left: importance scores of the original 16 features in the first model trained (features with importance score 0 are not shown). On the right: accuracy changes in the feature selection process.

Instances in the evaluation set ES are used to tune XGBoost to train LRN-SB and LRN-GSB for better performance. Finally, the performance of trained models is evaluated by solving instances in the test set TEST. Since OR-Library has relatively few instances, the 10 instances in sets OR-LIB_b1 and OR-LIB_b2 are both used in evaluation and testing. For randomly generated instances, the instance used for evaluation and testing are different.

Results

Let CPLEX-D denote CPLEX with default branching, CPLEX-SB denote CPLEX with its implementation of SB, and OUR-SB denote CPLEX with our implementation of SB. To measure the accuracy of the learned models, we use "Top k accuracy%" defined as the ratio of number of times LRN-SB branches on a variable in the top k of the ranking computed by OUR-SB and the total number of nodes in the search tree that were not fathomed.

The results for the instances in the test set can be found in Table 3.3 and 3.4. We report totals for each of the subsets, and, for convenience, also the percentage reduction compared to CPLEX with default branching – the largest percentage reduction is shown in red.

Table 3.3: Comparison of the number of nodes explored for set covering instances.

Name	Number of nodes explored					Top 5
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
Random_c1	17,005	3,415	7,862	11,956	9,877	63.52
Savings	—	79.92%	53.77%	29.69%	41.92%	
Random_c2	83,630	20,673	38,808	55,041	48,796	64.85
Savings	—	75.28%	53.60%	34.19%	41.65%	
Random_c3	677,597	147,493	289,853	407,605	280,568	65.58
Savings	—	78.23%	57.22%	39.85%	58.59%	
Random_c4	6,042,591	1,441,345	4,121,957	2,939,851	2,275,633	66.79
Savings	—	76.15%	31.78%	51.35%	62.34%	
OR-LIB_b1	379,338	91,224	147,818	192,697	124,413	67.31
Savings	—	75.95%	61.03%	49.20%	67.20%	
OR-LIB_b2	502,572	49,353	284,273	130,820	187,969	72.66
Savings	—	90.18%	43.44%	73.97%	62.60%	
Total	7,702,733	1,753,503	4,890,571	3,737,970	2,927,256	65.72
Savings	—	77.24%	36.51%	51.47%	62.00%	

We observe that LRN-SB significantly outperforms CPLEX-D, on average a reduction of more than 50% in terms of the number of nodes explored and of almost 40% in terms of computing time. LRN-GSB performs even better, as, on average, it reduces the number of nodes explored by more than 60% and the computing times by almost 43%. In Figure 3.5, we provide more detail and report the performance of CPLEX-D and LRN-GSB for all 90 test instances. We see that LRN-GSB consistently performs better than CPLEX-D (the values are on a logarithm scale, so the the actual differences are much larger than what is shown in the figure). Detailed results are reported in Table A.1 and A.2 in the Appendix A.

Table 3.4: Comparison of the solution times for set covering instances.

Name	Run time (seconds)				
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
Random_c1	24.01	65.95	222.53	18.42	17.5
Savings	—	-174.68%	-826.82%	23.28%	27.11%
Random_c2	117.77	497.69	1,225.79	93.51	93.78
savings	—	-322.59%	-940.83%	20.60%	20.37%
Random_c3	1,087.91	4,248.86	11,916.94	792.07	658.44
Savings	—	-290.55%	-995.40%	27.19%	39.48%
Random_c4	12,608.06	57,573.18	231,886.18	7,584.87	7,099.62
Savings	—	-356.64%	-1739.19%	39.84%	43.69%
OR-LIB_b1	566.12	2,648.44	6,233.16	374.06	294.43
Savings	—	-367.82%	-1001.03%	33.93%	47.99%
OR-LIB_b2	659.74	1,642.62	25,724.2	259.52	426.65
Savings	—	-148.98%	-3799.14%	60.66%	35.33%
Total	15,063.61	66,676.74	277,208.8	9,122.45	8,590.42
Savings	—	-342.63%	-1740.25%	39.44%	42.97%

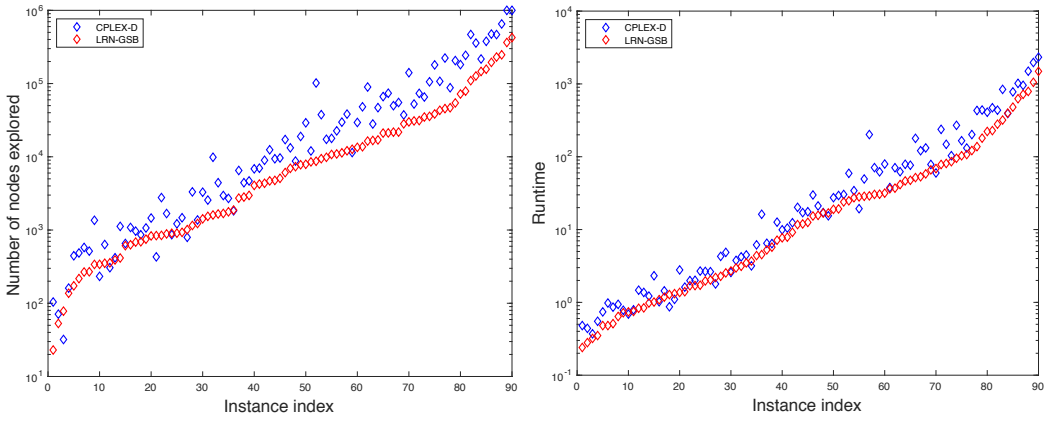


Figure 3.5: Performance of CPLEX-D and LRN-GSB on all instances in the test set (sorted in order of non-decreasing LRN-GSB values). On the left: Number of nodes explored. On the right: Computing time. The y-axis is set to log scale.

3.3.3 Set Packing Problem

Next, we consider the set packing problem, i.e., integer programs of the form:

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & Ax \leq \vec{\mathbf{1}}, \\ & x \in \{0,1\}^n, \end{aligned}$$

where $c \in \mathbb{R}^n$, A is a $m \times n$ (0,1)-matrix. Consider a constraint $\sum_{k \in K} x_k \leq 1$. When we branch on the variable pair (x_i, x_j) with $i, j \in K$, on the down branch x_i and x_j are fixed to 0, but on the up branch all variables x_k with $k \in K \setminus \{i, j\}$ are fixed to 0. Thus, the down branch has more freedom. This observation is used when evaluating the score function (3.1).

Instances

Instances are randomly generated in the same way as in Section 3.3.2 except that $n = 5m$. The details of the instances used in training, evaluation, and testing are shown in Table 3.5.

Table 3.5: Details of the instances used in training, evaluation, and testing.

TS1	m	n	w	q
Random_d1	100	500	10%	20
Random_d2	150	750	10%	20
Random_d3	200	1000	10%	20

ES	m	n	w	q
Random_f1	150	750	10%	10
Random_f2	200	1000	10%	10
Random_f3	250	1250	10%	10

TS2	m	n	w	q
Random_e1	100	500	10%	10
Random_e2	150	750	10%	10

TEST	m	n	w	q
Random_g1	100	500	10%	20
Random_g2	150	750	10%	20
Random_g3	200	1000	10%	20
Random_g4	250	1250	10%	20
Random_g5	300	1500	10%	20

Training and feature selection

Using feature selection as described in Section 3.2.1, the following five features are selected for training LRN-SB:

1. Number of constraints in which x_j appears divided by the number of constraints.
2. $x_j^{LP} - \lfloor x_j^{LP} \rfloor$.
3. $\lceil x_j^{LP} \rceil - x_j^{LP}$.
4. Number of times x_j appears in a constraint that is binding divided by the number of binding constraints.
5. Average of the value of x_j in the solution to the LP relaxation (so far).

Similarly, the following five features are selected for training LRN-GSB.

1. Number of times x_{i^*} and x_j appear in the same constraint divided by the number of constraints.
2. $x_{i^*}^{LP} + x_j^{LP} - \lfloor x_{i^*}^{LP} + x_j^{LP} \rfloor$.
3. $x_j^{LP} - \lfloor x_j^{LP} \rfloor$.
4. $(x_{i^*}^{LP} - \lfloor x_{i^*}^{LP} \rfloor) \cdot (x_j^{LP} - \lfloor x_j^{LP} \rfloor)$.
5. $(\lceil x_{i^*}^{LP} \rceil - x_{i^*}^{LP}) \cdot (\lceil x_j^{LP} \rceil - x_j^{LP})$.

Results

The results on the instances in the test set can be found in Table 3.6 and Table 3.7. We see that LRN-GSB outperforms CPLEX-D both in terms of the number of nodes explored and the computing time. However, the improvements are not as significant, on average, a 44% reduction in the number of nodes explored and a 12% reduction in computing time. Although LRN-GSB, on average, explores a smaller number of nodes than LRN-SB, its computing time, on average, is slightly higher than LRN-SB. This is because more time is spent on each branching decision. Furthermore, possibly more importantly, LRN-GSB adds constraints to enforce branching decisions, whereas LRN-SB only changes variables

Table 3.6: Comparison on the number of nodes explored for set packing instances.

Name	Number of nodes explored					Top 5
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
Random1_g1	35,489	5,604	13,348	18,008	21,659	57.24
savings	—	84.21%	62.39%	49.26%	38.97%	
Random1_g2	187,742	44,354	79,289	105,175	100,326	67.53
savings	—	76.38%	57.77%	43.98%	46.56%	
Random1_g3	217,579	72,191	108,008	134,939	124,652	77.43
savings	—	66.82%	50.36%	37.98%	42.71%	
Random1_g4	344,339	103,481	165,997	197,382	203,088	81.77
savings	—	69.95%	51.79%	42.68%	41.02%	
Random1_g5	345,463	104,276	168,075	184,140	183,097	83.91
savings	—	69.82%	51.35%	46.70%	47.00%	
Total	1,130,612	329,906	534,717	639,644	632,822	73.58
Savings	—	70.82%	52.71%	43.42%	44.03%	

Table 3.7: Comparison of solutions times for set packing instances.

Name	Run time (seconds)				
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
Random1_g1	24.44	64.6	137.54	14.75	19.26
savings	—	-164.32%	-462.77%	39.65%	21.19%
Random1_g2	217.02	823.88	1,576.46	159.92	166.8
savings	—	-279.63%	-626.41%	26.31%	23.14%
Random1_g3	549.65	1,993.62	4,039.72	434.57	442.44
savings	—	-262.71%	-634.96%	20.94%	19.51%
Random1_g4	991.08	3,714.48	9,016.01	833.85	862.34
savings	—	-274.79%	-809.72%	15.86%	12.99%
Random1_g5	1,682.15	6,104.38	17,382.97	1,511.29	1,548.72
savings	—	-262.89%	-933.38%	10.16%	7.93%
Total	3,464.34	12,700.96	32,152.7	2,954.38	3,039.56
Savings	—	-266.62%	-828.10%	14.72%	12.26%

bounds to enforce branching decisions. Another interesting observation is that the prediction accuracy of LRN-SB for instances of the set packing problem appears to be higher than for instances of the set covering problem, which suggests that LRN-SB already performs very well and it is more difficult for LRN-GSB to achieve further improvements. More detailed results are reported in Table A.3 and A.4 in the Appendix A.

3.3.4 0-1 Knapsack Problem

Next, we consider the 0-1 knapsack problem, i.e., integer programs of the form:

$$\begin{aligned} \max \quad & p^T x \\ \text{s.t.} \quad & w^T x \leq b, \\ & x \in \{0,1\}^n, \end{aligned}$$

where $p, w \in \mathbb{R}^n$. We make a minor change in the way we select candidate pairs of variables. Rather than requiring that $x_j^{LP} > 0$, we require that x_j is not fixed, i.e., its lower and upper bound at the current node are not equal.

Instances

All instances are randomly generated as follows. The profit p_j and the weight w_j for $j = 1, \ldots, n$ are drawn from a discrete uniform distribution on $\{1, 2, \ldots, 10n\}$ and we set $b = \lfloor \sum_{i=1}^{n} w_i/5 \rfloor$. The details of the instances used in training, evaluating, and testing are shown in Table 3.8.

Training and feature selection

The five features selected for training LRN-SB are as follows:

1. $\frac{c_j - \min_{k\in[n]} c_k}{\max_{k\in[n]} c_k - \min_{k\in[n]} c_k}$.
2. Mean of $\frac{A_{ij}}{\sum_{k\in[n]} A_{ik}/n}$ w.r.t $i \in [m]$.
3. Max of $\frac{A_{ij}}{\sum_{k\in[n]} A_{ik}/n}$ w.r.t $i \in [m]$.

Table 3.8: Details of the instances used in training, evaluation, and testing.

TS1	n	q
Random_h1	2000	20
Random_h2	3000	20
Random_h3	4000	20

ES	n	q
Random_j1	2000	10
Random_j2	3000	10

TS2	n	q
Random_i1	2000	10
Random_i2	3000	10

TEST	n	q
Random_k1	2000	20
Random_k2	3000	20
Random_k3	4000	20
Random_k4	5000	20
Random_k5	6000	20

4. Up pseudo-cost of x_j divided by z.

5. Average of value of x_j in the solution to the LP relaxation (so far).

The five features selected for training LRN-GSB are as follows:

1. Number of times x_{i^*} and x_j appear in the same constraint divided by the number of constraints.

2. $x_{i^*}^{LP} + x_j^{LP} - \lfloor x_{i^*}^{LP} + x_j^{LP} \rfloor$.

3. Indicator function $\mathbb{1}_{\{x_{i^*}^{LP} + x_j^{LP} > 1\}}$.

4. $\frac{c_j - \min_{k \in [n]} c_k}{\max_{k \in [n]} c_k - \min_{k \in [n]} c_k}$.

5. Product of up pseudo-cost of x_{i^*} and x_j divided by z^2.

Results

Even though there is only a single constraint in the 0-1 knapsack problem (which should imply that there is only a single fractional variable in a solution to the LP relaxation), CPLEX-D, CPLEX-SB, OUR-SB, RAND, and LRN-SB can still exhibit different behavior because we allow cuts to be added at the root. However, we have observed that the number of cuts added at the root is very small and so is the number of fractional variables – usually fewer than five. Therefore, we report "Top 1 accuracy%" instead of "Top 5 accuracy%".

Table 3.9: Comparison of the number of nodes explored for 0-1 knapsack instances.

Name	Number of nodes explored					Top 1 Accuracy%
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	
Random_k1	33,813	121,798	27,219	25,862	21,056	99.62
Savings	—	-260.21%	19.50%	23.51%	37.73%	
Random_k2	269,145	113,748	37,421	36,372	50,045	99.06
Savings	—	57.74%	86.10%	86.49%	81.41%	
Random_k3	128,406	153,494	51,736	52,593	55,048	99.39
savings	—	-19.54%	59.71%	59.04%	57.13%	
Random_k4	401,585	151,681	69,477	65,193	34,219	99.46
savings	—	62.23%	82.70%	83.77%	91.48%	
Random_k5	403,323	441,979	78,708	86,559	45,403	99.57
savings	—	-9.58%	80.49%	78.54%	88.74%	
Total	1,236,272	982,700	264,561	266,579	205,771	99.42
Savings	—	20.51%	78.60%	78.44%	83.36%	

Table 3.10: Comparison of computing times for 0-1 knapsack instances.

Name	Run time (seconds)				
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
Random_k1	13.46	34.96	46.79	19.96	18.07
Savings	—	-159.73%	-247.62%	-48.29%	-34.25%
Random_k2	88.12	47.39	289.44	41.65	51.11
Savings	—	46.22%	-228.46%	52.73%	42.00%
Random_k3	80.64	72.51	463.88	72.74	72.23
Savings	—	10.08%	-475.25%	9.80%	10.43%
Random_k4	248.98	98.73	698.41	112.33	75.78
Savings	—	60.35%	-180.51%	54.88%	69.56%
Random_k5	301.1	303.76	870.23	172.96	115.32
Savings	—	-0.88%	-189.02%	42.56%	61.70%
Total	732.3	557.35	2,368.75	419.64	332.51
Savings	—	23.89%	-223.47%	42.70%	54.59%

The results on the instances in the test set can be found in Table 3.9 and Table 3.10. We observe, again, that LRN-SB and LRN-GSB significantly outperform CPLEX-D in terms of both number of nodes explored and computing time, where LRN-GSB is, again, better than LRN-SB. What is most surprising is that CPLEX-SB has the worst performance, even in terms of number of nodes explored. Detailed results can be found in Table A.5 and A.6 in the Appendix A.

Default Settings

In the computational experiments present in the previous subsections, we have turned off CPLEX root preprocessing, CPLEX heuristics, and CPLEX cut generation in nodes other than the root, in order to be able to "isolate" the effect of the branching strategy. One may argue that provides only limited information on the value of the proposed learned branching strategies. Therefore, we have rerun all set covering and set packing instances, but this time with all default settings of CPLEX, except for the branching scheme. With default CPLEX settings, the 0-1 knapsack instances solve in only a few nodes, so not much can be learned from these instances, which is why we are not reporting results for them.

Set Covering Problem

The results of the computational experiments can be found in Table 3.11 and 3.12. We observe that LRN-SB and LRN-GSB still significantly outperform CPLEX-D, with, on average, a reduction of 49% and 51% in computing time, respectively. In fact, the improvements are even greater!

Set Packing Problems

The results of the computational experiments can be found in Table 3.13 and 3.14. As with the set covering instances, LRN-SB and LRN-GSB still significantly outperform CPLEX-D, with, on average, a reduction of 54% and 24% in computing time, respectively.

Table 3.11: Comparison of the number of nodes explored for set covering instances (default CPLEX settings).

	Number of nodes explored					Top 5
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
Random_c1	9,238	2,643	5,191	8,559	6,695	62.04
Savings	—	71.39%	43.81%	7.35%	27.53%	
Random_c2	62,769	17,474	30,884	41,185	39,026	58.98
savings	—	72.16%	50.80%	34.39%	37.83%	
Random_c3	540,532	126,479	221,347	334,208	277,814	54.06
Savings	—	76.60%	59.05%	38.17%	48.60%	
Random_c4	5,987,841	1,091,251	1,802,029	2,487,096	2,024,740	49.36
Savings	—	81.78%	69.91%	58.46%	66.19%	
OR-LIB_b1	473,797	54,144	122,734	205,697	83,895	51.53
Savings	—	88.57%	74.10%	56.59%	82.29%	
OR-LIB_b2	300,003	67,455	197,604	90,375	340,769	54.23
Savings	—	77.52%	34.13%	69.88%	-13.59%	
Total	7,374,180	1,359,446	2,379,789	3,167,120	2,772,939	55.75
Savings	—	81.56%	67.73%	57.05%	62.40%	

Table 3.12: Comparison of computing times for set covering instances (default CPLEX settings).

	Run time (seconds)				
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
Random_c1	24.21	43.32	71.55	21.92	20.83
Savings	—	-78.93%	-195.54%	9.46%	13.96%
Random_c2	96.3	295.66	433.41	78.23	82.74
Savings	—	-207.02%	-350.06%	18.76%	14.08%
Random_c3	772.68	2,611.14	3,846.96	582.49	556.84
Savings	—	-237.93%	-397.87%	24.61%	27.93%
Random_c4	10,841.31	31,735.29	39,472.03	5,326.21	4,939.62
Savings	—	-192.73%	-264.09%	50.87%	54.44%
OR-LIB_b1	636.05	1,162.46	2,270.96	354.63	166.95
Savings	—	-82.76%	-257.04%	44.24%	73.75%
OR-LIB_b2	388.75	1,384.31	3,286.55	149.67	496.78
Savings	—	-256.09%	-745.41%	61.50%	-27.79%
Total	12,759.3	37,232.18	49,381.46	6,513.15	6,263.76
Savings	—	-191.80%	-287.02%	48.95%	50.91%

Table 3.13: Comparison of number of nodes explored for set packing instances (default CPLEX settings).

	Number of nodes explored					Top 5
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
Random_g1	26,189	4,654	11,070	15,937	15,299	57.24
Savings	—	82.23%	57.73%	39.15%	41.58%	
Random_g2	151,127	39,372	62,337	79,206	78,037	67.53
Savings	—	73.95%	58.75%	47.59%	48.36%	
Random_g3	213,988	56,722	84,836	94,442	96,379	77.43
Savings	—	73.49%	60.35%	55.87%	54.96%	
Random_g4	328,638	76,617	138,657	145,188	147,282	81.77
Savings	—	76.69%	57.81%	55.82%	55.18%	
Random_g5	315,255	77,508	138,175	143,052	142,063	83.91
Savings	—	75.41%	56.17%	54.62%	54.94%	
Total	1,035,197	254,873	435,075	477,825	479,060	73.58
Savings	—	75.38%	57.97%	53.84%	53.72%	

Table 3.14: Comparison of computing time for set packing instances (default CPLEX settings).

Name	Run time (seconds)				
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
Random_g1	28.11	61.58	118.58	20.6	21.77
savings	—	-119.07%	-321.84%	26.72%	22.55%
Random_g2	244.17	778.98	1,322.18	167.79	176.02
savings	—	-219.03%	-441.50%	31.28%	27.91%
Random_g3	674.14	1,879.89	3,731.11	503.74	518.55
savings	—	-178.86%	-453.46%	25.28%	23.08%
Random_g4	1,312.1	3,645.31	9,613.23	985.79	1,004.31
savings	—	-177.82%	-632.66%	24.87%	23.46%
Random_g5	2,284.04	5,989.27	17,901.58	1,704.14	1,737.75
savings	—	-162.22%	-683.77%	25.39%	23.92%
Total	4,542.56	12,355.03	32,686.68	3,382.06	3,458.4
Savings	—	-171.98%	-619.57%	25.55%	23.87%

3.4 Final Remarks

We have shown that complex branching strategies, e.g., strong branching and generalized strong branching, can be implemented efficiently by incorporating a model that mimics their behavior and is learned offline. In fact, the resulting performance, on specific classes of integer programs, is not only competitive with the branching strategies found in state-of-the-art commercial solver, they outperform these branching strategies significantly (even given the disadvantages of having to implement these branching strategies using callback functions and not having access to customized internal data structures). Equally important is the fact that the learned models are robust in the sense that even though they are trained on only small (easy) instances, the trained models are able to solve large (difficult) instances effectively and efficiently.

CHAPTER 4

INTEGER PROGRAMMING FOR SERVICE NETWORK DESIGN WITH IN-TREE CONSTRAINTS

The *service network design problem with in-tree constraints* (SNDPITC) seeks a minimum-cost transportation plan for moving multiple less-than-truckload (LTL) shipments (or commodities) from their origins to their destinations. In addition, if any shipments sharing the same destination should meet at any point in their respective paths, then they have to travel on the same path thereafter.

More formally, let $G = (N, A)$ be a directed graph with node set N and arc set A. Let the positive cost of a truck traversing arc $a \in A$ be $c_a \in \mathbb{R}_{>0}$. Let K denote the set of all commodities, $o(k)$, $d(k) \in N$ denote the origin and destination for commodity $k \in K$, respectively, and $q(k) \in (0, Q]$ denote its quantity, where Q is the capacity of a truck. In what follows, we take $Q = 1$ and commodity quantities are scaled accordingly. The SNDPITC seeks a shipping plan where each commodity $k \in K$ follows a single path in G from $o(k)$ to $d(k)$. Furthermore, for any $k_1, k_2 \in K$ such that $d(k_1) = d(k_2)$, if the paths for k_1 and k_2 visit a common node $n \in N$, then both paths must depart n via the same arc $a \in A$. The cost of the shipping plan (purely the transportation cost), which the SNDPITC seeks to minimize, is modeled by

$$\sum_{a \in A} c_a \cdot \left\lceil \sum_{k \in K : a \in P_k} q(k) \right\rceil$$

where $P_k \subseteq A$ denotes the arcs in the path for shipping commodity k.

The remainder of the chapter is organized as follows. In Section 4.1, we present three Steiner in-tree formulations and three IP formulations for the SNDPITC. In Section 4.2, we compare the size and strength of the LP relaxation of these formulations. Some simple

strengthening inequalities are present in Section 4.3. In Section 4.4, we derive six new classes of cutting planes for the flow-based SNDPITC formulation. The separation challenges and heuristics are also discussed. In Section 4.5, we present and analyze the results of a numerical study. Finally, in Section 4.6, we draw conclusions and give further research directions.

4.1 Problem Formulations

4.1.1 Notation

First, we introduce some additional notation that is used throughout the remainder of the chapter:

- $K(d) := \{k \in K : \ d(k) = d\}$ is the set of all commodities k with destination d;
- $K^a \subseteq K$ is the set of commodities that may use a path containing arc a;
- $K_V := \{k \in K : \ o(k) \in V, \ d(k) \in \bar{V}\}$ is the set of commodities with origin in V and destination in $\bar{V}$, where $\bar{V} := N \setminus V$;
- $K(a, d) = K^a \cap K(d)$ is the set of commodities with destination d that are allowed to use a path containing arc a;
- $D := \{d(k) : \ k \in K\} \subseteq N$ is the set of all destinations;
- $O(d) := \{o(k) : \ d(k) = d\}$ is the set of all nodes that are an origin for a commodity with destination d;
- $O := \{O(d) : d \in D\}$ is the set of all origins;
- $\delta^-(i)$ is the set of incoming arcs to node $i \in N$ in the network (N, A);
- $\delta^+(i)$ is the set of outgoing arcs from node $i \in N$ in the network (N, A); and
- $[n] = \{1, 2, \cdots, n\}$ for any $n \in \mathbb{Z}_{>0}$.

4.1.2 Steiner In-tree Formulations

The SNDPITC embeds a Steiner in-tree problem for each node that is a commodity destination: the union of paths for all commodities having the same destination node must induce a Steiner in-tree in the network, connecting the set of commodity origins with their common destination. In this section, we review existing approaches to modeling these this subproblem, and consider one alternative that is natural in the SNDPITC setting. For the remainder of this section, we fix a single destination node, $d \in D$, and consider IP formulations to ensure that the paths for all commodities in $K(d)$ satisfy the in-tree requirement.

Path-Only Formulation

The most direct approach is to use a binary flow variable, x_{ak}, to indicate whether an arc a is used ($x_{ak} = 1$) or not ($x_{ak} = 0$) to deliver commodity k, for each $a \in A$ and $k \in K(d)$. The following formulation, which we call the *path-only formulation*, directly models the requirement that if any two of these commodity paths meet at a node, then they must follow the same path to their destination thereafter.

$$\sum_{a \in \delta^+(i)} x_{ak} - \sum_{a \in \delta^-(i)} x_{ak} = \begin{cases} 1, & \text{if } i = o(k), \\ -1, & \text{if } i = d, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall i \in N,\ \forall k \in K(d), \tag{4.1}$$

$$x_{ak} + \sum_{\substack{a' \in \delta^+(i), \\ a' \neq a}} x_{a'k'} \leq 1 \quad \forall i \in N,\ \forall a \in \delta^+(i),\ \forall k, k' \in K(d) \text{ with } k \neq k', \tag{4.2}$$

$$x_{ak} \in \{0, 1\}, \quad \forall a \in A,\ \forall k \in K(d). \tag{4.3}$$

Constraints (4.1) and (4.3) ensure that $(x_{ka})_{a \in A}$ induces a path from $o(k)$ to d for each $k \in K(d)$. Note that, in the absence of an objective function with positive costs to the x_{ak} variables, extraneous cycles may also be included. Constraint (4.2) models the requirement

that if two distinct commodity paths meet at a node, they must use the same arc leaving that node. We denote the set of feasible solutions to these constraints by

$$S^1 := \{x \in \{0,1\}^{|A|\times|K(d)|} \ : \ x \text{ satisfies (4.1) and (4.2)}\}.$$

Path-Tree Formulation

The requirement of a common path after commodities meet can also be modeled as the problem of finding a Steiner arborescence $T \subseteq A$, rooted at d, such that T contains a path from i to d for all $i \in O(d)$, the set of target nodes. Note an arborescence is usually defined as directed from its root node towards the target nodes, so all nodes have in-degree at most one. Here, we use an equivalent definition in which all arcs are reversed, away from the target nodes and towards the root, so all nodes have out-degree at most one. Hence the arborescence we seek is actually an in-tree. The formulation, which we call the *path-tree formulation*, consisting of

$$x \text{ satisfies (4.1) and (4.3)},$$

$$x_{ak} \leq y_a, \quad \forall a \in A, \ \forall k \in K(d), \tag{4.4}$$

$$\sum_{a\in\delta^+(i)} y_a \leq 1, \qquad \forall i \in N, \tag{4.5}$$

$$y_a \in \{0,1\}, \quad \forall a \in A, \tag{4.6}$$

appears frequently in the literature [see 47, 53], where y represents the indicator vector for the desired in-tree T. Given a cost for each arc, the Steiner arborescence problem seeks to minimize the total costs of the arcs in T. This problem is known to be *NP-hard* [51]. We denote the set of feasible solutions to the above constraints by

$$S^2 := \{(x,y) \in \{0,1\}^{|A|\times|K(d)|+|A|} : \ (x,y) \text{ satisfies (4.1), (4.5), (4.4)}\}.$$

It is obvious that $(x, y) \in S^2$ if and only if $x \in S^1$.

Tree-Only Formulation

For the SNDPITC, the number of commodities, and hence the number of commodity flow variables in x, can grow prohibitively large for IP formulations to succeed. Thus we are especially interested in formulations for the Steiner in-tree rooted at d that do not require commodity flow variables, and hence are much more compact. One such formulation uses only the tree indicator variables and employs a constraint for each d-*cut*, defined to be any set $C \subseteq N \setminus \{d\}$ with $C \cap O(d) \neq \emptyset$. The d-*cut constraints* are defined as follows

$$\sum_{a \in \delta^+(C)} y_a \geq 1, \quad \text{for each } d\text{-cut } C, \tag{4.7}$$

where $\delta^+(C) = \{(i, j) \in A \ : \ i \in C, \ j \in N \setminus C\}$. We call this formulation the *tree-only formulation* and denote the resulting feasible region by

$$S^3 := \{y \in \{0, 1\}^{|A|} : \ y \text{ satisfies (4.5) and (4.7)}\}.$$

Polyhedral aspects of this formulation have been discussed in [40], [47], and others. A branch-and-cut algorithm for solving Steiner tree problems based on this formulation has been proposed in [53].

4.1.3 SNDPITC Formulations

This section presents three IP formulations for the SNDPITC. Each formulation uses n_a, for each $a \in A$, a non-negative integer variable representing the number of truckloads needed on arc a. This variable, along with additional constraints, allows us to model a linear objective function. The first two of the three formulations use the path-only and path-tree Steiner in-tree formulations from Section 4.1.2, respectively, for each destination node, $d \in D$. We use $S^1(d)$ and $S^2(d)$ to denote these two Steiner in-tree formulations for the

destination d, respectively. The third formulation is very compact: it does not require any variables indexed by commodity. Instead, it uses only continuous flow variables for each destinations together with tree arc indicator variables. The connectivity of commodity origins to their destination is ensured by the continuous flow variables together with a capacity-class constraint linking them to their corresponding tree arc variables. The in-tree requirement is forced by the out-degree constraints as well as the integrality of the tree arc indicator variables.

Commodity-based Formulation

This formulation is based on the path-only Steiner in-tree formulation: let x_{ak} for arc a in the network be a binary variable indicating whether the arc is used ($x_{ak} = 1$) or not ($x_{ak} = 0$) in the path for commodity k. We write x^d to denote the vector in $\mathbb{R}^{|A|\times|K(d)|}$ corresponding to $(x_{ak})_{a\in A,k\in K(d)}$. Our IP Formulation F1 is as follows:

$$\begin{aligned}
\min \quad & z = \sum_{a\in A} c_a n_a \\
\text{s.t.} \quad & x^d \in S^1(d), \qquad \forall d \in D, \\
& n_a \geq \sum_{k\in K} q_k x_{ak}, \quad \forall a \in A, && (4.8) \\
& n_a \in \mathbb{Z}_{\geq 0}, \quad \forall a \in A. && (4.9)
\end{aligned}$$

Constraints (4.8) and (4.9) ensure that there are enough trucks to transport the amount of freight on each arc.

Destination-based Formulation

This formulation is based on the path-tree Steiner in-tree formulation: binary variable y_{ad} indicates whether arc a is used ($y_{ad} = 1$) or not ($y_{ad} = 0$) in the in-tree for commodities with destination d. We write y^d to denote the vector in $\mathbb{R}^{|A|}$ corresponding to $(y_{ad})_{a\in A}$. Our

IP Formulation F2 is given below:

$$\min \left\{ z = \sum_{a \in A} c_a n_a : \ (x, y) \in S^2(d), \ \forall d \in D, \ n \text{ satisfies } (4.8) \text{ and } (4.9) \right\}.$$

Flow-based Formulation

This formulation introduces the continuous variable $w_{ad} \in \mathbb{R}_{\geq 0}$ indicating the quantity flowing on arc a that is destined for d, which replaces the binary x_{ak} variable for $k \in K(d)$. Our IP Formulation F3 is as follows:

$$\begin{aligned}
\min \quad & z = \sum_{a \in A} c_a n_a \\
\text{s.t.} \quad & \sum_{a \in \delta^+(i)} w_{ad} - \sum_{a \in \delta^-(i)} w_{ad} = \begin{cases} \sum\limits_{\substack{k \in K(d), \\ o(k) = i}} q_k, & \text{if } i \in O(d), \\ -\sum\limits_{k : d(k) = d} q_k, & \text{if } i = d, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall i \in N, \ \forall d \in D, \qquad (4.10) \\
& \sum_{a \in \delta^+(i)} y_{ad} \leq 1, \quad \forall i \in N, \ \forall d \in D, \\
& w_{ad} \leq \left(\sum_{k \in K(d)} q_k \right) y_{ad}, \quad \forall a \in A, \ \forall d \in D, \qquad (4.11) \\
& n_a \geq \sum_{d \in D} w_{ad}, \quad \forall a \in A, \qquad (4.12) \\
& n_a \in \mathbb{Z}_{\geq 0}, \quad \forall a \in A, \\
& w_{ad} \geq 0, \quad \forall a \in A, \ \forall d \in D, \qquad (4.13) \\
& y_{ad} \in \{0, 1\}, \quad \forall a \in A, \ \forall d \in D.
\end{aligned}$$

Constraint (4.10) is the flow balance constraint and Constraint (4.11) models the relationship between the w_{ad} and y_{ad} variables. Constraints (4.12) and (4.9) ensure that there are enough trucks to transport the amount of freight on each arc.

4.2 Comparing the Size and Strength of Formulations

4.2.1 For the Steiner In-Tree Formulations

The aforementioned three Steiner In-tree formulations have quite different numbers of variables and constraints. The path-only formulation, with feasible set S^1, has a large number of variables ($|A||K|$) and a very large number of constraints ($\mathcal{O}(|A||K(d)|^2)$). The path-tree formulation, with feasible set S^2, has $|A|$ more variables, but has far fewer constraints ($\mathcal{O}(|A||K(d)|)$). The tree-only formulation, with feasible set S^3, has very few variables ($|A|$) but has an exponentially large number of constraints.

Here we consider the comparative strength of the LP relaxation of these formulations, analytically. We find that the path-only formulation is surprisingly weak. Although it can be strengthened, it still does not attain the strength of the other two formulations. We also confirm that the path-tree and tree-only formulations have equivalent LP relaxations. Although the basis of this observation has been established in [10], and mentioned in [47], there is some variation in the details of the models considered, so we provide a careful proof. We give this result first.

We denote by $LP(F)$ the feasible region of the LP relaxation for the formulation having feasible set F, and by $Proj_v LP(F)$ its projection onto the v variables.

Proposition 4.2.1. $Proj_y LP(S^2) = LP(S^3)$

Proof. Assume that $(x, y) \in LP(S^2)$. To show that $y \in LP(S^3)$, it suffices to show that the d-cut constraints (4.7) are satisfied. For any C with $d \notin C$ and $O(d) \cap C \neq \emptyset$, there

exists $k \in K(d)$ with $o(k) \in O(d) \cap C$. Thus, we have

$$\begin{aligned}
\sum_{a \in \delta^+(C)} y_a &\geq \sum_{a \in \delta^+(C)} x_{ak} \\
&\geq \sum_{a \in \delta^+(C)} x_{ak} - \sum_{a \in \delta^-(C)} x_{ak} \\
&= \sum_{i \in C} \left(\sum_{a \in \delta^+(i)} x_{ak} - \sum_{a \in \delta^-(i)} x_{ak} \right) \\
&= 1,
\end{aligned}$$

where the first inequality is due to (4.4), and the last equality is due to (4.1) and the fact that $o(k) \in C$ and $d \notin C$. Thus $Proj_y LP(S^2) \subseteq LP(S^3)$.

To prove the converse, we observe that for a given, fixed, $y \in LP(S^3)$, there exists x such that $(x, y) \in LP(S^2)$ if and only if for each $k \in K(d)$, some vector $(x_{ak})_{a \in A}$ induces a flow of one unit from $o(k)$ to d, where the flow capacity of arc $a \in A$ is given by y_a. In other words, $y \in Proj_y LP(S^2)$ if and only if the solution to the maximum flow problem with source $o(k)$, sink d and arc capacities given by y must have value at least one, for each k. Since the d-cut constraints ensure that the capacities given by y are at least one for any cut separating $o(k)$ from d for some $k \in K(d)$, the result follows from the max flow-min cut theorem. ■

We note that the d-cut constraints can thus be interpreted as Benders cuts, if formulation S^2 is approached via Benders decomposition. Furthermore, the same max flow-min cut argument as is given in the proof above shows that separation of the d-cut constraints can be done by solving a max flow problem for each $k \in K(d)$, with arc capacities given by y. This is exploited in [53], for example, in their branch-and-cut approach.

We now consider the strength of $LP(S^1)$. We first show that it is strictly weaker than the other formulations, in general.

Proposition 4.2.2. $Proj_x LP(S^2) \subsetneq LP(S^1)$

Proof. We first show the inclusion relation holds and then give a specific example that shows the containment is strict.

Assume $(x, y) \in S^2$. To show that $x \in S^1$, it suffices to show that the in-tree constraints (4.2) are satisfied. In view of (4.4) and (4.5), we have, for any $i \in N$, $a \in \delta^+(i)$, and $k, k' \in K(d)$ with $k \neq k'$, that

$$x_{ak} + \sum_{\substack{a' \in \delta^+(i), \\ a' \neq a}} x_{a'k'} \leq y_a + \sum_{\substack{a' \in \delta^+(i), \\ a' \neq a}} y_{a'} = \sum_{a' \in \delta^+(i)} y_{a'} \leq 1.$$

Thus, $Proj_x LP(S^2) \subseteq LP(S^1)$.

Now, consider an instance with three commodities, k_1, k_2, k_3, each having a different origin, but the same destination, d. Figure 4.1 contains a snapshot of the network at some node i that is neither an origin nor the destination for these three commodities.

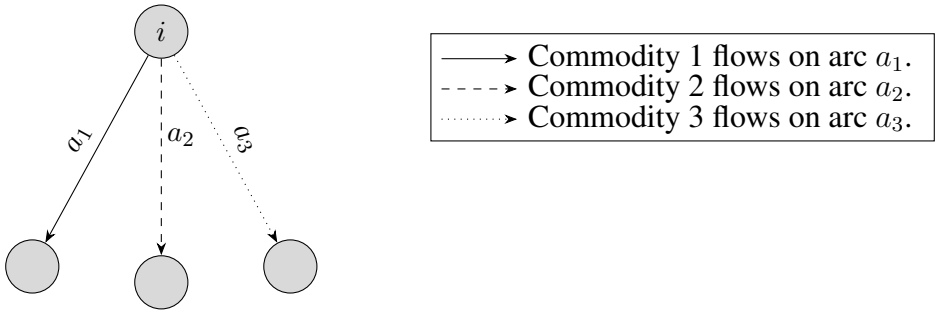


Figure 4.1: Three outgoing arcs at node i.

Each commodity has entered node i, and is leaving node i via distinct arcs, a_1, a_2, a_3, none of which end at the destination. Assume that $x_{a_1k_1} = \frac{3}{8}$, $x_{a_2k_2} = \frac{3}{8}$, and $x_{a_3k_3} = \frac{3}{8}$. These values can appear in a complete feasible solution of $LP(S^1)$ (see Section B.1) since (4.2) only applies to two commodities at a time. However, x_{ak} values and (4.4) force $y_{a_1} \geq \frac{3}{8}$, $y_{a_2} \geq \frac{3}{8}$, and $y_{a_3} \geq \frac{3}{8}$. Thus, Constraint (4.5) is violated since $\sum_{a \in \delta^+(i)} y_a \geq \frac{9}{8} > 1$, which completes the proof. ■

The above example, which shows that the path-only formulation is strictly weaker than

the path-tree formulation, uses three commodities and three distinct arcs with the same tail node. In fact, S^1 has a weaker LP relaxation than S^2 if and only if there are at least two commodities in $K(d)$ and at least three arcs with the same tail node. Otherwise, the two formulations are equivalent.

Before proving this, we first show how to strengthen the path-only formulation. Observe that, for any given node i and $a \in \delta^+(i)$, the two sets $\{a\}$ and $\{a' \in \delta^+(i) : \ a' \neq a\}$ form a partition of $\delta^+(i)$. Constraint (4.2) considers flow of commodity k on the arcs in the first set of this partition and flow of commodity k' on the arcs in the second set. This can be generalized as follows. Let $\tilde{K} \subseteq K(d)$ have cardinality $p = |\tilde{K}|$ with $2 \leq p \leq |\delta^+(i)|$ and let $\Delta_i : \tilde{K} \to 2^{\delta^+(i)} \setminus \{\emptyset\}$ be a function mapping each commodity in $\tilde{K}$ to a non-empty subset of $\delta^+(i)$ such that sets $\{\Delta_i(k)\}_{k \in \tilde{K}}$ form a partition of $\delta^+(i)$. We call such a pair $(\tilde{K}, \Delta_i)$ a *commodity-out-i partition pair*. It's easy to see that the constraint

$$\sum_{k \in \tilde{K}} \sum_{a \in \Delta_i(k)} x_{ak} \leq 1, \qquad \forall i \in N, \ \forall \text{ commodity-out-}i \text{ partition pairs } (\tilde{K}, \Delta_i), \tag{4.14}$$

is valid for S^1. (If commodity $k \in \tilde{K}$ uses an arc $a \in \Delta_i(k)$, then all other commodities leaving i must use the same arc, and no other arcs in $\Delta_i(k')$ can be used for any $k' \neq k$.) We define the strengthened path-only formulation to be

$$\tilde{S}^1 := \{x \in \{0,1\}^{|A| \times |K(d)|} : \ x \text{ satisfies (4.1) and (4.14)}\}.$$

It is obvious that $\tilde{S}^1$ is at least as strong as S^1 since (4.2) is a special case of (4.14). Furthermore, it's strictly stronger, since (4.14) cuts off the fractional point in $LP(S^1)$ given in the proof of Proposition 4.2.2: take $\tilde{K} = \{k_1, k_2, k_3\}$ and $\Delta_i(k_r) = \{a_r\}$ for each $r = 1, 2, 3$.

Actually, the strengthened path-only formulation is now as strong as the path-tree and tree-only formulations. In proving this, it is helpful to observe that if $(\tilde{K}, \Delta_i)$ is a commodity-out-i partition pair for some $i \in N$, then, since $\{\Delta_i(k) : \ k \in \tilde{K}\}$ is a partition

of $\delta^+(i)$, we have the identity

$$\sum_{k\in\tilde{K}}\sum_{a\in\Delta_i(k)}\gamma_a = \sum_{a\in\delta^+(i)}\gamma_a, \tag{4.15}$$

for any vector $\gamma \in \mathbb{R}^{|A|}$.

Proposition 4.2.3. $LP(\tilde{S}^1) = Proj_x LP(S^2)$.

Proof. We first show that $Proj_x LP(S^2) \subseteq LP(\tilde{S}^1)$. Suppose $(x, y) \in LP(S^2)$. It suffices to show that Constraint (4.14) is satisfied. Let $i \in N$ and $(\tilde{K}, \Delta_i)$ be a commodity-out-i partition pair, chosen arbitrarily. In view of (4.4), the above identity (4.15), and (4.5), we have

$$\sum_{k\in\tilde{K}}\sum_{a\in\Delta_i(k)} x_{ak} \leq \sum_{k\in\tilde{K}}\sum_{a\in\Delta_i(k)} y_a = \sum_{a\in\delta^+(i)} y_a \leq 1.$$

Thus, $x \in LP(\tilde{S}^1)$ and $Proj_x LP(S^2) \subseteq LP(\tilde{S}^1)$.

Now suppose $x \in LP(\tilde{S}^1)$. We will prove by construction that $\exists y$ such that $(x, y) \in LP(S^2)$. For each $a \in A$, set $y_a := \max_{k\in K(d)} x_{ak}$. Clearly, by definition, the pair (x, y) must satisfy (4.4). Also, for each $a \in A$, choose $k^*_a \in \arg\max_{k\in K(d)} x_{ak}$. (Break ties arbitrarily; in particular, if the total flow on an arc is zero, any commodity may be selected.) Thus $y_a = x_{ak^*_a}$ for all $a \in A$. Let $i \in N$ and define

$$\tilde{K} := \{k^*_a : a \in \delta^+(i)\} \quad \text{and} \quad \Delta_i(k) = \{a \in \delta^+(i) \ : \ k^*_a = k\},$$

for each $k \in \tilde{K}$. Clearly $(\tilde{K}, \Delta_i)$ is commodity-out-i partition pair and since x satisfies (4.14), it must be that

$$1 \geq \sum_{k\in\tilde{K}}\sum_{a\in\Delta_i(k)} x_{ak} = \sum_{k\in\tilde{K}}\sum_{a\in\Delta_i(k)} x_{ak^*_a} = \sum_{k\in\tilde{K}}\sum_{a\in\Delta_i(k)} y_a = \sum_{a\in\delta^+(i)} y_a,$$

by the identity (4.15). Thus y satisfies (4.5), and the result follows. ■

Though the addition of Constraint (4.14) does improve the strength of the formulation S^1, it also increases its size significantly. Let $S(n,k)$ be the number of ways to partition a set of size n into k subsets. We have $S(n,k) = k*S(n-1,k)+S(n-1,k-1)$ and $S(n,1) = 1$ for all n. Then the number of commodity-out-i partition pairs is $\sum_{j=0}^{\min\{n,p\}} \frac{n!}{(n-j)!} S(n,j)$, where $n = |\delta^+(i)|$ and $p = |K(d)|$. This number grows exponentially with n and p.

We conclude this section by observing that the LP relaxation of $\tilde{S}^1$ is stronger than that of S^1 if and only if the number of commodities and out-degrees of nodes is larger than 2 and 3 respectively.

Proposition 4.2.4. *If $|K(d)| \leq 2$ and $|\delta^+(i)| \leq 3$ for all $i \in N$ then $LP(\hat{S}^1) = LP(S^1)$. Otherwise, $LP(\hat{S}^1) \neq LP(S^1)$.*

Proof. Note the problems of interest always have more than one commodity, that is $|K(d)| \geq 2$. We consider the following three cases:

- *Case 1: $|K(d)| \leq 2$ and $|\delta^+(i)| \leq 3$ for all $i \in N$.* In this case, any commodity-out-i partition pair, $(\tilde{K}, \Delta_i)$ has $|\tilde{K}| = 2$, and any partition of $\delta^+(i)$ into two non-empty sets must have a set of cardinality one. In other words, $\tilde{K} = \{k, k'\}$ for some $k \neq k'$ with $|\Delta_i(k)| = 1$. Thus, $\Delta_i(k') = \{a' \in \delta^+(i) \ : \ a' \neq a\}$, where $\Delta_i(k) = \{a\}$. Constraint (4.2) for this i, a, k and k' is precisely Constraint (4.14) for this commodity-out-i partition pair. In this case, $LP(\hat{S}^1) = LP(S^1)$.

- *Case 2: $|K(d)| \geq 3$.* The instance used in the proof of Proposition 4.2.2 has $|K(d)| = 3$ and we have shown $\exists x \in LP(S^1) \setminus Proj_x LP(S^2)$. By Proposition 4.2.3, $x \in LP(S^1) \setminus LP(\hat{S}^1)$. Thus, for this instance, $LP(\hat{S}^1) \neq LP(S^1)$.

- *Case 3: $|\delta^+(i)| \geq 4$ for some $i \in N$.* Consider an instance with two commodities, k_1, k_2, each having a different origin, but the same destination d. Figure 4.2 contains a snapshot of the network at some node i that is neither an origin nor the destination for the two commodities.

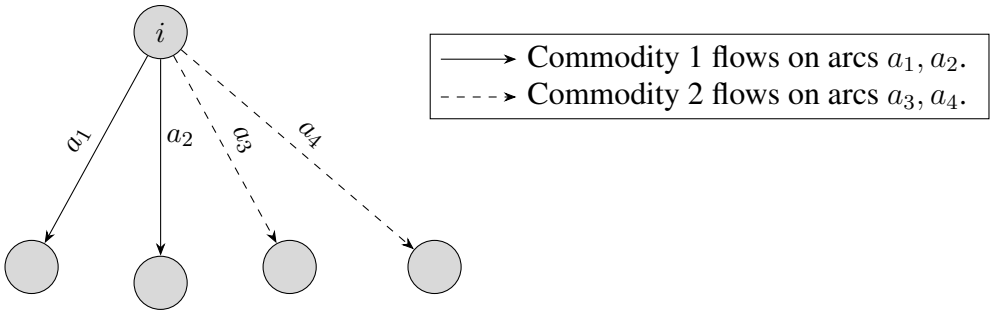


Figure 4.2: Four outgoing arcs at node i

Consider $x \in LP(S^1)$ for this instance with $x_{a_1k_1} = \frac{1}{3}$, $x_{a_2k_1} = \frac{1}{3}$, $x_{a_3k_2} = \frac{1}{3}$, and $x_{a_4k_2} = \frac{1}{3}$. One can easily check that these values will satisfy Constraint (4.2). However, these x_{ak} values and Constraint (4.4) imply $y_{a_1} \geq \frac{1}{3}$, $y_{a_2} \geq \frac{1}{3}$, $y_{a_3} \geq \frac{1}{3}$, and $y_{a_4} \geq \frac{1}{3}$. Thus, Constraint (4.5) is violated since $\sum_{a \in \delta^+(i)} y_a \geq \frac{4}{3} > 1$. Again, this implies, by Proposition 4.2.3, that $LP(\hat{S}^1) = Proj_x LP(S^2) \subsetneqq LP(S^1)$. ■

4.2.2 For the SNDPITC Formulations

Table 4.1 summarizes the size comparison for all the three SNDPITC formulations, in terms of the number of variables and constraints. Note that it is generally the case that $|N| << |K|$. Thus, F3 is significantly more compact that F1 and F2.

Table 4.1: Formulation size comparison.

Formulation	No. Variables	No. Constraints
F1	$\mathcal{O}(\|K\|\|A\|)$	$\mathcal{O}(\|K\|^2\|A\|)$
F2	$\mathcal{O}(\|K\|\|A\|)$	$\mathcal{O}(\|K\|\|A\|)$
F3	$\mathcal{O}(\|N\|\|A\|)$	$\mathcal{O}(\|N\|\|A\|)$

We consider the LP relaxations of F1, F2, and F3, where we relax the integrality constraints for variables x_{ak}, y_{ad}, and n_a, where applicable. The analysis in Section 4.2.1 leads us to conclude that not much is gained by considering F1, even with strengthening; the

most interesting trade-off is between the strength of F2 and the size of F3. The following proposition establishes a comparison of the LP relaxations of the formulations presented in Section 4.1.3:

Proposition 4.2.5. *F2 is the strongest of these formulations.*

Proof. Note that $Proj_{x,n}LP(F2) \subsetneqq LP(F1)$ follows easily from Proposition 4.2.2. Thus, the remainder of this proof shows that $Proj_{y,n}LP(F2) \subsetneqq Proj_{y,n}LP(F3)$.

Suppose $(x, y, n) \in LP(F2)$. We want to show that there exists w such that $(w, y, n) \in LP(F3)$. We can construct w variables by

$$w_{ad} := \sum_{k \in K(d)} q_k x_{ak}, \tag{4.16}$$

for each arc $a \in A$ and destination $d \in D$. We now consider the constraints of F3 in turn.

- Constraint (4.10): for each $d \in D$ and $i \in N$, in view of the flow balance constraint (4.1) for $x^d \in S^1(d)$, we have

$$\begin{aligned}
\sum_{a \in \delta^+(i)} w_{ad} - \sum_{a \in \delta^-(i)} w_{ad} &= \sum_{a \in \delta^+(i)} \sum_{k \in K(d)} q_k x_{ak} - \sum_{a \in \delta^-(i)} \sum_{k \in K(d)} q_k x_{ak} \\
&= \sum_{k \in K(d)} \sum_{a \in \delta^+(i)} q_k x_{ak} - \sum_{k \in K(d)} \sum_{a \in \delta^-(i)} q_k x_{ak} \\
&= \sum_{k \in K(d)} \left[q_k \left(\sum_{a \in \delta^+(i)} x_{ak} - \sum_{a \in \delta^-(i)} x_{ak} \right) \right] \\
&= \begin{cases} \sum\limits_{k \in K(d)} q_k, & \text{if } i = o(k), \\ -\sum\limits_{k \in K(d)} q_k, & \text{if } i = d, \\ 0, & \text{otherwise.} \end{cases}
\end{aligned}$$

- Constraint (4.11): for each $a \in A$ and $d \in D$, due to (4.4) for $(x^d, y^d) \in S^2(d)$, we

have

$$w_{ad} = \sum_{k \in K(d)} q_k x_{ak} \leq \sum_{k \in K(d)} q_k y_{ad}.$$

- Constraint (4.12): for each $a \in A$, in view of (4.8), we have

$$\sum_{d \in D} w_{ad} = \sum_{d \in D} \sum_{k \in K(d)} q_k x_{ak} = \sum_{k \in K} q_k x_{ak} \leq n_a.$$

Thus, $Proj_{y,n} LP(F2) \subseteq Proj_{y,n} LP(F3)$.

The strictness can be shown by the following instance, where there are two commodities, k_1, k_2, each having a different origin, but the same destination d. Figure 4.3 displays the network where nodes o_1 and o_2 are the origins of k_1 and k_2, respectively, node i is some intermediate node, and node d is the destination of both k_1 and k_2.

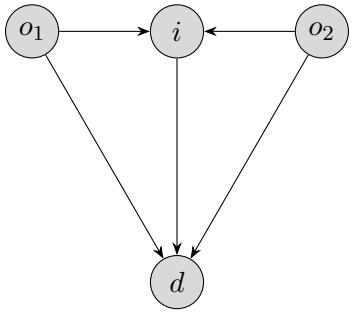


Figure 4.3: Network for the instance with a single destination, d, and two commodities, k_1 and k_2, having $o(k_1) = o_1$, $q(k_1) = \frac{1}{10}$, $o(k_2) = o_2$ and $q(k_2) = \frac{1}{2}$.

Consider the following feasible solution for the LP relaxation of F3, on this instance:

$$w_{(1,i),d} = \frac{1}{40}, \quad w_{(1,d),d} = \frac{3}{40}, \quad w_{(2,i),d} = \frac{3}{8}, \quad w_{(2,d),d} = \frac{1}{8}, \quad w_{(i,d),d} = \frac{2}{5},$$
$$y_{(1,i),d} = \frac{1}{4}, \quad y_{(1,d),d} = \frac{3}{4}, \quad y_{(2,i),d} = \frac{3}{4}, \quad y_{(2,d),d} = \frac{1}{4}, \quad y_{(i,d),d} = \frac{2}{3}.$$

It is straightforward to verify the feasibility of this solution. The computation is shown in Section B.2.

Given the above solution, it suffices to show that there does not exist a corresponding

feasible solution to F2. Suppose that there does exist x such that (x, y, n) is a feasible solution to F2. By Constraint (4.4), we have

$$x_{(2,d),k_2} \leq y_{(2,d),d} = \frac{1}{4} \qquad \text{and} \qquad x_{(i,d),k_2} \leq y_{(i,d),d} = \frac{2}{3}.$$

In view of the commodity flow balance constraint (4.1) at node d and $x_{(2,d),k_2} \leq \frac{1}{4}$, it must be that $x_{(i,d),k_2} \geq \frac{3}{4}$, which contradicts to the fact that $x_{(i,d),k_2} \leq \frac{2}{3}$. Thus, $Proj_{y,n}LP(F2) \subsetneqq Proj_{y,n}LP(F3)$. ■

4.3 Strengthening the Formulations

In this section, we present some simple valid inequalities that strengthen the aforementioned Steiner in-tree and SNDPITC formulations.

4.3.1 Strengthening the Steiner In-tree Formulations

In the absence of an objective function that puts a positive cost on using arcs in the Steiner in-tree, the tree arc indicator variable y may not induce a *minimal* tree: it may induce extraneous cycles. Although eliminating them is not needed for a valid SNDPITCformulation, doing so may help to solve these problems in practice.

The following class of inequalities, the so-called flow-balance inequalities, are discussed in both [47] and [53], which can strengthen the LP relaxation of S^2 and S^3:

$$y_a \leq \sum_{\hat{a} \in \delta^+(i)} y_{\hat{a}}, \qquad \forall i \in N \backslash \{d\},\ \forall a \in \delta^-(i). \tag{4.17}$$

This constraint eliminates extraneous subgraphs where a non-root node has a positive in-degree and a zero out-degree.

In the case of S^2, we propose to eliminate subgraphs, on which there are no commodity

flows, by adding the constraints

$$y_a \leq \sum_{k \in K(d)} x_{ak}, \qquad \forall a \in A. \tag{4.18}$$

It is easy to see for S^2, if (4.18) is satisfied then (4.17) must be satisfied. Actually, $\forall i \in N \setminus \{d\}$ and $a \in \delta^-(i)$, if $y_a = 0$, then (4.17) is satisfied. If $y_a = 1$, then $x_{ak} = 1$ for some $k \in K(d)$. According to (4.1), there exists $\hat{a} \in \delta^+(i)$ such that $x_{\hat{a}k} = 1$. In view of (4.4) and (4.6), we have $y_{\hat{a}} = 1$, and thus, (4.18) holds.

4.3.2 Strengthening the SNDPITC Formulations

Depending on the network structure, some arcs may not appear in any (simple) path for some commodities. Furthermore, in practical applications, it is likely that the LTL business would wish to forbid flow of some commodities on some arcs. The sets K^a may either be taken as given instance data, or determined at a preprocessing step, and can be used for coefficient reduction. In particular, they are helpful for strengthening Constraint (4.11) as follows:

$$w_{ad} \leq \left(\sum_{k \in K(a,d)} q_k \right) y_{ad}, \qquad \forall a \in A,\ \forall d \in D. \tag{4.19}$$

Furthermore, $\forall a \in A$, the variables x_{ak} can be removed for all $k \notin K^a$ and the variables y_{ad} and w_{ad} can be removed for any d with $K(a,d) = \emptyset$. The sets $\delta^+(i)$ and $\delta^-(i)$ can then be replaced by $\delta_d^{\pm}(i) = \{a \in \delta^{\pm}(i) :\ K(a,d) \neq \emptyset\}$, respectively, and the model constraints should be adjusted accordingly. In some cases, an arc *must* be used by a commodity: define $\underline{K}_a$ to be the set of commodities that must use arc a in their paths. Then for all $a \in A$ and $d \in D$ with $K(d) \cap \underline{K}_a \neq \emptyset$, y_{ad} can be fixed to 1.

As mentioned in the discussion of the Steiner in-tree formulations, we may use the

flow-balance inequalities (4.17) for each destination: the constraint

$$y_{ad} \leq \sum_{\hat{a} \in \delta_d^+(i)} y_{\hat{a}d}, \qquad \forall d \in D,\ \forall i \in N \backslash \{d\},\ \forall a \in \delta_d^-(i), \tag{4.20}$$

may be added to F2 and F3. We have observed that fractional solutions can violate a mirrored version of the flow-balance inequality:

$$y_{ad} \leq \sum_{\hat{a} \in \delta_d^-(i)} y_{\hat{a}d}, \qquad \forall d \in D,\ \forall i \in N \backslash O,\ \forall a \in \delta_d^+(i), \tag{4.21}$$

so adding it may also strengthen F2 and F3. Similarly, for each destination, (4.18) can be added to F1 and F2, which is

$$y_{ad} \leq \sum_{k \in K(a,d)} x_{ak}, \qquad \forall a \in A,\ \forall d \in D. \tag{4.22}$$

Also, for a given arc a, it may be that $\sum_{k \in K^a} q_k < 1$, in which case (4.8) is weaker than

$$n_a \geq x_{ak}, \qquad \forall a \in A,\ \forall k \in K^a, \tag{4.23}$$

and adding these to F1 and F2 improves the strength.

Since positive flow for destination d must leave every origin node in $O(d)$, the following special case of the d-cut constraints may strengthen F2 and F3:

$$\sum_{a \in \delta_d^+(i)} y_{ad} \geq 1, \qquad \forall d \in D,\ \forall i \in O(d). \tag{4.24}$$

These two formulations may also be strengthened by the addition of

$$n_a \geq y_{ad},\ \forall d \in D,\ \forall a \in A,\ K(a,d) \neq \emptyset. \tag{4.25}$$

Note that in the case of F2, due to Constraint (4.4) in each Steiner in-tree set $S^2(d)$, (4.25) dominates (4.23). Thus, (4.23) is no longer helpful when (4.25) is included for F2.

F3 can be further strengthened by forcing the relationship of variables w_{ad} and y_{ad} as follows:

$$w_{ad} \geq \begin{cases} q_k y_{ad}, & \text{if } a \in \delta_d^+(o(k)), \\ \left(\min_{k \in K(a,d)} q_k \right) y_{ad}, & \text{otherwise,} \end{cases} \qquad \forall d \in D,\ \forall a \in A,\ K(a,d) \neq \emptyset. \tag{4.26}$$

4.4 Novel Cutting Planes For the Flow-Based Formulation

All service network design formulations suffer from weakness of their LP relaxations. In particular, they allow weak lower bounds on the n_a variables. The well-known cut-set inequalities address this weakness by increasing the lower bound on a sum of such variables for arcs that cross a cut in the network. Theoretical strength of the cut-set inequalities has been shown in [12, 67] and an extensive computational study demonstrating their potential can be found in [66]. However, the separation is *NP-hard* [see 26] even for the single commodity-single facility version of the problem with a single source and a single sink. As a result, a practical implementation of the separation usually considers a relatively restricted version, limiting the size of cut sets as well as the choice of subsets of commodities, which may fail to cut off a fractional solution. To address this issue, we provide six alternative classes of cutting planes that supplement the cut-set inequalities. They are specially designed for the flow-based formulation, which is intriguing due to its compact size. For comparison, we use the limited version of cut-set inequalities defined as follows:

$$\sum_{a \in A(C,\bar{C})} n_a \geq \left\lceil \sum_{k \in K_C} q_k \right\rceil, \quad \text{for any nonempty cut set } C \subsetneqq N. \tag{4.27}$$

Throughout the remainder of the chapter, let $(\tilde{w}, \tilde{y}, \tilde{n})$ denote the optimal solution to the LP relaxation.

4.4.1 Wheat-Stalk Inequalities

This class of valid inequalities drive up the lower bound of the n_a variable on a single arc a, by inference on subtrees that use it. For a given arc $\hat{a} = (\hat{i}, \hat{j}) \in A$, the new inequality is defined for

- a subset of destinations $H \subseteq \{d \in D : \ K(\hat{a}, d) \neq \emptyset\}$,
- a collection of nonempty commodity subsets $\kappa_d \subseteq K(\hat{a}, d)$ for each $d \in H$, and
- a collection of in-trees, $\tau_d \subseteq A$, one for each $d \in H$, having terminals $\{o(k) : \ k \in \kappa_d\}$ and root $\hat{j}$, such that $\tau_d \cap \delta^-(\hat{j}) = \{\hat{a}\}$.

Another way of describing τ_d is as the union over all $k \in \kappa_d$ of the arcs in a path starting from $o(k)$ and ending with the arc $\hat{a}$. The *wheat-stalk inequalities* (see Figure 4.4 for an illustration) are defined as follows:

$$n_{\hat{a}} \geq \left\lceil \sum_{d \in H} \sum_{k \in \kappa_d} q_k \right\rceil \left(1 + \sum_{d \in H} \left(\sum_{a \in \tau_d} y_{ad} - |\tau_d| \right) \right). \tag{4.28}$$

The validity of (4.28) is straightforward to see: if all the in-trees in the collection are used, then all commodities in the collection of commodity subsets must flow on arc $\hat{a}$; if any arc in the collection of in-trees is no in use, the right hand side will take value 0, given the integrality of y_{ad}. Thus, the inequality holds.

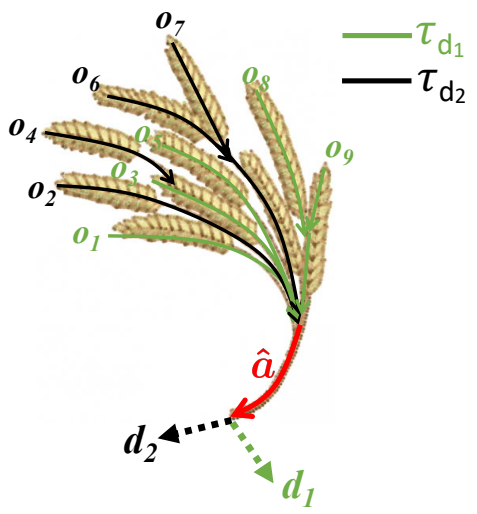


Figure 4.4: An illustrative example of the wheat-stalk analogue.

In cases where there is a nonzero lower bound on the number of trucks needed on arc $\hat{a}$, arising when the set $\underline{K}^{\hat{a}}$ of commodities that must use arc $\hat{a}$ is non-empty, then the valid inequality can be strengthened. In this case, let $L_{\hat{a}}$ denote the lower bound on the number of trucks needed on arc $\hat{a}$, so

$$L_{\hat{a}} = \left\lceil \sum_{k \in \underline{K}^{\hat{a}}} q_k \right\rceil .$$

Now there is no need or benefit to include $k \in \kappa_d$ for any $k \in \underline{K}^{\hat{a}}$, so we modify the criteria for selection of the destination subset and the commodity subsets to require

- a subset of destinations $H \subseteq \{d \in D : \ K(\hat{a}, d) \setminus \underline{K}^{\hat{a}} \neq \emptyset\}$, and
- a collection of nonempty commodity subsets $\kappa_d \subseteq K(\hat{a}, d) \setminus \underline{K}^{\hat{a}}$ for each $d \in H$.

Then the inequality

$$n_{\hat{a}} \geq L_{\hat{a}} + \left(\left\lceil \sum_{d \in H} \sum_{k \in \kappa_d} q_k + \sum_{k \in \underline{K}^{\hat{a}}} q_k \right\rceil - L_{\hat{a}} \right) \left(1 + \sum_{d \in H} \left(\sum_{a \in \tau_d} y_{ad} - |\tau_d| \right) \right), \qquad (4.29)$$

is valid. (4.29) is stronger than (4.28) since

$$1 + \sum_{d \in H} \left(\sum_{a \in \tau_d} y_{ad} - |\tau_d| \right) \leq 1.$$

An illustrative instance is included in the section B.3, which shows the effect of the simple strengthening inequalities, the cut-set inequalities as well as the wheat-stalk inequalities. It turns out that the wheat-stalk inequalities are strong, actually facet defining, for this particular instance.

Separation

We consider a simplified version of (4.28), where $\tau_d = \{\hat{a}\}$ is a single arc, to get a sense of the difficulty of separation. In this case, the tail of the arc $\hat{a}$, $\hat{i}$, is an origin, and the inequality takes the following form:

$$n_{\hat{a}} \geq \left\lceil \sum_{d \in H} q_d \right\rceil \left(1 + \sum_{d \in H} (y_{\hat{a}d} - 1) \right). \tag{4.30}$$

To check if (4.30) is satisfied or not for a fixed arc $\hat{a}$, we need to decide if there exists a the set of destination $H \subseteq D$ such that the right hand side of (4.30) is larger than $\tilde{n}_{\hat{a}}$. Equivalently, the optimization version of it can be stated as follows:

$$\max_{H \subseteq D} \left\lceil \sum_{d \in H} q_d \right\rceil \left(1 - \sum_{d \in H} p_d \right), \tag{4.31}$$

where $q_d \in (0, 1]$ and $p_d := 1 - \tilde{y}_{\hat{a}d} \in [0, 1]$, for $d \in D$. Actually, it suffices to just consider those $p_d \in (0, 1)$, since if for some $d \in D$, $p_d = 0$, then we should always include it in the set H to makes the objective larger, if $p_d = 1$, then we never want to include it.

Proposition 4.4.1. *The optimization problem* (4.31) *is NP-hard.*

Proof. We prove it by reducing from the PARTITION, a decision problem asking to decide if there exists $\bar{S} \subseteq S$ such that $\sum_{k \in \bar{S}} a_k = b$ for a given a set of positive integers a_k with

$\sum_{k\in S} a_k = 2b$, and $a_k < b$ for all $k \in S$. Observe that the answer to the PARTITION is "YES" if and only if

$$\begin{aligned}
&\max_{\bar{S}\subseteq S} \left(\sum_{k\in\bar{S}} a_k\right)\left(2b - \sum_{k\in\bar{S}} a_k\right) = b^2 \\
\Longleftrightarrow &\max_{\bar{S}\subseteq S} \left(\sum_{k\in\bar{S}} \frac{a_k}{b}\right)\left(2 - \sum_{k\in\bar{S}} \frac{a_k}{b}\right) = 1 \\
\Longleftrightarrow &\max_{\bar{S}\subseteq S} \left[\frac{1}{b^{|S|}} + \sum_{k\in\bar{S}} \frac{a_k}{b}\right]\left(2 - \frac{1}{b^{|S|}} - \sum_{k\in\bar{S}} \frac{a_k}{b}\right) = 2\left(1 - \frac{1}{b^{|S|}}\right).
\end{aligned}$$

The first equivalence is straightforward by dividing both sides by b^2, and the second is because

$$\max_{\bar{S}\subseteq S} \left[\frac{1}{b^{|S|}} + \sum_{k\in\bar{S}} \frac{a_k}{b}\right]\left(2 - \frac{1}{b^{|S|}} - \sum_{k\in\bar{S}} \frac{a_k}{b}\right) \geq 2\left(1 - \frac{1}{b^{|S|}}\right)$$

if and only if $\exists \bar{S} \subseteq S$ such that either

$$\sum_{k\in\bar{S}} \frac{a_k}{b} \in \left(0, \frac{1}{b^{|S|}}\right] \quad \text{or} \quad \sum_{k\in\bar{S}} \frac{a_k}{b} \in \left(1 - \frac{1}{b^{|S|}}, 1\right].$$

Since a_k and b are positive integers for $k \in S$, the only possibility is $\sum_{k\in\bar{S}} \frac{a_k}{b} = 1$. Let $D = S \cup \{d'\}$,

$$q_d = \begin{cases} \dfrac{a_d}{b}, & d \in S, \\ \dfrac{1}{b^{|S|}}, & d = d', \end{cases} \qquad \text{and } \; p_d = q_d, \; \forall d \in D.$$

Obviously, $q_b, p_b \in (0, 1)$ for all $d \in D$. The size of the PARTITION is $|S|\log(N)$, with $N = \max\{b, \; a_d : \; d \in D\}$. The size of the reduced optimization problem (4.31) is a polynomial of $|S|\log(N)$ since $\log\left(b^{|S|}\right) = |S|\log(b) \leq |S|\log(N)$, which completes the proof. ■

Actually, the arguments above only show that (4.31) weakly NP-hard. Although we are not able to prove its strong NP-hardness directly, we have results for the following two

closely related problems:

$$\max_{H \subseteq D} \left(\sum_{d \in H} q_d \right) \left(1 - \sum_{d \in H} p_d \right), \quad \text{and} \tag{4.32}$$

$$\max_{H \subseteq D} \left\lfloor \sum_{d \in H} q_d \right\rfloor \left(1 - \sum_{d \in H} p_d \right). \tag{4.33}$$

Proposition 4.4.2. *Optimization problems* (4.32) *and* (4.33) *are both strongly NP-hard.*

Proof. The proof reduces PARTITION with rational weights (PARTITION-RW), which has been proved to be strongly NP-complete in [72], to (4.32) and (4.33). The PARTITION-RW asks to dicide if there exists $S_1 \subseteq S$ such that $\sum_{k \in S_1} \frac{r_k}{w_k} = \frac{r}{w}$ for a given a set of positive rational numbers $\frac{r_k}{w_k}$ with $\sum_{k \in S} \frac{r_k}{w_k} = \frac{2r}{w}$, and r_k, w_k, r, w being positive integers, and $\frac{r_k}{w_k} < \frac{r}{w}$ for all $k \in S$. Similarly, the answer to the PARTITION-RW is "YES" if and only if

$$\begin{aligned}
&\max_{\bar{S} \subseteq S} \left(\sum_{k \in \bar{S}} \frac{r_k}{w_k} \right) \left(\frac{2r}{w} - \sum_{k \in \bar{S}} \frac{r_k}{w_k} \right) = \frac{r^2}{w^2} \\
\iff &\max_{\bar{S} \subseteq S} \left(\sum_{k \in \bar{S}} \frac{wr_k}{rw_k} \right) \left(2 - \sum_{k \in \bar{S}} \frac{wr_k}{rw_k} \right) = 1 \\
\iff &\max_{\bar{S} \subseteq S} \left\lfloor \sum_{k \in \bar{S}} \frac{wr_k}{rw_k} \right\rfloor \left(2 - \sum_{k \in \bar{S}} \frac{wr_k}{rw_k} \right) = 1.
\end{aligned}$$

The first equivalence is obtained by dividing both sides by $\frac{r}{w}$, and the second can be seen by considering the folowing cases:

$$\left\lfloor \sum_{k \in \bar{S}} \frac{wr_k}{rw_k} \right\rfloor \left(2 - \sum_{k \in \bar{S}} \frac{wr_k}{rw_k} \right) \begin{cases} = 0 & \text{if } \sum_{k \in \bar{S}} \frac{wr_k}{rw_k} \in (0,1), \\ = 1 & \text{if } \sum_{k \in \bar{S}} \frac{wr_k}{rw_k} = 1, \\ \in (0,1) & \text{if } \sum_{k \in \bar{S}} \frac{wr_k}{rw_k} \in (1,2), \\ = 0 & \text{if } \sum_{k \in \bar{S}} \frac{wr_k}{rw_k} = 2. \end{cases}$$

Take $D = S$ and $p_d = q_d = \frac{wr_d}{rw_{ad}} \in (0, 1)$, we have two optimization problems, one in the form of (4.32), the other of (4.33), whose sizes grow as a polynomial function of that of the original PARTITION-RW in unary, which completes the proof. ■

The above hardness results suggest that the separation is challenging and we propose the following heuristic when $\tau_d = \hat{a}$, is a single arc from an origin i. For each $i \in O$, let $K_i := \{k \in K : \ o(k) = i\}$, $H_i := \{d(k) : \ k \in K_i\}$, $\tilde{\delta}_i = \left\{a \in \delta^+(i) : \ \sum_{d \in H_i} \tilde{y}_{ad} > 0\right\}$, $D_j \subseteq D$ such that $\forall d \in D_j$, there is a directed path from j to d, $k(i, d)$ be the commodity with origin i and destination d, and $D_{ij} := \{d \in D_j : \ k(i, d) \text{ exists}\}$.

At each iteration, the heuristic first decides an origin, which is the tail of the arc of interest, in a greedy way. It picks and removes the origin i from the remaining nodes in O, from which the number of commodities originating, i.e., $|K_i|$, is the largest. The reason for this is that we will have more choice and the possibility of separating the current fractional solution may be larger. Then the arc $\hat{a} \in \tilde{\delta}_i$ is also decided greedily, where one with a larger $\sum_{d \in H_i} q_{k(i,d)} \tilde{y}_{\hat{a}d}$ is picked first. The intuition is that it may provide a larger right hand side of (4.28). After fixing $\hat{a}$, more destinations are added to form the set H. This step is also done greedily such that $d \in D_{ij}$ with the largest $\tilde{y}_{ad}$ will be added first, and it stops until we find a violated inequality or conclude we can not find one on this arc. The full algorithm is summarized in Algorithm 2.

4.4.2 Wheat-Sheaf Inequalities

Similar to wheat-stalk inequalities, this class of inequalities also depend on inferring the subtrees in use. Instead of considering the truckloads on a single arc, the wheat-sheaf inequalities include all outbound arcs from a node. For a given node i, a subset of destinations $H \subseteq D \setminus \{i\}$, a collection of subsets of commodities $\kappa_d \subseteq K(d)$ and a collection of subtrees τ_d that connects the origin of each commodity in κ_d to i, for each $d \in H$, the

Algorithm 2: Heuristic for separating wheat-stalk inequalities of cardinality one.

Initialization: $O = \{O(d) : d \in D\}$.
while $O \neq \emptyset$ **do**
  Select and remove a node i with $|K_i|$ largest from O.
  while $\tilde{\delta}_i \neq \emptyset$ **do**
    Select and remove an arc $\hat{a} = (i, j) \in \tilde{\delta}_i$ with the largest $\sum_{d \in H_i} q_{k(i,d)} \tilde{y}_{\hat{a}d}$.
    Compute D_{ij} and sort $q_{k(i,d)} \tilde{y}_{\hat{a}d}$ for $d \in D_{ij}$ in descending order $q_{k(i,d_1)} \tilde{y}_{\hat{a}d_1}, \ldots, q_{k(i,d_{|D_{ij}|})} \tilde{y}_{\hat{a}d_{|D_{ij}|}}$.
    for *h = 1 to* $|D_{ij}|$ **do**
      if $\sum_{p=1}^{h} \tilde{y}_{\hat{a}d_p} > h - 1$ **then**
        if $\tilde{n}_{\hat{a}} < \left\lceil \sum_{p=1}^{h} q_{k(i,d_p)} \right\rceil \left(\sum_{p=1}^{h} \tilde{y}_{\hat{a}d_p} - h + 1 \right)$ **then**
          $n_{\hat{a}} \geq \left\lceil \sum_{p=1}^{h} q_{k(i,d_p)} \right\rceil \left(\sum_{p=1}^{h} y_{\hat{a}d_p} - h + 1 \right)$ is a violated wheat-stalk inequality, **stop**.
      else
        Break from the for loop.

wheat sheaf inequalities (see Figure 4.4 for an illustration) take the following form:

$$\sum_{a \in \delta^+(i)} n_a \geq \left\lceil \sum_{d \in H} \sum_{k \in \kappa_d} q_k + \sum_{k : o(k) = i} q_k \right\rceil \left(\sum_{d \in H} \left(\sum_{a \in \tau_d} y_{ad} - |\tau_d| \right) + 1 \right). \tag{4.34}$$

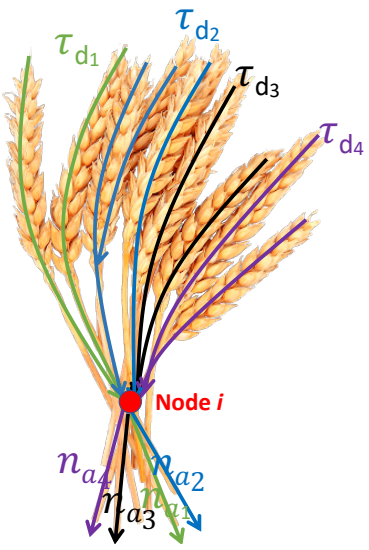


Figure 4.5: An illustrative example of the wheat-sheaf analogue.

Separation

The difficulty of separation can be seen from that of wheat-stalk inequalities since the right hand side of (4.34) is exactly the same as that of (4.28) for any $i \notin O$. A similar separation heuristic as follows can be used to find a violated wheat-sheaf inequality. For each node i, it identifies the set of origin nodes O_i having an arc to i. At each iteration, the algorithm chooses and removes the node i with the largest $|O_i|$ from N. Then it chooses and removes the node u with the largest $\sum_{d \in H_u} q_{k(u,d)} \tilde{y}_{ad}$ from O_i, and the arc of interest now is $a = (u, i)$. The intuition for doing this is similar to that of Algorithm 2 and then the heuristic also decides the destinations to include in the same way. The difference is that more than one arcs can be included in this case, which means $|\tau_d|$ may not be 1 for the right hand side of (4.34), but all arcs in τ_d are chosen from $\delta^-(i)$. The full description of the heuristic is shown in Algorithm 3.

Algorithm 3: Heuristic for separating the wheat-sheaf inequalities.

Initialization: $O_i = \{u \in O : (u, i) \in A\}$ for all $i \in N$.
while $N \neq \emptyset$ **do**
  Select and remove a node i with $|O_i|$ largest from N.
  Set $q = s = t = 0$, $\mathcal{Y} = \emptyset$.
  while $O_i \neq \emptyset$ **do**
    Select and remove a node $u \in O_i$ with $\sum_{d \in H_u} q_{k(u,d)} \tilde{y}_{ad}$ largest, and set $a = (u, i)$.
    Sort $q_{k(i,d)} \tilde{y}_{ad}$ for $d \in H_u$ in descending order $q_{k(u,d_1)} \tilde{y}_{ad_1}, \ldots, q_{k(u,d_{|H_u|})} \tilde{y}_{ad_{|H_u|}}$.
    for *h = 1 to* $|H_u|$ **do**
      Set $q = q + q_{k(u,d_h)}$, $s = s + \tilde{y}_{ad_h}$, $t = t + 1$, $\mathcal{Y} = \mathcal{Y} \cup \{(a, d_h)\}$.
      if $s > t - 1$ **then**
        if $\tilde{n}_a < \lceil q \rceil (s - t + 1)$ **then**
          $n_a \geq \lceil q \rceil \left(\sum_{(a,d) \in \mathcal{Y}} y_{ad} - t + 1 \right)$ is a violated wheat-stalk inequality, **stop**.
      else
        Break from the while loop.

4.4.3 Combinatorial Rounding Inequalities

A critical relationship in this problem is that captured in constraints (4.8), which, together with the integrality of the truck use variables, ensures that the number of trucks assigned to an arc is an integer no less than the total quantities of commodities flowing on that arc. These constraints, used in both F1 and F2, have the structure of a binary knapsack constraint with a variable integer right-hand side. Thus, we expect current state-of-the-art solvers to have good built-in capability to find valid inequalities that will strengthen this relationship in the LP relaxation.

With the omission of the binary commodity flow variables, Formulation F3 only indirectly models this relationship, via the mixed integer constraint (4.12). The requirement that the truck use variable is an integer at least as big as a combination of commodity quantities is not explicit forced in this formulation. We discuss how to rectify this, by considering a class of cuts that we call *combinatorial rounding inequalities*.

Consider the following structure in F3 for each arc $a \in A$ (For simplicity, we drop the arc subscript for variables w and n in describing the form of the inequalities and how to separate them.):

$$S = \left\{ (w,n) : w_d = \sum_{k \in K(a,d)} q_k z_k, \forall d \in D,\ \exists z \in \{0,1\}^{|K^a|} \cap \mathcal{Z},\ n \geq \sum_d w_d, n \in \mathbb{Z} \right\}.$$

Here $\mathcal{Z}$ may capture any other constraints known on z, for example, if $k, k' \in K(a,d)$ with $k \neq k'$ and $o(k)$ is the tail of the arc, then the in-tree structure requires that $z_{k'} \leq z_k$.

The combinatorial rounding inequalities applied to S are defined to be

$$n \geq \beta + \sum_d \alpha_d w_d, \tag{4.35}$$

where $\alpha_d \geq 0$, for all $d \in D$, and $\beta \in \mathbb{R}$ is upper bounded by $\min_{(w,n)\in S} \{n - \sum_d \alpha_d w_d\}$.

Proposition 4.4.3. *All nontrivial facet defining inequalities of S take the form of* (4.35).

By nontrivial we mean the inequalities are not of the form $w_d \geq \gamma_d$*, where* $\gamma_d = \min_{(w,n)\in S} w_d$*, for* $d \in D$*.*

Proof. Let $F_{\alpha,\beta} = \{(w,n) : n \geq \beta + \sum_d \alpha_d w_d\} \cap S$ be a nontrivial facet. Since for $\hat{d} \in D$, $F_{\alpha,\beta}$ is not contained in the face defined by $w_{\hat{d}} \geq \gamma_{\hat{d}}$, so $\exists(\bar{w},\bar{n})$ that belongs to $F_{\alpha,\beta}$ such that

$$\bar{n} = \beta + \sum_d \alpha_d \bar{w}_d \tag{4.36}$$

with $\bar{w}_{\hat{d}} > \gamma_{\hat{d}}$. Let $\hat{w}_d = \bar{w}_{ad}$ for all $d \in D \setminus \{\hat{d}\}$, and $\hat{w}_{\hat{d}} = \gamma_{\hat{d}}$, then $(\hat{w},\bar{n}) \in S$. Since $n \geq \beta + \sum_d \alpha_d w_d$ is valid for all S, we have

$$\bar{n} \geq \beta + \sum_d \alpha_d \hat{w}_d = \sum_{d\in D\setminus\{\hat{d}\}} \alpha_d \bar{w}_d + \gamma_{\hat{d}} \tag{4.37}$$

Combine (4.36) and (4.37), we have $\alpha_{\hat{d}}\bar{w}_{\hat{d}} \geq \gamma_{\hat{d}} \Rightarrow \alpha_{\hat{d}} \geq 0$, which completes the proof. ■

Separation

Since the relationship between such valid inequalities and the well-known MIR cuts is readily apparent, we do not expect to find a closed form for the coefficients $\alpha_a = (\alpha_d)_{d\in D}$ and β, and nor do we expect separation to be easy.

Instead, we propose the following separation procedure to identify (α,β) cutting off given $(\tilde{w},\tilde{n})$ or show that none exists. The procedure solves, at each iteration, an LP of the form

$$\max_{\alpha,\beta}\{\alpha^T\tilde{w} + \beta :\ \alpha^T w + \beta \leq n,\ \forall(w,n) \in M,\ \alpha \geq 0\},$$

where $M \subseteq S$. The set M is augmented until it assures that the optimal solution, (α^*,β^*), yields a valid inequality for S when $\alpha := \alpha^*, \beta = \beta^*$ is used in (4.35). In this case we say that (α^*,β^*) is *valid* for S. If at any stage, $(\alpha^*)^T\tilde{w} + \beta^* \leq \tilde{n}$ then the procedure may stop: there is no nontrivial valid facet defining inequality separating $(\tilde{w},\tilde{n})$ from S. Otherwise, we check the validity of (α^*,β^*), by solving an IP over S to identify a point in S that

(α^*, β^*) cuts off or proves that there is none: determine

$$\mu := \max_{(w,n)} \left\{ (\alpha^*)^T w + \beta^* - n : \ (w,n) \in S \right\}.$$

If $\mu > 0$, the solution (w^*, n^*) may be added to M; otherwise it must be that (α^*, β^*) is valid. To see that this procedure must terminate finitely, first observe that it suffices to include in M only points $(w,n) \in S$ that satisfy $n = \lceil \sum_d w_d \rceil$, which yields a finite set. Furthermore, it is clear that if $(w,n) \in S$ and $(w',n) \in S$ and w' dominates w, i.e., $w' \geq w$ (componentwise) and $w' \neq w$, then (w,n) cannot lie on a facet of $\mathrm{conv}(S)$. Thus, we can ignore $(w,n) \in S$, where w is dominated by w' for some $(w',n) \in S$, and we instead propose to solve, in practice, the hierarchical optimization problem

$$\mathrm{lex} \max_{(w,n) \in S} \left\{ (\alpha^*)^T w + \beta^* - n, \sum_d w_d \right\}.$$

We may also want to solve

$$\mathrm{lex} \max_{\alpha,\beta} \left\{ \alpha^T w + \beta, \ \sum_d \alpha_d + \beta : \ \alpha^T w + \beta \leq n, \ \forall (w,n) \in M, \ \alpha \geq 0 \right\},$$

to try to get a "stronger" inequality, but this may come at the cost of more iterations, so it is not clear if this is helpful on balance, or not.

To complete the description of the separation procedure, we need to explain how to initialize M to avoid the LP being unbounded. One approach is to first solve, for each d, the hierarchical IP $\mathrm{lex} \max_{(w,n) \in S} \{w_d, -n\}$, and include the result in M. Alternatively, M could simply be initialized to

$$M := \left\{ \left(\left(\sum_{k \in K(a,d)} q_k \right) e_d, \left\lceil \sum_{k \in K(a,d)} q_k \right\rceil \right) : \ \forall d \in D \right\}$$

where e_d denotes the d-th unit vector. Regardless of the structure of $\mathcal{Z}$, these points provide

a valid upper bound on α_d, for each $d \in D$.

4.4.4 Generalized Cut-Set Inequalities

Given a cut, the original cut-set inequalities focus on lower bounding the truck use variables n via the commodity quantities q as well as the flow variables w (excluded for the limited version considered in this paper). We can generalize them by incorporating the design variables y. Given $V \subsetneqq N$, for any $S \subseteq A(V, \bar{V})$ with $s = |S| \leq r$, where $A(V, \bar{V})$ is the set of arcs that going from set V to its compliment $\bar{V}$, i.e., arcs that lie in the cut set generated by V. Suppose $K_V = \{k_1, k_2, \cdots, k_r\}$ and $\mathcal{D}_V = \{d(k_i), 1 \leq i \leq r\}$, then a valid inequality, which we call a *generalized cut-set inequality*, is defined as follows:

$$\sum_{a \in A(V,\, \bar{V})} n_a \geq C_V^s - \left(s - \frac{1}{|\mathcal{D}_V|} \sum_{d \in \mathcal{D}_V} \sum_{a \in S} y_{ad} \right), \tag{4.38}$$

where C_V^s is defined to be the optimal value of the following optimization problem:

$$\begin{aligned}
C_V^s = \min \quad & \sum_{j=1}^{s} z_j & & \\
\text{s.t.} \quad & \sum_{j=1}^{s} x_{ij} = 1, & & \forall i \in [r], \\
& \sum_{i=1}^{r} x_{ij} \geq 1, & & \forall j \in [s], \\
& z_j \geq \sum_{i=1}^{r} x_{ij} q_{k_i}, & & \forall j \in [s], \\
& z_j \in \mathbb{Z}_+, x_{ij} \in \{0, 1\}, & & \forall i \in [r],\ j \in [s],
\end{aligned} \tag{4.39}$$

which computes a lower bound on the number of trucks needed to deliver all commodities in K_V when all arcs in S are used.

We can further strengthen (4.38) to

$$\sum_{a \in A(V,\, \bar{V})} n_a \geq C_V^s - \left(s - \sum_{a \in S} z_a \right), \tag{4.40}$$

where for $a \in A$, the variable z_a indicates whether the arc is used by any commodity or not. It suffice to use a continuous variable since all y variables are binary. To ensure this happens, we should also add the following constraints to the flow-based formulation:

$$\begin{aligned} z_a &\geq y_{ad}, && a \in A, d \in \mathcal{D}_V, \\ z_a &\leq \sum_{d \in \mathcal{D}_V} y_{ad}, && a \in A, \\ z_a &\in [0,1], && a \in A. \end{aligned}$$

(4.40) is stronger than (4.38) since

$$z_a \geq y_{ad} \Rightarrow \sum_{a \in S} z_a = \frac{1}{|\mathcal{D}_V|} \sum_{d \in \mathcal{D}_V} \sum_{a \in S} z_a \geq \frac{1}{|\mathcal{D}_V|} \sum_{d \in \mathcal{D}_V} \sum_{a \in S} y_{ad}.$$

To show the validity of (4.38), it suffice to show that (4.40) is valid. We first prove the following intermediate result.

Lemma 4.4.1.

$$C_V^{\ell+1} - 1 \leq C_V^{\ell} \leq C_V^{\ell+1}, \qquad \forall 1 \leq \ell \leq r-1, \tag{4.41}$$

where C_V^{ℓ} is defined as in (4.39)*.*

Proof. Let (x^ℓ, z^ℓ) be an optimal solution to (4.39) when $s = \ell$. Since $r \geq \ell + 1$ and by the constraints of the optimization problem (4.39), we have

$$r = \sum_{i=1}^{r} \sum_{j=1}^{\ell} x_{ij}^{\ell} = \sum_{j=1}^{\ell} \sum_{i=1}^{r} x_{ij}^{\ell} \geq \ell + 1.$$

According to the pigeonhole principle, there $\exists j' \in [\ell]$ such that $\sum_{i=1}^{r} x_{ij'}^{\ell} \geq 2$, meaning

that there are two commodities using a common arc in the cut set to go into $\bar{V}$. So we may move one of them to another arc to construct a feasible solution $(x^{\ell'}, z^{\ell'})$ when $\ell' = \ell + 1$. Suppose $x^{\ell}_{i'j'} = 1$, then $(x^{\ell'}, z^{\ell'})$ can be obtained as follows:

$$x^{\ell'}_{ij} = \begin{cases} 0, & \text{if } i = i', j = j', \\ 1, & \text{if } i = i', j = \ell', \\ 0, & \text{if } i \neq i', j = \ell', \\ x^{\ell}_{ij}, & \text{otherwiser,} \end{cases} \qquad z^{\ell'}_{j} = \begin{cases} z^{\ell}_{j}, & \text{if } j \in [\ell] \setminus \{j'\}, \\ z^{\ell}_{j} - \lfloor q_{k_{i'}} \rfloor, & \text{if } j = j', \\ \lceil q_{k_{i'}} \rceil, & \text{if } j = \ell + 1. \end{cases}$$

Thus,

$$C^{\ell+1}_{V} = C^{\ell'}_{V} \leq \sum_{j \in [\ell] \setminus \{j'\}} z^{\ell}_{j} + z^{\ell}_{j'} - \lfloor q_{k_{i'}} \rfloor + \lceil q_{k_{i'}} \rceil \leq \sum_{j \in [\ell]} z^{\ell}_{j} + 1 = C^{\ell}_{V} + 1.$$

Similarly, suppose $2 \leq \ell \leq r$, $\ell' = \ell - 1$ and $x^{\ell}_{i'\ell} = 1$, we can construct a feasible solution $(x^{\ell'}, z^{\ell'})$ as follows:

$$x^{\ell'}_{ij} = \begin{cases} x^{\ell}_{ij} + x^{\ell}_{i\ell}, & \text{if } i \in [r], j = 1, \\ x^{\ell}_{ij}, & \text{if } i \in [r], j = [\ell - 1] \setminus \{1\}, \end{cases} \qquad z^{\ell'}_{j} = \begin{cases} z^{\ell}_{1} + z^{\ell}_{\ell}, & \text{if } j = 1, \\ z^{\ell}_{j}, & \text{if } j \in [\ell - 1] \setminus \{1\}, \end{cases}$$

Thus,

$$C^{\ell-1}_{V} = C^{\ell'}_{V} \leq \sum_{j \in [\ell-1] \setminus \{1\}} z^{\ell}_{j} + z^{\ell}_{1} + z^{\ell}_{\ell} \leq \sum_{j \in [\ell]} z^{\ell}_{j} = C^{\ell}_{V},$$

or equivalently, $C^{\ell}_{V} \leq C^{\ell+1}_{V}$ for $1 \leq \ell \leq r - 1$, which completes the proof. ■

Proposition 4.4.4. *For any $V \subsetneq N$ and $S \subseteq A(V, \bar{V})$ with $|S| \leq |\mathcal{D}_V|$,* (4.40) *is valid.*

Proof. By definition of C^{s}_{V}, we have

$$\sum_{a \in A(V, \bar{V})} n_a \geq C^{\ell}_{V}, \quad \text{with } \ell = \sum_{a \in A(V, \bar{V})} z_a. \tag{4.42}$$

In view of the right part of (4.4.1), we have

$$C_V^{\ell} \geq C_V^{\ell'}, \quad \text{with } \ell' = \sum_{a \in S} z_a \leq \ell. \tag{4.43}$$

According to the left part of (4.4.1), we have

$$C_V^{\ell'} + \left(s - \sum_{a \in S} z_a \right) \geq C_V^{s}. \tag{4.44}$$

Combining (4.42), (4.43), and (4.44) completes the proof. ■

Separation

To compute the constant C_V^s for a subset $V \subseteq N$ and a number s, we need to solve a small MIP, which can be difficult by itself. Thus, we don't expect the separation to be easy. For a practical separation heuristic, we only consider cut sets (by enumeration) induced by subsets of destinations, i.e., i is a destination of some commodity $\forall i \in V$. Note, in this case, (4.38) and (4.40) are equivalent, and thus, it suffices to use (4.38). For each such set V, let $t = \min\{|A(V, \bar{V})|, |K_V|\}$. For each $s = 1, \cdots, t$, we compute the constant C_V^s. By definition, C_V^1 can be computed trivially, which is simply $\lceil \sum_{i=1}^{r} q_{k_i} \rceil$. After that, for each subset $S \subseteq V$ such that $|S| = s$, we check if (4.38) is violated by $(\tilde{n}, \tilde{y})$. The full

description of the heuristic is shown below.

Algorithm 4: Heuristic for separating the generalized cut-set inequalities.

Initialization: $\mathcal{V} := \{V : V \subsetneqq D, V \neq \emptyset\}$.

while $\mathcal{V} \neq \emptyset$ **do**

Select and remove a set V from $\mathcal{V}$, and compute $t = \min\{|A(V, \bar{V})|, |K_V|\}$.

for *s = 1 to t* **do**

Compute the constant C_V^s, and identify $\mathcal{S} := \{S : S \subseteq A(V, \bar{V}), |S| = s\}$.

while $\mathcal{S} \neq \emptyset$ **do**

Select and remove a set S from $\mathcal{S}$.

if $\sum_{a \in A(V,\, \bar{V})} \tilde{n}_a < C_V^s - \left(s - \frac{1}{|\mathcal{D}_V|} \sum_{d \in \mathcal{D}_V} \sum_{a \in S} \tilde{y}_{ad}\right)$ **then**

$\sum_{a \in A(V,\, \bar{V})} n_a \geq C_V^s - \left(s - \frac{1}{|\mathcal{D}_V|} \sum_{d \in \mathcal{D}_V} \sum_{a \in S} y_{ad}\right)$ is a violated generalize cut-set inequality, **Stop**.

4.4.5 Commodity-Merging Inequalities

This class of inequalities also try to bound from below the truck use variables n via the commodity quantities q and the design variables y. Instead of looking at all arcs in a cut set, the focus here is for a single arc. In general, for an arc $a \in A$ and $H \subseteq D$, $q_{ad} := \min_{k \in K(a,d)} q_k$, if

$$\sum_{d \in H} \hat{q}_{ad} > |H| - 1, \tag{4.45}$$

then we have a valid inequality, which we call a *commodity-merging inequality*, as follows:

$$n_a \geq \sum_{d \in H} y_{ad} - (|H| - 2). \tag{4.46}$$

Proposition 4.4.5. *$\forall a \in A$ and $H \subseteq D$,* (4.46) *is valid if* (4.45) *is satisfied.*

Proof. Firstly, we have the following inequality:

$$n_a \geq \mu_H^a \left(\sum_{d \in H} y_{ad} - (|H| - 1) \right), \tag{4.47}$$

where $\mu_H^a := \left\lceil \sum_{d \in H} q_{ad} \right\rceil$. Because if for each destination $d \in D$, there is a commodity with destination d that uses the arc a, then the truck use variable n_a should be no less than μ_H^a. Adding all inequalities $n_a \geq y_{ad}$ for all $d \in H$ to (4.47), we obtain

$$\begin{aligned}
(|H|+1)n_a &\geq (\mu_H^a + 1)\sum_{d \in H} y_{ad} - \mu_H^a(|H|-1) \\
\Longrightarrow n_a &\geq \left\lfloor \frac{\mu_H^a + 1}{|H|+1} \right\rfloor \sum_{d \in H} y_{ad} - \frac{\mu_H^a(|H|-1)}{|H|+1} \\
\xRightarrow{n_a, y_{ad} \in \mathbb{Z}} n_a &\geq \left\lfloor \frac{\mu_H^a + 1}{|H|+1} \right\rfloor \sum_{d \in H} y_{ad} - \left\lfloor \frac{\mu_H^a(|H|-1)}{|H|+1} \right\rfloor . \qquad (4.48)
\end{aligned}$$

When (4.45) holds, we have $\mu_H^a = |H|$, and (4.48) becomes

$$n_a \geq \sum_{d \in H} y_{ad} - \left\lfloor \frac{|H|(|H|-1)}{|H|+1} \right\rfloor .$$

Observe that

$$|H| - 2 < \frac{|H|(|H|-1)}{|H|+1} < |H| - 1 \Longrightarrow \left\lfloor \frac{|H|(|H|-1)}{|H|+1} \right\rfloor = |H| - 2,$$

which completes the proof. ■

If $\underline{K}_a \cap K(a,d) \neq \emptyset$ for some $d \in H$ then the definition of μ_H^a could be strengthened, and the requirement (4.45) could be weaker.

Separation

The separation of this class of inequalities can be relatively easy. First note $\sum_{d \in H} y_{ad} - (|H| - 2) \leq 2$, which suggests (4.46) can not be stronger than $n_a \geq 2$ and we are safe to leave out arcs with $\tilde{n}_a \geq 2$.

For each $a \in A$ with $\tilde{n}_a < 2$, we compute $\hat{q}_{ad}$, this can be done in $\mathcal{O}(K)$ operations.

Then we sort $\hat{q}_{ad}$ for all $d \in D$ in descending order $\hat{q}_{ad_1}, \cdots, \hat{q}_{ad_{|D|}}$, which can be done in $\mathcal{O}(|D| \log |D|)$ operations. Finally, we check if $\tilde{n}_a \geq \sum_{i=1}^{p} \tilde{y}_{ad_i} - (p-2)$, for $p = 1, \cdots, |D|$, which needs $\mathcal{O}(|D|)$ operations. Thus, in total, the separation runs in $|A|(|K| + |D| \log |D|)$ time, which is a polynomial of the size of the problem.

4.4.6 Truck-Balancing Inequalities

For a given node i, the *truck-balancing inequalities* focus on the relationship between its incoming and outgoing trucks. More specifically, the truck load on any particular incoming arc to a given node should be no larger than the sum of those on all outgoing arcs after some modification:

$$\sum_{a \in \delta^+(i)} n_a - \left\lfloor Q^+(i) \right\rfloor \geq n_{\hat{a}} - \left\lceil Q^-(i) \right\rceil, \qquad \forall \hat{a} \in \delta^-(i), \tag{4.49}$$

where $Q^+(i) := \sum_{k \in K, o(k) = i} q_k$, $Q^-(i) := \sum_{k \in K, d(k) = i} q_k$. The left hand side is a upper bound on the round up of the net total commodity quantities going out of node i, while the right hand side is a lower bound on the round up of the net commodity quantities coming into i on any given arc $\hat{a} \in \delta^-(i)$.

Similarly, we can have a mirrored version as follows:

$$\sum_{a \in \delta^-(i)} n_a - \lfloor Q^-(i) \rfloor \geq n_{\hat{a}} - \lceil Q^+(i) \rceil, \qquad \forall \hat{a} \in \delta^+(i). \tag{4.50}$$

The above two inequalities can be generalized even further, but may not necessarily be stronger, as follows:

$$\sum_{a \in \delta^+(i)} n_a - \left\lfloor Q^+(i) \right\rfloor \geq \sum_{a \in \delta^-(i)} \left(n_a - \sum_{\substack{d \in D, \\ K(a,d) \neq \emptyset}} y_{ad} \right) - \left\lceil Q^-(i) \right\rceil + 1, \tag{4.51}$$

$$\sum_{a\in\delta^-(i)} n_a - \lfloor Q^-(i)\rfloor \geq \sum_{a\in\delta^+(i)} \left(n_a - \sum_{\substack{d\in D,\\ K(a,d)\neq\emptyset}} y_{ad} \right) - \lceil Q^+(i)\rceil + 1. \tag{4.52}$$

Their validity is due to the right hand side is a valid lower bound on the round up of the net total commodity quantities coming into i for (4.51) (going out of i for (4.52)). Note (4.51) can be weak since we sum up y_{ad} to determine if an arc is in use or not. It can be tightened in the same way as for (4.40) by introducing some additional continuous variables z_a for $a \in \delta^+(i)$:

$$\begin{aligned}
\sum_{a\in\delta^+(i)} n_a - \left\lfloor Q^+(i)\right\rfloor \geq & \sum_{a\in\delta^-(i)} (n_a - z_a) - \left\lceil Q^-(i)\right\rceil + 1, \\
z_a \geq & \; y_{a,d} \qquad \forall a \in \delta^+(i),\ d \in D, k(a,d) \neq \emptyset, \\
z_a \leq & \sum_{\substack{d\in D,\\ K(a,d)\neq\emptyset}} y_{a,d}, \qquad \forall a \in \delta^+(i), \\
0 \leq & \; z_a \leq 1, \qquad \forall a \in \delta^+(i).
\end{aligned} \tag{4.53}$$

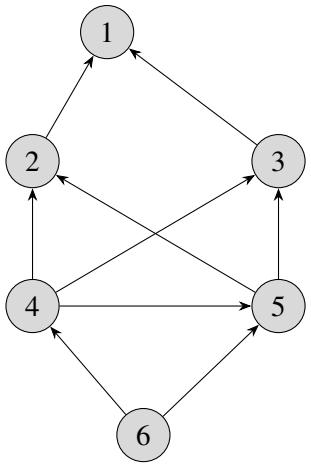
We can refine (4.52) similarly.

Separation

The separation can be done by enumeration in $\mathcal{O}(|V|(|K| + |A|))$ time.

4.4.7 An Illustrative Instance

We end this section by an illustrative instance where the above six classes of instance can be identified and help to improve the dual bound significantly after all simple strengthening inequalities and cut-set inequalities have been added. The network of the instance and the commodity data are shown in Figure 4.6. The cost of each arc is computed as the distance between its two end nodes: $c_{(2,1)} = 5.39$, $c_{(3,1)} = 9.43$, $c_{(4,2)} = 5.00$, $c_{(5,2)} = 11.18$, $c_{(4,3)} = 11.18$, $c_{(5,3)} = 5.00$, $c_{(6,4)} = 6.40$, $c_{(4,5)} = 10.00$, $c_{(6,5)} = 7.81$.



Commodity	Origin	Destination	Quantity
k	$o(k)$	$d(k)$	$q(k)$
1	4	1	0.5
2	5	1	0.6
3	6	1	0.8
4	6	3	0.7

Figure 4.6: Left: complete network for the illustrative instance; Right: commodity data for the illustrative instance.

We call the LP relaxation of the flow-based model *root* as shown in Section 4.1.3 . The optimal value of the root is 36.35. After adding all simple strengthening inequalities shown in (4.20), (4.21), (4.24), (4.25), (4.26), the optimal value becomes 39.21. On top of that, we add cut-set inequalities (4.27) (all of them by enumeration), and the optimal value improves to 44.03. Then we start to identify violated inequalities of the six new classes manually, which is doable since it is not big instance. All the violated inequalities found are listed in Table 4.2. After adding them, the LP value increases to 50.34, and the corresponding optimality gap drops to 1.6%, which suggests those new cutting planes can do a very good drop in closing the gap when no more violated cut-set inequalities can be found. More details on the optimal values and optimality gps are presented in Table 4.3.

Table 4.2: Violated inequalities of the six new classes.

Wheat-stalk	$n_{(2,1)} \geq 2(y_{(2,1),1} + y_{(4,2),1} + y_{(6,4),1} - 2)$	(4.2.1a)
	$n_{(4,2)} \geq 2(y_{(4,2),1} + y_{(6,4),1} - 1)$	(4.2.1b)
Wheat-sheaf	$n_{(4,2)} + n_{(4,3)} + n_{(4,5)} \geq 2y_{(6,4),1}$	(4.2.2)
Combinatorial rounding	$n_{(4,2)} \geq (1/0.8)w_{(4,2)}$	(4.2.3a)
	$n_{(2,1)} \geq (2/1.9)w_{(2,1),1}$	(4.2.3b)
	$n_{(6,5)} \geq (1/0.8)w_{(6,5),1} + (1/0.7)w_{(6,5),3}$	(4.2.3c)
	$n_{(4,5)} \geq (1/1.3)w_{(4,5),1} + (1/0.7)w_{(4,5),3}$	(4.2.3d)
	$n_{(4,3)} \geq (1/1.3)w_{(4,3),1} + (1/0.7)w_{(4,3),3}$	(4.2.3e)
Generalized cut-set	$n_{(2,1)} + n_{(3,1)} \geq y_{(2,1),1} + y_{(3,1),1} + 1$	(4.2.4)
Commodity-merging	$n_{(4,3)} \geq y_{(4,3),1} + y_{(4,3),3}$	(4.2.5)
Truck-balancing	$n_{(2,1)} \geq n_{(4,2)}$	(4.2.6)

Table 4.3: LP solution values with different valid inequalities added. From left to right, inequalities are added incrementally.

	Root	Simple strengthening	Cut-set	(4.2.1a) (4.2.1b) (4.2.2) (4.2.3a) (4.2.3b) (4.2.3c) (4.2.3d) (4.2.3e) (4.2.4) (4.2.5) (4.2.6)
LP value	36.35	39.21	46.16	50.34
Optimality gap	28.09%	23.36%	9.77%	1.60%

4.5 Computational Study

In this section, we present the results of a small proof-of-concept computational study to assess the potential of some of the new classes of inequalities. Specifically, we focus on the wheat-stalk and the generalized cut-set inequalities, and implement the separation heuristics described in Algorithms 2 and 4.

Instances, which are characterized by three parameters n, n_d, and n_c, are generated randomly. The networks are randomly generate 6-regular directed graphs with n nodes,

with each node having 3 incoming and 3 outgoing arcs. The coordinates of the nodes are normally distributed with mean 0 and variance 10. To obtain commodities, we randomly select n_d nodes to form the set of destinations D and for each $d \in D$, we randomly select n_o nodes in $N \setminus \{d\}$ as the origins with a distance of at least 2 to d (i.e., the number of arcs in a path from from origin o to destination d in the network is at least 2) for a total of $n_o \cdot n_d$ commodities. The size of a commodity is randomly chosen from $\{0.4, 0.45, 0.5, \cdots, 0.95\}$. The cost c_a for traversing arc a is set to the length of a, i.e., the Euclidean distance between its two end nodes. For each configuration, i.e., each a combination of n, n_d and n_o, we generate 1000 instances.

In all our experiments, we start from Formulation F3 with all simple strengthening constraints added (i.e., (4.20), (4.21), (4.24), (4.25), and (4.26)) and denote the value of the optimal solution to the LP relaxation by Initial. For convenience, we use CSI, GCSI, and WSI to denote the cut-set, the generalized cut-set, and the wheat-stalk inequalities, respectively.

In the first experiment, we compare the optimality gaps obtained after adding CSI (4.27), GCSI (4.38), and WSI (4.28), respectively, where in each case we add violated inequalities one at a time (and resolve) until no more violated inequalities can be found. The results can be found in Table 4.4 where #-Instances indicates the number of instances for which at least one violated inequality was found and and Gap represent be the average optimality gap after adding all violated inequalities.

We observe that cut-set inequalities are found for every instance and that the cut-set inequalities are most effective in reducing the optimality gap. When the number of origins $n_o = 2$, no violated generalized cut set inequalities can be found even if we increase the number of destinations from 2 to 4. Interestingly, $n_o = 3$ seems to be the best in this case, which can be seen by comparing all configurations with the same n and n_d. When the number of destinations increases, the number of instances for which wheat-stalk inequalities are found increases as well. This is also true when the number of origins

Table 4.4: Comparison on the number of instances where a violated CSI, GCSI and WSI can be found, respectively, and the corresponding optimality gap after all violated inequalities of each class have been added.

				#-Instances			Gap			
	n	n_o	n_d	CSI	GCSI	WSI	Initial	CSI	GCSI	WSI
Config_1	8	2	2	998	0	11	18.14%	0.29%	18.14%	17.89%
Config_2	8	2	3	1000	0	56	19.45%	1.25%	19.45%	18.45%
Config_3	8	2	4	1000	0	152	19.57%	2.67%	19.57%	16.91%
Config_4	8	3	2	1000	135	28	17.33%	1.32%	15.27%	16.84%
Config_5	8	3	3	1000	202	161	17.77%	2.82%	15.00%	15.27%
Config_6	8	3	4	1000	276	352	17.22%	4.18%	13.78%	12.57%
Config_7	8	4	2	1000	109	74	16.40%	2.09%	14.95%	15.23%
Config_8	8	4	3	1000	166	244	16.29%	3.63%	14.36%	13.05%
Config_9	8	4	4	1000	190	511	15.55%	4.81%	13.61%	10.10%
Config_{10}	10	2	2	1000	0	12	18.39%	0.32%	18.39%	18.13%
Config_{11}	10	2	3	1000	0	56	19.21%	1.20%	19.21%	18.20%
Config_{12}	10	2	4	1000	0	155	19.92%	2.49%	19.92%	17.20%
Config_{13}	10	3	2	1000	134	24	17.35%	1.18%	15.23%	16.91%
Config_{14}	10	3	3	1000	193	141	17.66%	2.56%	14.88%	15.47%
Config_{15}	10	3	4	1000	243	331	17.31%	3.79%	14.23%	12.77%
Config_{16}	10	4	2	1000	108	54	16.44%	1.79%	14.90%	15.58%
Config_{17}	10	4	3	1000	145	238	16.37%	3.33%	14.58%	13.20%
Config_{18}	10	4	4	1000	187	432	15.66%	4.72%	13.79%	10.97%
Config_{19}	12	2	2	1000	0	6	18.40%	0.31%	18.40%	18.28%
Config_{20}	12	2	3	1000	0	35	19.39%	1.07%	19.39%	18.69%
Config_{21}	12	2	4	1000	0	90	19.57%	1.94%	19.57%	17.89%
Config_{22}	12	3	2	1000	118	16	17.38%	1.07%	15.34%	17.08%
Config_{23}	12	3	3	1000	189	104	17.82%	2.22%	14.87%	16.11%
Config_{24}	12	3	4	1000	240	220	17.36%	3.53%	14.20%	14.27%
Config_{25}	12	4	2	1000	94	45	16.47%	1.60%	15.10%	15.75%
Config_{26}	12	4	3	1000	149	201	16.42%	3.13%	14.53%	13.71%
Config_{27}	12	4	4	1000	189	403	15.96%	4.47%	13.93%	11.38%

increases. We also observe that when the total number of commodities is large, WSI can be more useful than GCSI. This conclusion can also be confirmed by comparing different configurations when the number of nodes n is fixed. For $n_o = 3$ and $n_d = 2$, which is 6 commodities in total, WSI is much worse than GCSI in terms of #-Instances and Gap. When $n_o = 4$ and $n_d = 4$, i.e., 16 commodities in total, the reverse applies. With $n_o = 3$ and $n_d = 4$, which is 8 commodities in total, the difference between those two are much less pronounced.

However, if we focus on those instances where a GCSI can be found, we conclude that GCSI on average is stronger than CSI and WSI. Similarly, if we only consider instances where a WSI can be found, the conclusion will be WSI on average is stronger than CSI and GCSI. The detailed comparison can be found in Table 4.5 and Table 4.6.

Table 4.5: Comparison on the number of instances where a violated CSI, GCSI and WSI can be found, respectively, and the corresponding optimality gap after all violated inequalities of each class have been added, when only considering instances where a GCSI can be found.

				#-Instances			Gap			
	n	n_o	n_d	CSI	GCSI	WSI	Root	CSI	GCSI	WSI
Config_4	8	3	2	135	135	7	18.48%	3.70%	3.15%	17.49%
Config_5	8	3	3	202	202	43	17.88%	4.37%	4.18%	14.67%
Config_6	8	3	4	276	276	112	17.10%	4.74%	4.65%	12.03%
Config_7	8	4	2	109	109	5	16.80%	3.79%	3.52%	16.19%
Config_8	8	4	3	166	166	47	16.41%	4.93%	4.83%	13.11%
Config_9	8	4	4	190	190	117	15.84%	5.72%	5.66%	9.34%
Config_{13}	10	3	2	134	134	7	18.73%	3.40%	2.87%	17.87%
Config_{14}	10	3	3	193	193	30	18.20%	3.98%	3.81%	15.79%
Config_{15}	10	3	4	243	243	92	17.29%	4.67%	4.59%	12.32%
Config_{16}	10	4	2	108	108	12	17.42%	3.51%	3.16%	15.69%
Config_{17}	10	4	3	145	145	43	17.00%	4.72%	4.61%	13.18%
Config_{18}	10	4	4	187	187	78	15.86%	5.91%	5.85%	11.72%
Config_{22}	12	3	2	118	118	5	20.49%	3.74%	3.24%	19.72%
Config_{23}	12	3	3	189	189	22	19.18%	3.73%	3.55%	17.20%
Config_{24}	12	3	4	240	240	66	17.91%	4.82%	4.73%	14.20%
Config_{25}	12	4	2	94	94	10	17.73%	3.45%	3.15%	16.13%
Config_{25}	12	4	3	149	149	31	16.91%	4.34%	4.24%	14.22%
Config_{27}	12	4	4	189	189	87	16.24%	5.59%	5.52%	11.26%

Table 4.6: Comparison on the number of instances where a violated CSI, GCSI and WSI can be found, respectively, and the corresponding optimality gap after all violated inequalities of each class have been added, when only considering instances where a WSI can be found.

				#-Instances			Gap			
	n	n_o	n_d	CSI	GCSI	WSI	Root	CSI	GCSI	WSI
Config_{1}	8	2	2	11	0	11	24.85%	2.40%	24.85%	2.35%
Config_{2}	8	2	3	56	0	56	21.52%	3.78%	21.52%	3.72%
Config_{3}	8	2	4	152	0	152	21.31%	3.89%	21.31%	3.84%
Config_{4}	8	3	2	28	7	28	20.32%	2.83%	15.58%	2.73%
Config_{5}	8	3	3	161	43	161	19.00%	3.54%	14.98%	3.47%
Config_{6}	8	3	4	352	112	352	17.74%	4.59%	13.77%	4.53%
Config_{7}	8	4	2	74	5	74	18.48%	2.84%	17.58%	2.79%
Config_{8}	8	4	3	244	47	244	17.29%	4.07%	15.06%	4.02%
Config_{9}	8	4	4	511	117	511	15.85%	5.23%	13.44%	5.18%
Config_{10}	10	2	2	12	0	12	25.25%	3.38%	25.25%	3.35%
Config_{11}	10	2	3	56	0	56	21.27%	3.21%	21.27%	3.16%
Config_{12}	10	2	4	155	0	155	21.22%	3.68%	21.22%	3.65%
Config_{13}	10	3	2	24	7	24	21.88%	3.73%	17.13%	3.46%
Config_{14}	10	3	3	141	30	141	19.00%	3.58%	15.71%	3.50%
Config_{15}	10	3	4	331	92	331	17.91%	4.23%	14.28%	4.18%
Config_{16}	10	4	2	54	12	54	19.29%	3.32%	15.83%	3.20%
Config_{17}	10	4	3	238	43	238	17.15%	3.87%	14.83%	3.83%
Config_{18}	10	4	4	432	78	432	15.99%	5.17%	14.20%	5.13%
Config_{19}	12	2	2	6	0	6	25.26%	4.28%	25.26%	4.28%
Config_{20}	12	2	3	35	0	35	23.01%	2.94%	23.01%	2.92%
Config_{21}	12	2	4	90	0	90	22.23%	3.60%	22.23%	3.56%
Config_{22}	12	3	2	16	5	16	20.81%	2.13%	15.15%	2.03%
Config_{23}	12	3	3	104	22	104	19.51%	3.16%	15.93%	3.10%
Config_{24}	12	3	4	220	66	220	18.39%	4.36%	14.36%	4.31%
Config_{25}	12	4	2	45	10	45	18.54%	2.68%	15.22%	2.62%
Config_{26}	12	4	3	201	31	201	17.24%	3.78%	15.25%	3.74%
Config_{27}	12	4	4	403	87	403	16.27%	4.96%	13.94%	4.92%

Table 4.7: Comparison on the corresponding optimality gap for all instances.

	n	n_o	n_d	Gap		
				Root	CSI	CSI + GCSI + WSI
Config_4	8	3	2	17.33%	1.32%	1.24%
Config_5	8	3	3	17.77%	2.82%	2.78%
Config_6	8	3	4	17.22%	4.18%	4.14%
Config_7	8	4	2	16.40%	2.09%	2.06%
Config_8	8	4	3	16.29%	3.63%	3.61%
Config_9	8	4	4	15.55%	4.81%	4.77%
Config_{13}	10	3	2	17.35%	1.18%	1.10%
Config_{14}	10	3	3	17.66%	2.56%	2.52%
Config_{15}	10	3	4	17.31%	3.79%	3.76%
Config_{16}	10	4	2	16.44%	1.79%	1.75%
Config_{17}	10	4	3	16.37%	3.33%	3.30%
Config_{18}	10	4	4	15.66%	4.72%	4.69%
Config_{22}	12	3	2	17.38%	1.07%	1.01%
Config_{23}	12	3	3	17.82%	2.22%	2.19%
Config_{24}	12	3	4	17.36%	3.53%	3.50%
Config_{25}	12	4	2	16.47%	1.60%	1.57%
Config_{26}	12	4	3	16.42%	3.13%	3.11%
Config_{27}	12	4	4	15.96%	4.47%	4.44%

In our final experiment, we explore whether GCSI and WSI can add value on top of CSI. More specifically, after solving the initial linear programming relaxation (with simple strengthening inequalities), we first add violated CSI (one by one until we can no longer find any), then we add violated GCSI (one by one and until we can no longer find any), and then violated WSI (until we can no longer find any). The results are shown in Table 4.7. We observe that only modest gains are achieved when adding the two new sets of inequalities for these instances.

4.6 Conclusions and Future Work

In this chapter, we try to strengthen the compact but relatively weak flow-based formulation for the SNDPITC. The newly proposed six classes of cutting planes show promise in closing the optimality gap when the well-known cut-set inequalities can not provide any further improvement. One future direction is to develop more efficient separation heuris-

tics. A more extensive numerical study will be helpful for identifying special structures where the new cutting planes can be more helpful.

CHAPTER 5

SUBSTITUTION-BASED EQUIPMENT BALANCING IN SERVICE NETWORKS WITH MULTIPLE EQUIPMENT TYPES

A load plan for the coming week, i.e., the loads to be moved and the driver schedules to make this happen, often induces a change in the inventory of an equipment type at a facility at the end of the week. In other words, the number of a specific equipment type departing from the facility is not equal to the number arriving. The imbalance (induced by a plan) of a facility is defined to be the sum of the imbalances (surplus or deficit) of all equipment types at the facility, and the total imbalance (induced by a plan) is the sum of the imbalances of the facilities in the network.

We seek to decrease the imbalance introduced by a plan via substituting the original equipment types assigned to the loads according to the plan. Equipment substitution complements empty repositioning of equipment, but is only possible if companies operate multiple, exchangeable equipment types. It has the advantage (over empty repositioning) of not incurring any costs. The primary goal of substitution-based equipment balancing is to minimize the total imbalance induced by a plan. A secondary goal is to achieve the minimum total imbalance with as few equipment substitutions as possible. That is, substitution-based equipment balancing is a hierarchical optimization problem.

In this chapter, we explore computational complexity questions arising when using an MIP to decrease the imbalance of a given load plan.

The remainder of the chapter is organized as follows. Section 5.1 introduces notation and presents integer programming formulations. Section 5.2 gives the results of our complexity analysis. We include some concluding remarks in Section 5.3.

5.1 Notation and formulations

We first introduce the notation used throughout this chapter. A network N is represented as a directed graph, $N = (V, A)$, with each vertex representing a facility and each arc representing a load. A load represents a movement of equipment that is scheduled to dispatch during the planning horizon and deliver a quantity of packages for an origin-destination pair. It is also also characterized by the initial equipment type assigned to it which is used to compute the initial imbalance. Let $\mathbb{Z}_{\geq 0}$ be the set of non-negative integers, $n = |V|$ be the number of vertices, and $m = |A|$ be the number of arcs. Let $\mathcal{E}$ be the set of basic equipment types and $\mathcal{C}$ be the set of equipment type configurations used operationally, which can be a basic type or a composite type formed by combining basic types. For $c \in \mathcal{C}$, we have $c = \sum_{e \in \mathcal{E}} f_{ce} e$ with $f_{ce} \in \mathbb{Z}_{\geq 0}$ indicating how many units of basic equipment type $e \in \mathcal{E}$ are used in the composite type c. For each $v \in V$, let $\mathcal{C}_v \subseteq \mathcal{C}$ be the set of allowable equipment types at vertex v, and $\delta_v^+ = \{(v, u) \in A : u \in V\}$ and $\delta_v^- = \{(u, v) \in A : u \in V\}$ be the sets of outgoing and incoming arcs at v, respectively. Let $\sigma_v^+ := |\delta_v^+|$ and $\sigma_v^- := |\delta_v^-|$. An equipment assignment (or assignment for short) is a function $\mathscr{A} : A \to \mathcal{C}$ that assigns an equipment type to each arc. The initial assignment is denoted by $\mathscr{A}_0$. Let $\mathcal{C}_a \subseteq \mathcal{C}$ be the set of equipment types that can be assigned to arc $a \in A$, $A_{vc}^+ := \{a \in \delta_v^+ : c \in \mathcal{C}_a\}$, and $A_{vc}^- := \{a \in \delta_v^- : c \in \mathcal{C}_a\}$. For a given network N, let I^* be the minimum imbalance and $I(\mathscr{A})$ be the imbalance of an assignment $\mathscr{A}$. When $I(\mathscr{A}) = I^*$, we say $\mathscr{A}$ is optimal for N.

Our goal is to find an optimal $\mathscr{A}^*$ that is closest to $\mathscr{A}_0$, i.e., $\mathscr{A}^* \in \text{argmin}_{I(\mathscr{A})=I^*} \|\mathscr{A} - \mathscr{A}_0\|$, where $\|\mathscr{A} - \mathscr{A}_0\| = |\{a \in A : \mathscr{A}(a) \neq \mathscr{A}_0(a)\}|$. We use a two-stage hierarchical optimization approach, where we compute I^* for network N in Stage 1, and find the desired $\mathscr{A}^*$ in Stage 2.

5.1.1 Minimizing imbalance

We define the following variables. For $a \in A$ and $c \in \mathcal{C}_a$, let

$$y_{ac} = \begin{cases} 1, & \text{if equipment } c \text{ is used on arc } a, \\ 0, & \text{otherwise.} \end{cases}$$

For $v \in V$, $e \in \mathcal{C}_v \cap \mathcal{E}$, let $r_{ve} \in \mathbb{Z}_{\geq 0}$ be the imbalance for basic equipment type e at vertex v. The following model minimizes the total imbalance I^*:

$$I^* = \min \quad \sum_{v \in V} \sum_{e \in \mathcal{C}_v \cap \mathcal{E}} r_{ve} \tag{5.1}$$

$$\text{s.t.} \quad \sum_{c \in \mathcal{C}} f_{ce} \left(\sum_{a \in A_{vc}^+} y_{ac} - \sum_{a \in A_{vc}^-} y_{ac} \right) \leq r_{ve}, \quad v \in V,\ e \in \mathcal{C}_v \cap \mathcal{E}, \tag{5.2}$$

$$\sum_{c \in \mathcal{C}} f_{ce} \left(\sum_{a \in A_{vc}^-} y_{ac} - \sum_{a \in A_{vc}^+} y_{ac} \right) \leq r_{ve}, \quad v \in V,\ e \in \mathcal{C}_v \cap \mathcal{E}, \tag{5.3}$$

$$\sum_{c \in \mathcal{C}_a} y_{ac} = 1, \quad a \in A, \tag{5.4}$$

$$y_{ac} \in \{0, 1\}, \quad a \in A,\ c \in \mathcal{C}_a. \tag{5.5}$$

Constraints (5.2) and (5.3) ensure that r_{ve} is set to the net surplus or deficit of equipment type e at vertex v, while Constraint (5.4) guarantees that exactly one equipment type is assigned to each arc (i.e., the assigned equipment type remains the same or is replaced by exactly one other equipment type). Note that the minimum imbalance induced by a plan depends on both the loads and the initial assignment, $\mathscr{A}_0$, since $\mathcal{C}_a$ depends on $\mathscr{A}_0$.

5.1.2 Minimizing the number of changes required to achieve the minimum imbalance

In Stage 2, we minimize the number of changes Ω and adding Constraint (5.6) ensures that the obtained equipment assignment is an optimal assignment:

$$\Omega = \min \sum_{a \in A} (1 - y_{a\mathscr{A}_0(a)})$$

s.t.

$$\sum_{v \in V} \sum_{e \in \mathcal{C}_v \cap \mathcal{E}} r_{ve} \leq I^*, \tag{5.6}$$

$$(5.2), (5.3), (5.4), (5.5).$$

Note that $\Omega < m$, and, thus, the two optimization models can be combined into a single optimization model as follows:

$$\min \quad \sum_{v \in V} \sum_{e \in \mathcal{C}_v \cap \mathcal{E}} m r_{ve} + \sum_{a \in A} (1 - y_{a\mathscr{A}_0(a)})$$

$$\text{s.t.} \quad (5.2), (5.3), (5.4), (5.5).$$

However, for real-life instances, m can be as large as hundreds of thousands, which makes it more difficult to solve than the two-stage hierarchical optimization model.

5.2 Complexity of equipment substitution problems

In this section, we analyze the computational complexity of some simplified settings of the equipment substitution problem. For simplicity, we only consider basic equipment types, i.e., $\mathcal{C} = \mathcal{E}$ and an assignment $\mathscr{A}$ can be viewed as a function $A \to \mathcal{E}$. We consider two variants: one where there is *full interchangeability*, i.e., the equipment assigned to all arcs in the network can be changed, and the other where there is *partial interchangeability*, i.e., the equipment assigned to a given subset of arcs cannot be changed. Furthermore, we consider settings with two and three equipment types. The findings, to be discussed next,

are summarized in Table 5.1.

Table 5.1: Complexity of equipment substitution problems in different simplified settings.

	full interchangeability		partial interchangeability	
	2 equipment types	3 equipment types	2 equipment types	3 equipment types
minimizing imbalance	P	P	P	NP-hard
minimizing number of changes	P	NP-hard	P	NP-hard

For networks with full interchangeability, the following claim gives a necessary and sufficient condition for an assignment function $\mathscr{A}$ to be optimal. Let $\sigma_{ve}^{+} = |\{a \in \delta_v^{+} : \mathscr{A}(a) = e\}|$ and $\sigma_{ve}^{-} = |\{a \in \delta_v^{-} : \mathscr{A}(a) = e\}|$.

Claim 1. *Given a network N with full interchangeability and an assignment $\mathscr{A}$, then $I(\mathscr{A}) = I^*$ if and only if for all $v \in V$ and for all $e \in \mathcal{E}$, $(\sigma_{ve}^{+} - \sigma_{ve}^{-})(\sigma_v^{+} - \sigma_v^{-}) \geq 0$ if $\sigma_v^{+} - \sigma_v^{-} \neq 0$ and $\sigma_{ve}^{+} - \sigma_{ve}^{-} = 0$ if $\sigma_v^{+} - \sigma_v^{-} = 0$.*

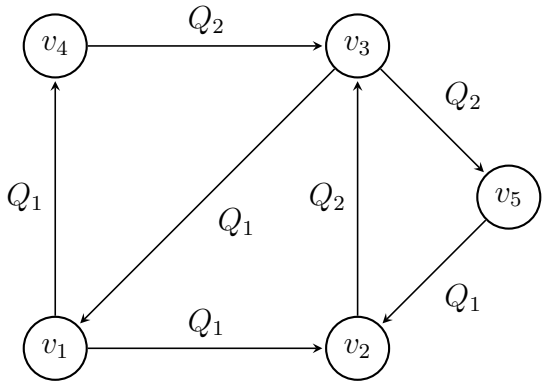
Proof. For each node v, the minimum imbalance is at least $|\sigma_v^{+} - \sigma_v^{-}|$, which implies $\sum_{v \in V} |\sigma_v^{+} - \sigma_v^{-}|$ is a lower bound of I^*. It's easy to see this bound can be achieved by $\mathscr{A}(a) := e, \forall a \in A$ and a fixed $e \in \mathcal{E}$. By definition, for all $v \in V$, $\sum_{e \in \mathcal{E}} \sigma_{ve}^{+} = \sigma_v^{+}$ and $\sum_{e \in \mathcal{E}} \sigma_{ve}^{-} = \sigma_v^{-}$, which implies $\sum_{e \in \mathcal{E}} (\sigma_{ve}^{+} - \sigma_{ve}^{-}) = \sigma_v^{+} - \sigma_v^{-}$. Now $I(\mathscr{A}) = I^*$ if and only if for all $v \in V$, $\sum_{e \in \mathcal{E}} |\sigma_{ve}^{+} - \sigma_{ve}^{-}| = |\sigma_v^{+} - \sigma_v^{-}|$. This, in turn, is true if and only if for all $e \in \mathcal{E}$ we have $(\sigma_{ve}^{+} - \sigma_{ve}^{-})(\sigma_v^{+} - \sigma_v^{-}) \geq 0$ if $\sigma_v^{+} - \sigma_v^{-} \neq 0$ and $\sigma_{ve}^{+} - \sigma_{ve}^{-} = 0$ if $\sigma_v^{+} - \sigma_v^{-} = 0$. ■

5.2.1 Two-equipment type networks with full interchangeability

Given a network N with two equipment types, i.e., $\mathcal{E} = \{Q_1, Q_2\}$, and full interchangeability, it is easy to see that both $\mathscr{A}(a) = Q_1$ for all $a \in A$ and $\mathscr{A}(a) = Q_2$ for all $a \in A$ achieve minimum imbalance. However, it is not obvious how to find an assignment that minimizes the number of changes required to achieve minimum imbalance. We start by giving an example that demonstrates that the natural greedy heuristic, which identifies

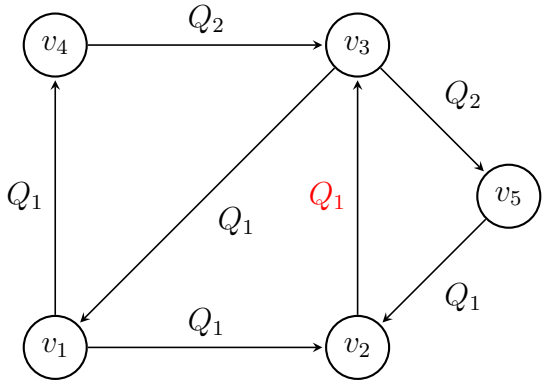
an arc for which swapping the equipment type reduces the imbalance the most and then performs that swap, does not necessarily yield the desired $\mathscr{A}^*$.

A counterexample for the greedy heuristic



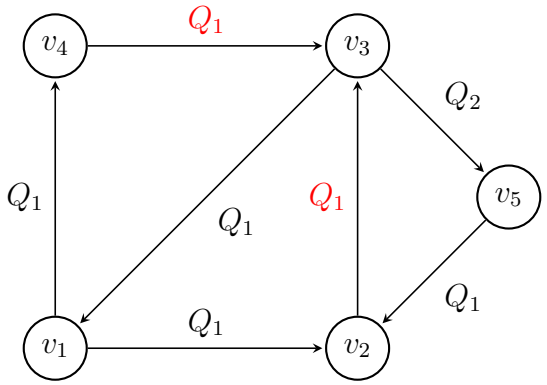
Imbalance	Q_1	Q_2
v_1	1	0
v_2	2	1
v_3	1	1
v_4	1	1
v_5	1	1
Total	10	

In Iteration 1, the greedy heuristic may change the equipment type on (v_2, v_3) from Q_2 to Q_1 as it reduces the imbalance by 4 (the maximum possible).



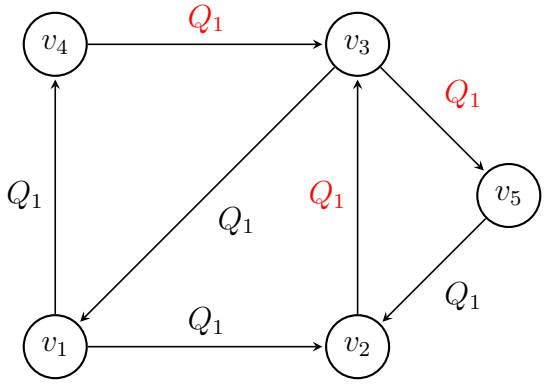
Imbalance	Q_1	Q_2
v_1	1	0
v_2	1	0
v_3	0	0
v_4	1	1
v_5	1	1
Total	6	

In Iteration 2, the greedy heuristic may change the equipment type on (v_4, v_3) from Q_2 to Q_1 even though it does not reduce the imbalance.



Imbalance	Q_1	Q_2
v_1	1	0
v_2	1	0
v_3	1	1
v_4	0	0
v_5	1	1
Total	6	

Finally, in Iteration 3, the greedy heuristic will change the equipment type on (v_3, v_5) from Q_2 to Q_1, which reduces the imbalance by 4. The minimum imbalance is achieved in three iterations.



Imbalance	Q_1	Q_2
v_1	1	0
v_2	1	0
v_3	0	0
v_4	0	0
v_5	0	0
Total	2	

However, it is easy to see that the minimum imbalance can be reduced to 2 by only 2 equipment type changes: change the equipment type on (v_5, v_2) from Q_1 to Q_2 and on (v_4, v_3) from Q_2 to Q_1.

To show that an optimal assignment closest to the initial assignment can be found in polynomial time, we formulate the problem as an integer program and show that the coefficient matrix is totally unimodular.

IP formulation and polynomial solvability

Given that there are only two equipment types, a simpler formulation for the Stage 2 problem can be constructed. Let $A_1 := \{a \in A : \mathscr{A}_0(a) = Q_1\}$ and $A_2 := \{a \in A : \mathscr{A}_0(a) = Q_2\}$, and for all $a \in A$ let

$$x_a = \begin{cases} 1 & \text{if equipment type } Q_1 \text{ is assigned to arc } a, \\ 0 & \text{otherwise.} \end{cases}$$

Minimizing the number of changes required to achieve the minimum imbalance can be modeled as an integer program (IP) as follows

$$\begin{aligned}
\text{min} \quad & \sum_{a \in A_1} (1 - x_a) + \sum_{a \in A_2} x_a \\
\text{s.t.} \quad & 0 \leq \sum_{a \in \delta_v^+} x_a - \sum_{a \in \delta_v^-} x_a \leq \sigma_v^+ - \sigma_v^-, \quad v \in V, \sigma_v^+ > \sigma_v^-, && (5.7) \\
& \sigma_v^+ - \sigma_v^- \leq \sum_{a \in \delta_v^+} x_a - \sum_{a \in \delta_v^-} x_a \leq 0, \quad v \in V, \sigma_v^+ < \sigma_v^-, && (5.8) \\
& \sum_{a \in \delta_v^+} x_a - \sum_{a \in \delta_v^-} x_a = 0, \quad v \in V, \sigma_v^+ = \sigma_v^-, && (5.9) \\
& x_a \in \{0, 1\}, \quad a \in A.
\end{aligned}$$

Validity of the formulation follows from the fact that Constraints (5.7), (5.8), and (5.9) guarantee that the condition in Claim 1 is satisfied. In addition, the objective function forces the solution to be closest to the initial assignment $\mathscr{A}_0$.

The above IP can be written in the following form

$$\begin{aligned}
\text{min} \quad & w^T x \\
\text{s.t.} \quad & l \leq Lx \leq u, \\
& x \in \{0, 1\}^m.
\end{aligned}$$

where L is the node-arc incidence matrix of the network. Because a node-arc incidence matrix of a network is totally unimodular and $l, u \in \mathbb{Z}^n$, the LP relaxation will return an integral solution. Since LPs can be solved in polynomial time, we have shown its polynomial solvability.

5.2.2 Two-equipment type networks with partial interchangeability

As before, let $A_1 := \{a \in A : \mathscr{A}_0(a) = Q_1\}$ and $A_2 := \{a \in A : \mathscr{A}_0(a) = Q_2\}$. In this setting, we assume that the equipment types on the two given sets $X_1 \subseteq A_1$ and $X_2 \subseteq A_2$ cannot be changed, i.e., the equipment type Q_1 assigned to arcs in X_1 cannot be changed to Q_2, and the equipment Q_2 assigned to arcs in X_2 cannot be changed to Q_1.

Minimizing imbalance

Determining the minimum imbalance I^*, when certain equipment type assignments cannot be changed, is no longer trivial, but can be accomplished using the following IP:

$$\text{min} \quad \sum_{v \in V} (r_{v,1} + r_{v,2})$$

$$\text{s.t.} \quad \sum_{a \in \delta_v^+ \setminus X_1} x_a + |\delta_v^+ \cap X_1| - \sum_{a \in \delta_v^- \setminus X_1} x_a - |\delta_v^- \cap X_1| \leq r_{v1}, \quad v \in V, \tag{5.10}$$

$$\sum_{a \in \delta_v^+ \setminus X_2} (1 - x_a) + |\delta_v^+ \cap X_2| - \sum_{a \in \delta_v^- \setminus X_2} (1 - x_a) - |\delta_v^- \cap X_2| \leq r_{v2}, \quad v \in V, \tag{5.11}$$

$$\sum_{a \in \delta_v^- \setminus X_1} x_a + |\delta_v^- \cap X_1| - \sum_{a \in \delta_v^+ \setminus X_1} x_a - |\delta_v^+ \cap X_1| \leq r_{v1}, \quad v \in V, \tag{5.12}$$

$$\sum_{a \in \delta_v^- \setminus X_2} (1 - x_a) - |\delta_v^- \cap X_2| - \sum_{a \in \delta_v^+ \setminus X_2} (1 - x_a) - |\delta_v^+ \cap X_2| \leq r_{v2}, \quad v \in V, \tag{5.13}$$

$$x_a \in \{0, 1\}, \quad a \in A \setminus (X_1 \cup X_2), \tag{5.14}$$

$$r_{v1}, r_{v2} \in \mathbb{R}, \quad v \in V, \tag{5.15}$$

where

$$x_a = \begin{cases} 1 & \text{if equipment type } Q_1 \text{ is assigned to arc } a, \\ 0 & \text{otherwise,} \end{cases}$$

and $r_{v1}, r_{v2} \in \mathbb{Z}_{\geq 0}$ represent the imbalance for equipment type Q_1, Q_2 at vertex v, respectively.

The LP relaxation of the above IP can be written in the following form:

$$\begin{aligned} \min \quad & (0, w) \begin{pmatrix} x \\ r \end{pmatrix} \\ \text{s.t.} \quad & \begin{pmatrix} B & -I \\ -B & -I \end{pmatrix} \begin{pmatrix} x \\ r \end{pmatrix} \leq \begin{pmatrix} d \\ -d \end{pmatrix}, \\ & 0 \leq x_a \leq 1, \quad a \in A \setminus (X_1 \cup X_2), \\ & r \in \mathbb{R}^{2n}. \end{aligned}$$

Claim 2. *Polyhedron* $P := \left\{ (x, r) : \begin{pmatrix} B & -I \\ -B & -I \end{pmatrix} \begin{pmatrix} x \\ r \end{pmatrix} \leq \begin{pmatrix} d \\ -d \end{pmatrix}, \ 0 \leq x \leq 1, r \in \mathbb{R} \right\}$ *has integral extreme points.*

Proof. B has $2n$ rows and can be written in the form $\begin{pmatrix} \hat{L} \\ -\hat{L} \end{pmatrix}$, where $\hat{L} = (l_a)_{a \in A \setminus (X_1 \cup X_2)}$ is a submatrix of the node-arc incidence matrix L, and l_a is the column of L that corresponds to arc a. Since L is TU, so are $\hat{L}$ and B. Suppose (x^*, r^*) is an extreme point of P and $\mathcal{I} \subseteq [4n]$ is the index set of active constraints at (x^*, r^*). Note that if there exists a $j \in [2n]$ such that both j and $j+2n$ are in $\mathcal{I}$, then $b_j x^* - u_j r^* = d_j$ and $-b_j x^* - u_j r^* = -d_j$, where b_j is the j-th row of B and u_j is the j-th unit vector. Then $u_j r^* = 0$ and only one of the constraints is necessary. Let $\mathcal{I}_1 = \{j : j \in \mathcal{I} \cap [2n], j + 2n \notin \mathcal{I}\}$, $\mathcal{I}_2 = \{j : j \in \mathcal{I} \cap ([4n] \setminus [2n]), j - 2n \notin \mathcal{I}\}$, $\mathcal{I}_3 = \{j : j \in \mathcal{I} \cap [2n], j + 2n \in \mathcal{I}\}$, $H_1 = \{j : r^*_j = 0\}$,

$H_2 = [2n] \setminus H_1$, $r^1 = (r^*_j)_{j \in H_1}$, $r^2 = (r^*_j)_{j \in H_2}$. Then (x^*, r^2) is an extreme point of

$$\hat{P} = \left\{ (x, y) : \begin{pmatrix} \widehat{B} & -I \end{pmatrix} \begin{pmatrix} x \\ \\ y \end{pmatrix} = \hat{d}, \;\; 0 \leq x \leq 1, y \in \mathbb{R} \right\},$$

where $\widehat{B} = \begin{pmatrix} (b_j)_{j \in \mathcal{I}_1 \cup \mathcal{I}_3} \\ \\ -(b_j)_{j \in \mathcal{I}_2} \end{pmatrix}$ is a submatrix of B, $\hat{d} = \begin{pmatrix} (d_j)_{j \in \mathcal{I}_1 \cup \mathcal{I}_3} \\ \\ -(d_j)_{j \in \mathcal{I}_2} \end{pmatrix} \in \mathbb{Z}^{|\mathcal{I}_1|+|\mathcal{I}_2|+|\mathcal{I}_3|}$. Since B is TU, so are $\widehat{B}$ and $\begin{pmatrix} \widehat{B} & -I \end{pmatrix}$. Therefore, every extreme point of $\hat{P}$ is integral and (x^*, r^2) is integral. Consequently, (x^*, r^*) is integral (since $r^1 = 0$). ■

Minimizing the number of changes to achieve minimum imbalance

The integrality property derived in the previous section relies on the structure of the coefficient matrix. Therefore, if we modify the objective, but keep the coefficient matrix the same, the solution to the LP relaxation will still be integral. As mentioned in Section 5.1, we can combine the optimization models of the two stages into a single optimization model by changing the objective. In the case of two equipment types this results in

$$\begin{aligned} \min \quad & \sum_{v \in V} m(r_{v,1} + r_{v,2}) + \sum_{a \in A_1 \setminus X_1} (1 - x_a) + \sum_{a \in A_2 \setminus X_2} x_a \\ \text{s.t.} \quad & (5.10), (5.11), (5.12), (5.13), (5.14), (5.15). \end{aligned}$$

Observing that $\sum_{a \in A_1 \setminus X_1} (1 - x_a) + \sum_{a \in A_2 \setminus X_2} x_a \leq |A_1 \setminus X_1| + |A_2 \setminus X_2| = |A \setminus (X_1 \cup X_2)| < m$, the optimal solution is guaranteed to achieve the minimum imbalance while minimizing the number of changes made to the original assignment.

5.2.3 Two-equipment type models: Additional results

A more general question, for a given $K \in \mathbb{Z}_{\geq 0}$ and a given $\mathscr{A}_0$, is whether an $\mathscr{A}'$ can be found such that $I(\mathscr{A}') \leq K$ and $\|\mathscr{A} - \mathscr{A}'\|$ is minimum in a network with two equipment

types and full interchangeability. Next, we will show that to solve this problem, it suffices to solve K integer minimum cost circulation problems in an auxiliary network $G' = (V', E')$, which has size polynomial in the size of the original network G, and is constructed as follows.

We add three vertices $\{s, t, s'\}$ to V, and thus $V' = V \cup \{s, t, s'\}$. Let $E' = E \cup E^+ \cup E^- \cup \{s's, ts'\}$, where $E^+ = \{sv : \sigma_v^+ > \sigma_v^-, v \in V\}$ and $E^+ = \{vt : \sigma_v^- > \sigma_v^+, v \in V\}$. Define the capacity function $u : E' \to \mathbb{Z}_{\geq 0}$ and cost function $c : E' \to \mathbb{Z}$ as follows

$$u(e) = \begin{cases} \sigma_v^+ - \sigma_v^-, & \text{if } e \in E^+, \\ \sigma_v^- - \sigma_v^+, & \text{if } e \in E^-, \\ 1, & \text{if } e \in E, \\ k, & \text{if } e = s's \text{ or } ts', \end{cases} \qquad c(e) = \begin{cases} -1, & \text{if } e \in A_1, \\ 1, & \text{if } e \in A_2, \\ 0, & \text{if } e \in E^+ \cup E^- \cup \{s's, ts'\}, \end{cases}$$

where $k \in \mathbb{Z}_{\geq 0}, k \leq K$. It's easy to see that there exists an integer circulation in G'. Suppose we have a minimum cost integer circulation $f_k : E' \to \mathbb{Z}_{\geq 0}$ of G' and define the assignment function $\mathscr{A}^k$ as

$$\mathscr{A}^k(e) = \begin{cases} Q_1, & \text{if } e \in E, f_k(e) = 1, \\ Q_2, & \text{if } e \in E, f_k(e) = 0. \end{cases}$$

By construction, the imbalance of equipment Q_1 for $\mathscr{A}^k$ is less than or equal to k. Thus if we solve this problem for each $k \in \{0, 1, \ldots, K\}$, then we are able to find an $\mathscr{A}'$ such that $I(\mathscr{A}') \leq K$ and thus $T := \{0 \leq k \leq K : I(\mathscr{A}^k) \leq K\}$ is not empty. Let $k^* = \text{argmin}_{k \in T} \|\mathscr{A}^k - \mathscr{A}^0\|$. Since for each k, f_k achieves the minimum cost, then $\mathscr{A}^{k^*}$ is closest to $\mathscr{A}_0$ with $I(\mathscr{A}^{k^*}) \leq K$ over all assignment functions. Due to the integrality of capacities, we can find the minimum cost integer circulation in strongly polynomial-time ([69]). Given that $K \leq 2m$, finding the desired $\mathscr{A}' := \mathscr{A}^{k^*}$ can be done in polynomial time.

In case of partial interchangeability, the cost function has to be modified slightly:

$$c(e) = \begin{cases} -1, & \text{if } e \in A_1 \setminus X_1, \\ 1, & \text{if } e \in A_2 \setminus X_2, \\ -m, & \text{if } e \in X_1, \\ m, & \text{if } e \in X_2, \\ 0, & \text{if } e \in E^+ \cup E^- \cup \{s's, ts'\}, \end{cases}$$

where a large cost m or $-m$ is assigned to the arcs on which the equipment cannot be changed. If for some $0 \leq k \leq K$, the cost of f_k, $c(f_k) \geq m$, then the equipment on some of the fixed arcs has been changed, and thus the corresponding $\mathscr{A}^k$ is not feasible. Let $\hat{T} = \{0 \leq k \leq K : I(\mathscr{A}^k) \leq K,\ c(f_k) < m\}$. If $\hat{T} = \emptyset$, then the desired assignment function $\mathscr{A}'$ does not exist, otherwise, let $\hat{k}^* = \text{argmin}_{k \in \hat{T}} \|\mathscr{A}^k - \mathscr{A}_0\|$, and $\mathscr{A}' := \mathscr{A}^{\hat{k}^*}$ is the desired assignment function.

5.2.4 Three-equipment type networks

We call a network N balanced if for any vertex $v \in V$, $\sigma_v^+ = \sigma_v^-$. The following proposition characterizes the difficulty of finding an assignment function $\mathscr{A}$ such that $I(\mathscr{A}) = 0$ in a balanced three-equipment network with partial interchangeability.

Proposition 5.2.1. *The problem of deciding whether there exists an assignment $\mathscr{A}$ such that $I(\mathscr{A}) = 0$ for a balanced network N with three equipment types and a set $S \subseteq A$ of arcs on which the equipment type cannot be changed is NP-complete.*

Proof. Transformation from SIMPLE D2CIF, which is known to be NP-complete (see [39]).

SIMPLE D2CIF: Given a directed graph $D = (V, A)$, with two source vertices $s_1, s_2 \in V$, two sink vertices $t_1, t_2 \in V$, each arc $a \in A$ having capacity one, and two non-negative

integers R_1 and R_2. Do there exist two flow functions f_1 and f_2, both $A \to \mathbb{N}$, such that f_1 sends R_1 units of flow from s_1 to t_1 and f_2 sends R_2 units of flow from s_2 to t_2?

Without loss of generality, we assume $\sigma^-_{s_1} = \sigma^-_{s_2} = \sigma^+_{t_1} = \sigma^+_{t_2} = 0$ for D. We construct $D' = (V', A')$ as follows. Add a vertex s to V. For each $v \in V$, if $\sigma^+_v > \sigma^-_v$, add $\sigma^+_v - \sigma^-_v$ parallel arcs from s to v and if $\sigma^+_v < \sigma^-_v$, add $\sigma^-_v - \sigma^+_v$ parallel arcs from v to s. For $v \in V$, let A_{vs} and A_{sv} denote the set of parallel arcs from v to s, and from s to v, respectively.

Let the three equipment types be Q_1, Q_2 and Q_3. Let $\hat{A}_{ss_1} \subseteq A_{ss_1}$, $\hat{A}_{ss_2} \subseteq A_{ss_2}$, $\hat{A}_{t_1s} \subseteq A_{t_1s}$, and $\hat{A}_{t_2s} \subseteq A_{t_2s}$ with $|\hat{A}_{ss_1}| = |\hat{A}_{t_1s}| = R_1$, $|\hat{A}_{ss_2}| = |\hat{A}_{t_2s}| = R_2$. In the initial equipment assignment, let the equipment assigned to arcs in $\hat{A}_{ss_1}$ and $\hat{A}_{t_1s}$ be Q_1, in $\hat{A}_{ss_2}$ and $\hat{A}_{t_2s}$ be Q_2, and in $A_{ss_1} \setminus \hat{A}_{ss_1}$, $A_{ss_2} \setminus \hat{A}_{ss_2}$, $A_{t_1s} \setminus \hat{A}_{t_1s}$, $A_{t_2s} \setminus \hat{A}_{t_2s}$, and in A_{sv}, and A_{vs} for $v \in V \setminus \{s_1, s_2, t_1, t_2\}$ be Q_3. Furthermore, let these sets be such that the equipment type cannot be changed. For convenience, let Y_1 and Y_2 denote the sets of arcs with equipment type Q_1 and Q_2, respectively, for which the equipment type cannot be changed.

For a YES-instance of SIMPLE D2CIF, we have the required flow functions f_1 and f_2. We may assume f_1 does not send any flow from s_2 to t_2 and f_2 does not send any flow from s_1 to t_1 (otherwise, we can simply delete those flows from f_1 and f_2). Zero imbalance can be achieved by $\mathscr{A} : A' \to \{Q_1, Q_2, Q_3\}$ defined as follows:

$$\mathscr{A}(a') = \begin{cases} Q_1, & a' \in \{a \in A : f_1(a) = 1\} \cup Y_1, \\ Q_2, & a' \in \{a \in A : f_2(a) = 1\} \cup Y_2, \\ Q_3, & \text{otherwise.} \end{cases}$$

For equipment type Q_1, since f_1 is a flow, by conservation of flow for $v \in V \setminus \{s_1, s_2, t_1, t_2\}$, Q_1 is balanced. By the definition of $\mathscr{A}$, it is also balanced for $\{s_1, s_2, t_1, t_2, s\}$. Similarly, equipment type Q_2 is balanced. Since for all $v \in V'$, $\sigma^+(v) = \sigma^-(v)$, equipment type Q_3 is also balanced, and, thus, the imbalance is zero.

Now suppose there exists an assignment $\mathscr{A}$ for D' that achieves zero imbalance, then the corresponding instance of SIMPLE D2CIF is YES-instance, because the two flow functions f_1 and f_2 can be constructed as follows:

$$f_1(a) = \begin{cases} 1, & a \in \{a \in A : \mathscr{A}(a) = Q_1\}, \\ 0, & \text{otherwise,} \end{cases}$$

and

$$f_2(a) = \begin{cases} 1, & a \in \{a \in A : \mathscr{A}(a) = Q_2\}, \\ 0, & \text{otherwise.} \end{cases}$$

Since $M(D', \mathscr{A}) = 0$, for all $v \in V \setminus \{s_1, s_2, t_1, t_2\}$, the imbalance for Q_1 and Q_2 is zero, and, thus, there is conservation of flow. For s_1, since on R_1 arcs from s to s_1 the equipment type is Q_1 and on the remaining arcs from s to s_1 the equipment type is Q_3, and the equipment type on these arcs cannot be changed, we have $\sum_{a \in A \,:\, a \ni s_1} f_1(a) = R_1$. Similarly, we have $\sum_{a \in A \,:\, a \ni t_1} f_1(a) = R_1$, $\sum_{a \in A \,:\, a \ni s_2} f_2(a) = R_2$, and $\sum_{a \in A \,:\, a \ni t_2} f_2(a) = R_2$. Therefore, f_1 and f_2 are the desired flow functions. ■

Proposition 5.2.1 implies that in the case of partial interchangeability, both the problem of finding the minimum imbalance and the problem of finding the minimum number of changes required to reach minimum imbalance are NP-hard. Next, we show that the problem of finding the minimum number of changes required to reach minimum imbalance is also NP-hard for the case of full interchangeability.

Proposition 5.2.2. *The problem of deciding whether there exists an assignment $\mathscr{A}$ such that $I(\mathscr{A}) = I^*$ and $\|\mathscr{A} - \mathscr{A}_0\| \leq K$, for a given $K \in \mathbb{Z}_{\geq 0}$ and a network N with three equipment types and full interchangeability is NP-complete.*

Proof. For convenience, the decision problem in Proposition 5.2.1 is referred to as 3EPI (3-equipment with partial interchangeability). We prove the proposition by providing a transformation from 3EPI.

Given a balanced network $D = (V, A)$, three disjoint subsets Y_1, Y_2, and Y_3 of A such that the equipment types on arcs in Y_1, Y_2, and Y_3, are Q_1, Q_2, and Q_3, respectively, and cannot be changed, and $K \in \mathbb{Z}_{\geq 0}$, we construct a directed graph $D' = (V', A')$ as follows. For all arcs $a = (u, v) \in Y_i$, $i = 1, 2, 3$, replace a by a directed path P_a of length $m = |A|$ from u to v by adding $m - 1$ intermediate vertices and $m - 1$ arcs. Note that $|V'| \leq mn$ and $|A'| \leq m^2$ and the size of D' is polynomial in the size of D. Let the initial assignment for $a' \in A'$ be

$$\mathscr{A}_0(a') := \begin{cases} Q_1, & \text{if } a' \in P_a \text{ for } a \in Y_1, \\ Q_2, & \text{if } a' \in P_a \text{ for } a \in Y_2, \\ Q_3, & \text{otherwise.} \end{cases}$$

By construction, D' is also a balanced network. Let $\hat{I}^*$ be the minimum imbalance of D' and $\hat{I}(\mathscr{A})$ be the imbalance of an assignment of D'. Observe that in any assignment $\mathscr{A}'$ with $\hat{I}(\mathscr{A}) = 0$ and $\|\mathscr{A}' - \mathscr{A}_0\| \leq m - 1$, the equipment type on arcs $a' \in p_a$ for $a \in Y_1 \cup Y_2 \cup Y_3$ cannot have changed. Therefore, if we can find an $\mathscr{A}$ with $I(\mathscr{A}) = 0$ and $\|\mathscr{A}' - \mathscr{A}_0\| \leq m - 1$, then the instance of 3EPI is a YES-instance, verified by assignment

$$\mathscr{A}(a) := \begin{cases} Q_1, & \text{if } \mathscr{A}'(a) = Q_1, \\ Q_2, & \text{if } \mathscr{A}'(a) = Q_2, \\ Q_3, & \text{otherwise.} \end{cases}$$

If, on the other hand, for all $\mathscr{A}$ with $\hat{I}(\mathscr{A}) = 0$, we have $\|\mathscr{A}' - \mathscr{A}_0\| \geq m$, then the instance of 3EPI is a NO-instance. ■

5.3 Final Remarks

Striving for equipment balance, i.e., seeking to have the same equipment at a facility at the end of the week as at the start of the week, ignores what happens during the week, and does not account for seasonal changes in package flows. A natural future research direction,

therefore, is inventory-aware equipment management, in which time is modeled explicitly, e.g., days for planning periods of one or more weeks, and weeks for planning periods of one or more quarters.

CHAPTER 6

CONCLUDING REMARKS AND FUTURE DIRECTIONS

Serving as a tool to obtain a solution efficiently, optimization (mainly continuous) has always been an indispensable part of machine learning (ML). Various optimization algorithms have been developed and/or tailored to different ML models and have facilitated the tremendous success of ML, such as the alternating direction method of multipliers (ADMM) for compressed sensing, full gradient descent (GD) for logistic regression, stochastic gradient descent (SGD) and its variants for deep learning, to name a few. Encouraged by the promise of ML in helping to make smarter decisions given enough history information, there is a trend to apply ML to improve optimization (mainly discrete) algorithms. Although MIP solvers has achieved a speedup of more than 800 billions over the past 25 years (1991-2015), as reported in [21], there are still numerous problems faced by the industry that are beyond their solution capacities. We have demonstrated that ML combined with domain knowledge can result in noticeable improvement in performance, which is a direction deserving more research effort.

An efficient LP solver is of the essence for every MIP solver. Meanwhile, solving LPs itself sees tons of applications in reality. To solve LPs by simplex, longstanding questions are how to select the initial basis to start and how to pivot, i.e., choose entering and exiting variables at each step. If an interior point method is used, to return a basis for subsequent computation, an initial basis is also required for the crossover stage. To solve MIPs, how to pick effective primal heuristics, branch, add cutting planes, or identify potential decomposable structure to address problems of large size are all open questions. Currently, those important decisions in MIP solvers are made by some predefined rules that are mostly summarized manually through extensive numerical experiments. We believe most of the decisions can be made more intelligently with the aid of ML techniques.

In addition to branching strategies (see Chapter 3), cutting plane selection (see [68]) can also be improved by ML. Nevertheless, most of current research tries to enhance a simple component at a time without paying enough attention to the complex interplay of different decisions within MIP solvers. It will be interesting to investigate if a certain combinations of a branching strategies, cutting planes and primal heuristics can deliver better overall performance.

Appendices

APPENDIX A

DETAILED RESULTS OF NUMERICAL EXPERIMENTS

Table A.1: Set covering problems: comparison on number of nodes explored.

Name	Number of nodes explored					Top 5
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
gte_3_00	418	60	231	305	390	62.19
gte_3_01	104	54	52	25	23	65.00
gte_3_02	1,121	91	297	529	415	64.45
gte_3_03	634	82	296	455	352	64.01
gte_3_04	32	14	35	46	78	54.55
gte_3_05	864	257	360	838	904	66.85
gte_3_06	430	252	353	561	839	63.51
gte_3_07	1,084	178	323	786	629	63.92
gte_3_08	578	72	255	186	267	68.00
gte_3_09	1,217	410	716	888	906	65.41
gte_3_10	306	170	353	476	357	64.54
gte_3_11	486	174	181	396	217	59.00
gte_3_12	233	105	194	450	340	63.58
gte_3_13	655	215	291	332	611	70.37
gte_3_14	161	37	70	146	137	59.14
gte_3_15	966	145	367	917	684	68.20
gte_3_16	4,426	619	1,965	2,673	1,657	65.61
gte_3_17	445	115	323	530	173	59.88
gte_3_18	2,774	330	1,154	1,178	845	63.18
gte_3_19	71	35	46	239	53	58.95
total	17,005	3,415	7,862	11,956	9,877	63.52
savings	—	79.92%	53.77%	29.69%	41.92%	
gte_4_00	4,684	1,178	1,573	1,590	2,969	68.68
gte_4_01	22,598	6,252	9,723	15,603	10,938	68.44
gte_4_02	3,317	616	1,282	3,375	1,154	59.72
gte_4_03	1,372	358	1,454	2,139	1,227	66.05
gte_4_04	1,067	405	378	770	750	63.11
gte_4_05	2,563	615	1,136	1,739	1,538	66.77
gte_4_06	1,471	578	851	1,216	930	67.82
gte_4_07	1,680	480	678	1,148	885	63.08
gte_4_08	1,458	382	1,101	1,472	831	66.77
gte_4_09	4,427	710	1,751	3,278	2,817	64.84
gte_4_10	1,830	393	755	3,082	1,876	63.84
gte_4_11	861	412	538	955	691	63.19
gte_4_12	2,704	624	1,474	2,272	1,772	66.74
gte_4_13	515	112	425	294	269	61.44

Continued on next page

Table A.1 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
gte_4_14	1,358	312	639	1,166	339	58.49
gte_4_15	2,958	621	837	1,623	1,685	61.47
gte_4_16	12,018	2,981	6,680	3,502	8,519	69.53
gte_4_17	795	232	454	665	1,012	61.81
gte_4_18	8,964	1,637	3,764	4,351	4,350	68.31
gte_4_19	6,990	1,775	3,315	4,801	4,244	66.85
total	83,630	20,673	38,808	55,041	48,796	64.85
savings	—	75.28%	53.60%	34.19%	41.65%	
gte_5_00	17,339	6,521	16,502	16,765	9,839	73.14
gte_5_01	9,400	1,411	5,068	6,317	4,760	62.91
gte_5_02	87,696	18,804	32,903	49,048	46,740	64.94
gte_5_03	17,853	7,821	19,960	24,871	10,759	65.38
gte_5_04	38,278	11,677	30,433	24,263	12,081	65.78
gte_5_05	140,687	14,987	22,697	58,074	29,957	69.49
gte_5_06	29,724	5,346	8,663	10,976	11,331	62.76
gte_5_07	49,893	17,024	23,654	47,338	21,705	62.39
gte_5_08	28,051	5,862	9,821	15,874	16,705	60.00
gte_5_09	11,434	3,412	13,727	18,015	12,669	63.23
gte_5_10	6,503	1,206	2,260	5,262	2,726	70.33
gte_5_11	52,553	13,678	28,190	31,198	30,894	65.42
gte_5_12	17,190	3,739	10,566	8,606	6,085	68.00
gte_5_13	55,147	11,731	17,182	25,009	21,736	61.24
gte_5_14	6,846	3,645	5,539	2,611	4,066	66.09
gte_5_15	3,286	797	1,198	1,604	1,421	63.52
gte_5_16	13,255	5,497	8,534	3,072	6,962	63.53
gte_5_17	73,865	10,788	17,989	40,890	21,206	66.53
gte_5_18	8,741	2,244	9,455	8,932	7,323	67.09
gte_5_19	9,856	1,303	5,512	8,880	1,603	69.80
total	677,597	147,493	289,853	407,605	280,568	65.58
savings	—	78.23%	57.22%	39.85%	58.59%	
gte_6_00	66,359	8,819	18,722	19,969	21,074	66.91
gte_6_01	29,122	6,182	21,912	12,243	7,903	64.68
gte_6_02	377,863	109,474	423,699	139,043	156,993	67.62
gte_6_03	467,523	45,594	89,205	103,326	109,543	65.68
gte_6_04	206,069	26,132	56,984	59,855	54,365	67.62
gte_6_05	1,000,000	447,398	1,000,000	344,568	425,109	69.03
gte_6_06	651,731	132,161	1,000,000	611,423	246,961	68.58
gte_6_07	181,479	55,242	91,392	62,354	72,002	66.54
gte_6_08	29,434	5,869	33,537	14,570	13,467	65.97
gte_6_09	1,000,000	144,436	354,465	342,833	363,778	66.99
gte_6_10	223,047	18,220	27,699	38,824	45,479	66.00
gte_6_11	466,738	164,218	169,450	185,680	230,245	69.06
gte_6_12	216,485	30,796	105,397	159,153	146,332	63.88
gte_6_13	73,490	17,475	44,529	36,166	31,223	67.83
gte_6_14	107,282	21,774	40,863	48,613	43,151	64.86

Continued on next page

Table A.1 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
gte_6_15	473,913	130,895	399,090	521,841	194,681	67.85
gte_6_16	89,671	9,766	52,555	42,050	16,391	66.43
gte_6_17	37,587	10,888	24,557	30,688	9,420	65.66
gte_6_18	101,833	21,361	11,229	23,454	8,693	69.66
gte_6_19	242,965	34,645	156,672	143,198	78,823	64.98
total	6,042,591	1,441,345	4,121,957	2,939,851	2,275,633	66.79
savings	—	76.15%	31.78%	51.35%	62.34%	
scpnre1	46,901	7,598	15,256	17,697	16,998	68.84
scpnre2	179,779	58,980	88,501	138,099	38,463	67.30
scpnre3	37,324	7,318	23,163	15,390	28,189	67.96
scpnre4	105,694	15,116	17,215	15,568	35,678	64.28
scpnre5	9,640	2,212	3,683	5,943	5,085	68.17
total	379,338	91,224	147,818	192,697	124,413	67.31
savings	—	75.95%	61.03%	49.20%	67.20%	
scpnrf1	48,144	4,763	27,985	13,269	13,706	72.72
scpnrf2	18,950	3,488	12,980	8,449	7,787	71.26
scpnrf3	12,466	6,288	22,055	3,533	4,687	73.35
scpnrf4	65,506	12,096	50,019	76,450	34,927	73.37
scpnrf5	357,506	22,718	171,234	29,119	126,862	72.60
total	502,572	49,353	284,273	130,820	187,969	72.66
savings	—	90.18%	43.44%	73.97%	62.60%	
Total	7,702,733	1,753,503	4,890,571	3,737,970	2,927,256	65.72
Savings	—	77.24%	36.51%	51.47%	62.00%	

Table A.2: Set covering problems: comparison on run time.

Name	Run time (seconds)				
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
gte_3_00	0.79	1.51	6.07	0.59	0.75
gte_3_01	0.48	1.67	2.53	0.24	0.24
gte_3_02	1.47	2.13	9.66	0.86	0.83
gte_3_03	0.94	1.64	7.69	0.71	0.64
gte_3_04	0.37	0.64	1.91	0.27	0.32
gte_3_05	1.10	3.64	10.41	1.12	1.32
gte_3_06	0.87	4.46	10.21	0.91	1.28
gte_3_07	1.22	3.11	8.08	1.05	0.98
gte_3_08	0.86	1.71	6.72	0.40	0.51
gte_3_09	1.62	7.23	17.10	1.20	1.40
gte_3_10	0.78	3.27	10.29	0.82	0.72
gte_3_11	0.74	2.85	6.11	0.63	0.48
gte_3_12	0.69	2.26	6.18	0.76	0.74
gte_3_13	1.02	4.47	10.38	0.62	1.09
gte_3_14	0.55	1.03	2.90	0.33	0.35
gte_3_15	1.44	3.41	11.52	1.32	1.18

Continued on next page

Table A.2 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
gte_3_16	4.86	11.10	51.11	3.59	2.54
gte_3_17	0.98	2.84	10.58	0.88	0.48
gte_3_18	2.79	5.79	30.58	1.62	1.37
gte_3_19	0.44	1.19	2.50	0.50	0.28
total	24.01	65.95	222.53	18.42	17.5
savings	—	-174.68%	-826.82%	23.28%	27.11%
gte_4_00	6.41	28.81	50.65	3.15	5.73
gte_4_01	27.41	149.12	279.17	25.08	18.94
gte_4_02	4.28	15.33	40.76	5.24	2.28
gte_4_03	2.58	9.80	50.81	4.11	2.70
gte_4_04	2.01	11.33	14.33	1.51	1.70
gte_4_05	4.22	14.20	32.12	3.22	3.13
gte_4_06	2.64	14.38	34.36	2.32	2.03
gte_4_07	2.69	11.41	19.54	2.01	1.72
gte_4_08	2.66	10.60	40.62	2.76	1.97
gte_4_09	6.48	17.05	51.27	5.58	5.26
gte_4_10	3.17	10.52	25.91	4.66	3.69
gte_4_11	1.99	12.03	16.77	2.09	1.69
gte_4_12	4.48	15.24	51.61	3.86	3.48
gte_4_13	1.37	3.61	16.14	0.80	0.84
gte_4_14	2.32	8.12	21.42	2.20	1.01
gte_4_15	3.77	11.77	26.40	2.53	2.97
gte_4_16	17.10	75.76	237.95	6.68	16.88
gte_4_17	1.79	6.44	15.76	1.44	2.20
gte_4_18	10.47	32.67	107.58	6.62	7.86
gte_4_19	9.93	39.50	92.62	7.65	7.70
total	117.77	497.69	1,225.79	93.51	93.78
savings	—	-322.59%	-940.83%	20.60%	20.37%
gte_5_00	30.32	190.40	703.09	34.16	23.95
gte_5_01	17.09	43.97	226.96	13.58	11.95
gte_5_02	131.71	518.60	1159.14	91.51	107.04
gte_5_03	34.21	236.88	932.69	48.91	27.37
gte_5_04	62.05	348.61	1363.74	49.88	30.59
gte_5_05	237.66	470.99	1074.98	124.29	78.24
gte_5_06	49.48	169.64	318.61	22.51	28.62
gte_5_07	76.46	431.79	948.45	80.55	47.52
gte_5_08	37.73	146.81	338.95	27.33	36.32
gte_5_09	19.35	100.80	542.82	33.47	28.07
gte_5_10	12.63	38.39	87.97	12.30	7.14
gte_5_11	78.30	358.78	1044.39	54.40	66.01
gte_5_12	29.84	124.04	468.23	18.96	15.34
gte_5_13	78.74	321.40	641.10	44.29	46.54
gte_5_14	12.40	105.82	226.66	5.63	9.17
gte_5_15	6.16	25.83	50.73	3.74	4.39
gte_5_16	21.12	156.02	335.68	6.19	15.66

Continued on next page

Table A.2 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
gte_5_17	121.01	346.04	747.71	83.95	53.08
gte_5_18	15.44	69.42	406.65	17.82	16.91
gte_5_19	16.21	44.63	298.39	18.60	4.53
total	1,087.91	4,248.86	11,916.94	792.07	658.44
savings	—	-290.55%	-995.40%	27.19%	39.48%
gte_6_00	132.14	315.93	962.42	48.72	58.13
gte_6_01	59.20	227.67	1217.78	31.27	24.83
gte_6_02	777.32	4176.73	25036.77	348.78	479.65
gte_6_03	841.15	1562.97	4318.07	224.58	318.98
gte_6_04	437.85	1061.35	2887.39	171.31	179.88
gte_6_05	2330.47	19526.79	58224.99	994.56	1488.88
gte_6_06	1495.31	5201.63	54520.36	1627.42	788.34
gte_6_07	408.38	2141.05	5277.72	161.53	222.15
gte_6_08	62.30	214.40	1962.26	37.21	41.46
gte_6_09	1967.79	5240.57	18402.67	811.10	1051.40
gte_6_10	431.44	735.68	1793.04	100.85	137.68
gte_6_11	956.12	6383.23	9553.25	482.02	719.60
gte_6_12	392.46	1060.42	5214.38	382.65	396.93
gte_6_13	165.85	735.68	2990.66	95.06	103.08
gte_6_14	201.62	737.71	1898.54	111.96	121.69
gte_6_15	1017.94	5277.22	24016.23	1361.13	626.10
gte_6_16	178.86	382.84	2822.59	106.89	51.70
gte_6_17	79.39	439.08	1393.88	81.79	31.63
gte_6_18	201.73	894.95	687.32	69.96	29.00
gte_6_19	470.74	1257.28	8705.86	336.08	228.51
total	12,608.06	57,573.18	231,886.18	7,584.87	7,099.62
savings	—	-356.64%	-1739.19%	39.84%	43.69%
scpnre1	70.66	206.19	578.41	30.74	37.02
scpnre2	269.91	1742.58	3913.00	268.99	95.30
scpnre3	59.85	221.30	886.89	32.49	68.71
scpnre4	148.13	411.53	728.08	29.45	80.86
scpnre5	17.57	66.84	126.78	12.39	12.54
total	566.12	2,648.44	6,233.16	374.06	294.43
savings	—	-367.82%	-1001.03%	33.93%	47.99%
scpnrf1	70.92	151.63	5024.09	27.28	30.39
scpnrf2	29.37	119.60	951.49	17.95	19.20
scpnrf3	20.19	228.63	4579.44	8.40	11.70
scpnrf4	103.42	397.93	3568.10	144.56	85.48
scpnrf5	435.84	744.83	11601.08	61.33	279.88
total	659.74	1,642.62	25,724.2	259.52	426.65
savings	—	-148.98%	-3799.14%	60.66%	35.33%
Total	15,063.61	66,676.74	277,208.8	9,122.45	8,590.42
Savings	—	-342.63%	-1740.25%	39.44%	42.97%

Table A.3: Set packing problems: comparison on number of nodes explored.

Name	Number of nodes explored					Top 5
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
gte_100_00	189	38	153	168	201	40.26
gte_100_01	1,723	253	799	781	956	60.13
gte_100_02	2,900	530	1,156	1,132	1,809	62.65
gte_100_03	1,190	132	343	670	660	58.49
gte_100_04	706	96	298	318	481	54.00
gte_100_05	1,469	88	172	925	1,002	59.30
gte_100_06	484	170	350	239	172	55.00
gte_100_07	1,470	316	650	911	1,412	56.08
gte_100_08	348	67	235	171	415	58.94
gte_100_09	2,968	274	899	1,745	1,848	53.50
gte_100_10	582	100	322	267	298	57.14
gte_100_11	6,124	1,047	2,492	3,533	4,467	58.79
gte_100_12	5,259	757	1,791	3,059	2,854	61.90
gte_100_13	1,179	208	511	511	623	57.75
gte_100_14	1,590	206	512	721	865	58.94
gte_100_15	518	118	192	323	451	60.73
gte_100_16	1,994	413	801	1,367	1,013	57.26
gte_100_17	1,201	250	280	379	1,015	58.41
gte_100_18	2,403	301	1,014	511	872	50.40
gte_100_19	1,192	240	378	277	245	65.19
total	35,489	5,604	13,348	18,008	21,659	57.24
savings	—	84.21%	62.39%	49.26%	38.97%	
gte_150_00	6,680	1,591	3,165	4,015	2,558	71.71
gte_150_01	8,801	1,320	3,101	5,119	3,908	67.70
gte_150_02	8,130	1,821	4,323	4,376	4,928	65.52
gte_150_03	12,638	3,191	4,032	5,563	5,988	64.99
gte_150_04	10,656	2,154	4,249	6,460	4,654	67.53
gte_150_05	6,735	1,634	2,775	5,156	4,369	69.46
gte_150_06	6,969	1,281	2,750	3,215	3,557	67.76
gte_150_07	10,655	2,616	5,174	4,845	6,340	69.75
gte_150_08	12,290	3,753	4,726	7,140	6,130	65.51
gte_150_09	7,976	2,754	3,863	5,374	7,357	65.08
gte_150_10	11,852	2,638	5,793	8,184	6,676	67.26
gte_150_11	4,304	1,234	1,530	1,205	2,821	61.09
gte_150_12	11,901	2,545	4,103	4,446	5,418	69.43
gte_150_13	11,480	2,539	4,109	5,648	5,013	66.59
gte_150_14	18,107	4,081	7,284	9,817	9,344	69.12
gte_150_15	6,559	1,797	3,208	3,551	2,850	70.86
gte_150_16	9,655	1,710	4,745	5,812	5,499	63.83
gte_150_17	7,207	849	2,315	4,934	2,932	69.68
gte_150_18	5,236	1,916	2,015	3,066	2,091	68.86
gte_150_19	9,911	2,930	6,029	7,249	7,893	68.84
total	187,742	44,354	79,289	105,175	100,326	67.53

Continued on next page

Table A.3 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
savings	—	76.38%	57.77%	43.98%	46.56%	
gte_200_00	8,489	3,010	4,901	5,484	5,151	76.74
gte_200_01	10,733	3,133	5,228	7,187	6,575	75.30
gte_200_02	9,123	2,662	3,977	5,251	4,134	78.37
gte_200_03	13,039	3,904	5,971	6,740	6,417	78.40
gte_200_04	8,176	3,041	5,569	5,205	6,808	75.91
gte_200_05	9,829	2,644	3,087	6,062	2,788	76.72
gte_200_06	8,809	3,388	4,110	3,578	5,822	77.59
gte_200_07	11,079	3,659	5,640	5,615	6,691	79.98
gte_200_08	15,090	4,777	7,825	7,757	7,485	77.09
gte_200_09	10,496	3,990	5,288	8,757	7,928	77.45
gte_200_10	8,170	3,233	3,332	4,919	5,392	76.37
gte_200_11	9,121	2,849	4,774	5,637	5,679	76.62
gte_200_12	14,182	5,305	7,777	8,328	10,274	78.68
gte_200_13	10,436	4,363	5,622	6,374	5,329	79.71
gte_200_14	10,243	3,297	4,358	6,600	4,739	77.92
gte_200_15	11,544	3,365	5,221	7,116	6,654	78.69
gte_200_16	13,786	5,226	7,431	10,150	7,576	76.02
gte_200_17	11,933	3,433	5,281	8,601	5,877	77.20
gte_200_18	10,232	2,537	5,064	7,620	6,052	76.73
gte_200_19	13,069	4,375	7,552	7,958	7,281	77.18
total	217,579	72,191	108,008	134,939	124,652	77.43
savings	—	66.82%	50.36%	37.98%	42.71%	
gte_250_00	15,812	5,145	7,193	8,770	9,624	82.64
gte_250_01	22,469	6,643	11,506	11,993	11,738	81.99
gte_250_02	22,459	4,641	9,609	9,453	12,082	81.64
gte_250_03	16,449	4,323	9,187	10,610	11,564	82.15
gte_250_04	14,404	3,479	7,011	8,878	7,726	81.68
gte_250_05	14,444	4,928	4,771	8,020	8,309	82.02
gte_250_06	16,025	5,043	8,139	9,922	9,822	80.56
gte_250_07	24,003	6,505	11,038	12,620	15,005	80.81
gte_250_08	18,732	5,202	9,049	11,654	10,543	81.05
gte_250_09	14,579	4,972	8,481	10,253	9,945	82.94
gte_250_10	18,625	5,000	8,509	11,284	9,048	81.75
gte_250_11	16,893	4,844	8,792	11,751	10,661	81.63
gte_250_12	15,629	3,960	7,533	7,301	8,330	81.92
gte_250_13	17,520	5,329	7,464	8,769	9,498	81.89
gte_250_14	14,158	5,670	9,644	9,137	9,689	82.96
gte_250_15	17,957	4,363	9,176	8,767	9,147	82.87
gte_250_16	12,834	4,574	6,374	8,261	8,392	81.87
gte_250_17	15,836	5,637	6,273	7,947	8,657	81.15
gte_250_18	23,504	8,849	10,651	14,705	14,443	80.51
gte_250_19	12,007	4,374	5,597	7,287	8,865	81.41
total	344,339	103,481	165,997	197,382	203,088	81.77
savings	—	69.95%	51.79%	42.68%	41.02%	

Continued on next page

Table A.3 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
gte_300_00	17,131	4,534	8,256	9,213	9,019	83.90
gte_300_01	21,740	5,981	12,610	11,740	10,962	83.68
gte_300_02	18,146	5,106	8,651	9,848	8,796	83.84
gte_300_03	24,414	4,678	9,060	10,186	8,655	83.80
gte_300_04	14,361	6,237	8,436	8,010	8,140	84.01
gte_300_05	15,366	6,024	4,775	7,990	8,449	84.28
gte_300_06	14,316	5,219	7,992	8,411	8,279	83.47
gte_300_07	14,287	4,730	8,240	7,269	8,390	84.60
gte_300_08	14,077	4,383	8,423	9,665	9,659	83.26
gte_300_09	17,571	4,350	7,462	8,640	8,769	84.26
gte_300_10	20,705	5,036	10,170	9,943	10,497	83.90
gte_300_11	15,205	5,441	5,298	7,500	6,446	84.23
gte_300_12	18,644	5,267	8,454	9,332	9,757	83.00
gte_300_13	16,874	4,768	7,493	8,594	8,972	83.74
gte_300_14	16,711	3,661	6,542	7,378	6,864	83.86
gte_300_15	18,942	4,455	8,246	9,651	9,644	84.51
gte_300_16	17,063	5,540	9,727	10,777	10,869	84.18
gte_300_17	11,867	3,738	5,952	5,443	8,084	83.99
gte_300_18	23,093	8,650	13,765	14,405	13,768	83.77
gte_300_19	14,950	6,478	8,523	10,145	9,078	84.02
total	345,463	104,276	168,075	184,140	183,097	83.91
savings	—	69.82%	51.35%	46.70%	47.00%	
Total	1,130,612	329,906	534,717	639,644	632,822	73.58
Savings	—	70.82%	52.71%	43.42%	44.03%	

Table A.4: Set packing problems: comparison on run time.

Name	Run time (seconds)				
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
gte_100_00	0.38	0.47	2.13	0.23	0.26
gte_100_01	1.13	2.83	7.26	0.60	0.81
gte_100_02	1.81	5.83	12.35	1.02	1.65
gte_100_03	0.99	1.77	3.61	0.64	0.75
gte_100_04	0.64	1.18	3.18	0.31	0.49
gte_100_05	0.96	1.25	2.16	0.76	0.94
gte_100_06	0.59	2.08	4.38	0.34	0.29
gte_100_07	1.02	3.78	6.25	0.75	1.20
gte_100_08	0.47	0.95	2.98	0.25	0.47
gte_100_09	1.91	3.27	10.49	1.33	1.60
gte_100_10	0.79	1.54	4.35	0.40	0.45
gte_100_11	3.27	10.97	21.91	2.39	3.26
gte_100_12	2.97	8.24	16.24	2.08	2.15
gte_100_13	0.84	2.17	5.19	0.45	0.60
gte_100_14	1.07	2.44	5.09	0.59	0.82

Continued on next page

Table A.4 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
gte_100_15	0.51	1.48	2.31	0.31	0.46
gte_100_16	1.32	4.72	7.91	1.05	0.93
gte_100_17	1.04	3.32	3.36	0.46	1.04
gte_100_18	1.83	3.66	12.01	0.49	0.80
gte_100_19	0.90	2.65	4.38	0.30	0.29
total	24.44	64.6	137.54	14.75	19.26
savings	—	-164.32%	-462.77%	39.65%	21.19%
gte_150_00	8.20	35.64	84.57	6.31	6.26
gte_150_01	9.92	28.71	68.13	7.79	7.02
gte_150_02	10.27	34.37	85.19	6.88	7.82
gte_150_03	13.21	56.97	78.79	7.92	9.07
gte_150_04	10.52	39.47	77.46	8.78	7.72
gte_150_05	8.43	31.09	58.46	7.14	7.23
gte_150_06	8.92	25.86	63.41	5.87	6.36
gte_150_07	13.08	43.52	109.59	8.44	10.93
gte_150_08	13.03	63.24	82.28	9.69	11.55
gte_150_09	9.00	49.36	71.47	7.61	10.42
gte_150_10	13.69	51.29	109.84	15.39	10.53
gte_150_11	5.59	25.99	35.79	2.50	4.66
gte_150_12	13.57	46.53	82.65	7.42	9.05
gte_150_13	11.99	45.79	80.61	7.89	7.91
gte_150_14	19.33	65.93	114.71	13.14	13.83
gte_150_15	8.64	36.17	71.95	5.48	5.52
gte_150_16	11.43	32.29	84.78	8.32	8.96
gte_150_17	9.18	20.88	59.71	7.87	5.89
gte_150_18	7.47	38.05	47.76	5.47	4.09
gte_150_19	11.55	52.73	109.31	10.01	11.98
total	217.02	823.88	1,576.46	159.92	166.8
savings	—	-279.63%	-626.41%	26.31%	23.14%
gte_200_00	24.95	86.23	191.77	19.00	18.94
gte_200_01	28.02	89.05	185.89	22.86	22.03
gte_200_02	26.19	81.79	165.63	19.74	18.78
gte_200_03	31.27	100.25	213.22	22.24	21.90
gte_200_04	23.76	86.84	218.92	18.95	22.62
gte_200_05	24.62	84.89	133.17	19.79	12.20
gte_200_06	22.03	102.39	164.65	15.57	21.60
gte_200_07	30.65	103.23	209.58	22.04	26.50
gte_200_08	35.96	116.92	249.44	25.51	25.32
gte_200_09	27.49	102.43	187.50	23.46	23.82
gte_200_10	21.42	100.72	164.51	16.37	20.96
gte_200_11	28.12	84.40	190.16	19.09	22.87
gte_200_12	31.89	134.41	265.27	24.93	29.94
gte_200_13	28.95	127.09	221.79	23.15	21.54
gte_200_14	26.18	100.47	178.84	21.22	20.02
gte_200_15	30.97	98.42	224.94	24.27	23.95

Continued on next page

Table A.4 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
gte_200_16	28.77	116.21	251.70	25.27	23.28
gte_200_17	23.95	94.78	184.10	24.35	20.60
gte_200_18	26.01	72.67	189.18	22.55	21.60
gte_200_19	28.45	110.43	249.46	24.21	23.97
total	549.65	1,993.62	4,039.72	434.57	442.44
savings	—	-262.71%	-634.96%	20.94%	19.51%
gte_250_00	47.66	195.25	394.89	38.71	41.62
gte_250_01	55.41	217.20	538.20	48.29	47.72
gte_250_02	52.72	176.50	478.23	42.84	47.40
gte_250_03	47.62	168.36	480.91	38.50	44.15
gte_250_04	49.50	144.93	424.74	38.73	39.14
gte_250_05	43.30	189.73	333.68	34.88	39.41
gte_250_06	47.60	184.80	438.73	40.11	41.58
gte_250_07	56.38	218.05	540.38	46.00	53.92
gte_250_08	50.47	177.24	435.85	43.27	41.34
gte_250_09	47.99	181.98	458.01	43.27	43.61
gte_250_10	51.36	177.60	423.41	44.60	37.83
gte_250_11	54.61	187.84	524.91	48.84	50.00
gte_250_12	44.42	156.24	452.18	39.12	38.59
gte_250_13	52.36	191.69	463.42	43.78	45.67
gte_250_14	49.15	192.49	488.52	43.24	44.18
gte_250_15	51.66	169.80	499.42	41.55	44.25
gte_250_16	45.74	180.02	397.04	37.59	37.54
gte_250_17	48.78	199.58	386.09	38.15	38.31
gte_250_18	51.90	235.23	481.68	46.20	48.82
gte_250_19	42.45	169.95	375.72	36.18	37.26
total	991.08	3,714.48	9,016.01	833.85	862.34
savings	—	-274.79%	-809.72%	15.86%	12.99%
gte_300_00	85.18	294.45	878.00	76.21	78.56
gte_300_01	92.93	327.42	997.50	81.81	81.42
gte_300_02	89.85	328.07	950.80	81.13	81.52
gte_300_03	92.32	290.17	892.34	76.56	76.76
gte_300_04	81.74	329.00	911.35	76.91	77.74
gte_300_05	72.83	331.80	689.49	69.79	71.79
gte_300_06	80.73	323.52	871.74	73.62	75.48
gte_300_07	78.28	287.04	827.59	67.88	73.61
gte_300_08	83.46	289.92	874.43	76.14	78.14
gte_300_09	86.07	293.58	848.02	80.05	79.91
gte_300_10	80.47	291.72	895.09	71.75	75.73
gte_300_11	84.65	304.55	783.77	72.77	75.04
gte_300_12	86.95	303.04	831.93	73.86	73.87
gte_300_13	78.16	269.06	817.22	72.48	74.80
gte_300_14	84.33	272.25	866.60	71.76	72.25
gte_300_15	94.42	309.18	904.62	80.78	81.03
gte_300_16	80.48	305.77	860.75	74.93	79.27

Continued on next page

Table A.4 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
gte_300_17	81.02	289.59	834.77	70.19	82.54
gte_300_18	92.73	349.95	963.31	81.15	81.83
gte_300_19	75.55	314.30	883.65	81.52	77.43
total	1,682.15	6,104.38	17,382.97	1,511.29	1,548.72
savings	—	-262.89%	-933.38%	10.16%	7.93%
Total	3,464.34	12,700.96	32,152.7	2,954.38	3,039.56
Savings	—	-266.62%	-828.10%	14.72%	12.26%

Table A.5: 0-1 knapsack problems: comparison on number of nodes explored.

Name	Number of nodes explored					Top 1
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
gte_2_00	476	2,402	1,239	1,239	389	99.92
gte_2_01	1,285	2,302	1,212	1,178	6,446	99.58
gte_2_02	982	2,446	1,221	1,800	591	99.83
gte_2_03	1,506	3,202	1,996	1,996	621	99.95
gte_2_04	1,444	7,370	1,207	1,204	1,322	99.83
gte_2_05	1,518	24,807	1,223	1,174	614	99.83
gte_2_06	6,883	2,440	1,161	1,277	753	99.53
gte_2_07	2,171	4,902	1,183	1,210	506	99.75
gte_2_08	1,470	2,447	1,356	1,208	1,226	99.59
gte_2_09	1,220	2,432	1,440	1,073	694	98.60
gte_2_10	1,228	2,445	1,223	1,015	621	99.51
gte_2_11	4,885	2,450	1,226	1,193	632	99.75
gte_2_12	881	41,094	1,245	1,230	1,596	99.67
gte_2_13	902	2,541	1,758	1,201	614	99.83
gte_2_14	1,408	1,461	2,522	1,569	664	99.04
gte_2_15	1,363	3,178	1,180	892	555	99.55
gte_2_16	1,730	2,699	1,238	1,941	1,273	99.85
gte_2_17	1,161	5,736	1,144	914	613	99.78
gte_2_18	1,176	2,575	1,212	1,213	613	99.75
gte_2_19	124	2,869	1,233	1,335	713	99.32
total	33,813	121,798	27,219	25,862	21,056	99.62
savings	—	-260.21%	19.50%	23.51%	37.73%	
gte_3_00	13,421	16,601	2,196	1,586	896	91.99
gte_3_01	1,729	12,231	2,173	2,597	1,078	99.88
gte_3_02	2,249	3,646	1,821	1,822	1,116	99.84
gte_3_03	25,710	4,599	1,831	1,831	928	99.95
gte_3_04	251	2,724	1,815	1,814	927	99.83
gte_3_05	4,175	5,337	1,823	1,853	970	99.89
gte_3_06	747	3,738	1,686	1,686	930	99.94
gte_3_07	1,149	3,671	2,269	1,785	30,499	99.78
gte_3_08	1,844	57	1,834	1,788	922	99.89
gte_3_09	27,366	6,654	1,836	1,807	945	99.83

Continued on next page

Table A.5 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
gte_3_10	266	3,222	1,791	1,789	1,017	99.72
gte_3_11	1,941	3,348	1,769	1,429	875	99.72
gte_3_12	5,988	2,925	1,843	1,841	933	99.84
gte_3_13	6,527	11,325	1,852	1,852	1,779	99.89
gte_3_14	1,761	5,187	1,858	1,837	937	99.89
gte_3_15	1,319	4,358	1,322	1,323	836	99.77
gte_3_16	2,273	8,273	2,232	1,458	880	99.31
gte_3_17	1,724	3,570	1,785	894	666	93.29
gte_3_18	23,775	2,405	1,847	1,846	943	99.13
gte_3_19	144,930	9,877	1,838	3,534	1,968	99.89
total	269,145	113,748	37,421	36,372	50,045	99.06
savings	—	57.74%	86.10%	86.49%	81.41%	
gte_4_00	27,023	5,606	2,438	2,437	1,233	99.75
gte_4_01	2,468	4,924	2,437	2,437	1,243	99.96
gte_4_02	2,439	4,874	2,438	2,438	31,656	99.96
gte_4_03	3,990	4,899	2,451	2,451	2,936	99.92
gte_4_04	2,435	21,401	2,128	2,126	1,431	99.86
gte_4_05	2,428	8,231	2,425	2,427	1,257	99.84
gte_4_06	1,932	3,458	2,450	2,111	1,230	95.55
gte_4_07	2,428	4,854	2,428	2,428	1,227	99.96
gte_4_08	1,971	10,134	2,426	2,426	1,047	99.96
gte_4_09	2,690	5,056	2,452	801	670	97.50
gte_4_10	2,769	5,327	2,456	4,583	641	99.89
gte_4_11	2,146	3,710	2,402	2,403	1,215	99.88
gte_4_12	3,212	5,011	2,422	4,265	482	99.86
gte_4_13	2,672	43,403	4,414	4,314	1,181	99.95
gte_4_14	3,626	4,909	2,442	1,439	1,236	99.72
gte_4_15	2,452	4,362	3,957	3,277	1,651	98.72
gte_4_16	2,841	580	2,468	1,570	1,236	99.62
gte_4_17	3,294	5,013	2,459	3,786	1,026	98.15
gte_4_18	1,021	4,904	2,449	2,449	1,243	99.92
gte_4_19	54,569	2,838	2,194	2,425	1,207	99.92
total	128,406	153,494	51,736	52,593	55,048	99.39
savings	—	-19.54%	59.71%	59.04%	57.13%	
gte_5_00	4,015	13,846	3,018	5,005	1,535	99.94
gte_5_01	2,982	6,665	4,166	4,166	2,121	99.95
gte_5_02	3,035	21,897	3,034	3,034	1,548	99.87
gte_5_03	3,727	6,109	3,012	3,012	1,547	99.97
gte_5_04	2,676	5,294	4,323	3,063	1,976	99.93
gte_5_05	3,945	4,860	3,062	3,060	1,483	99.93
gte_5_06	3,218	6,086	3,039	3,039	1,451	99.93
gte_5_07	4,552	6,424	3,063	1,783	1,262	99.66
gte_5_08	3,191	6,608	3,073	2,269	1,436	99.74
gte_5_09	81,753	6,138	3,026	579	532	99.48
gte_5_10	168,757	5,024	3,058	4,646	1,574	99.89

Continued on next page

Table A.5 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
gte_5_11	74,604	6,652	4,634	4,791	1,618	99.94
gte_5_12	3,129	6,127	4,767	3,297	1,411	99.79
gte_5_13	3,097	6,164	3,082	1,698	1,558	91.81
gte_5_14	3,034	6,064	3,030	2,856	1,533	99.89
gte_5_15	955	5,048	2,968	2,980	2,461	99.87
gte_5_16	28,227	18,423	3,093	3,089	2,842	99.81
gte_5_17	2,699	5,938	5,864	6,110	3,211	99.93
gte_5_18	2,451	3,487	3,078	3,079	1,551	99.94
gte_5_19	1,538	4,827	3,087	3,637	1,569	99.86
total	401,585	151,681	69,477	65,193	34,219	99.46
savings	—	62.23%	82.70%	83.77%	91.48%	
gte_6_00	5,401	26,737	3,704	2,090	1,505	99.47
gte_6_01	3,615	6,800	5,000	6,052	2,367	99.92
gte_6_02	30,072	7,335	3,610	3,449	1,495	93.97
gte_6_03	2,092	7,127	3,685	3,620	4,424	99.94
gte_6_04	3,512	7,074	3,599	3,654	1,844	99.95
gte_6_05	1,634	6,501	3,676	5,277	1,855	99.89
gte_6_06	3,201	7,285	3,578	3,578	1,735	99.97
gte_6_07	4,562	5,563	6,351	6,353	1,881	99.95
gte_6_08	3,665	7,309	3,654	5,879	1,843	99.90
gte_6_09	3,654	7,304	3,653	3,653	4,026	99.97
gte_6_10	52,233	1,355	3,673	3,177	1,796	99.84
gte_6_11	5,613	7,153	3,654	3,657	2,625	99.67
gte_6_12	3,472	7,728	3,647	6,914	1,843	99.91
gte_6_13	4,680	133,091	3,624	4,668	2,438	99.94
gte_6_14	4,459	9,657	3,679	3,677	1,764	99.92
gte_6_15	5,734	7,337	3,669	3,667	1,551	99.86
gte_6_16	217,320	164,062	3,691	2,844	4,180	99.65
gte_6_17	3,703	7,392	5,243	6,997	1,783	99.93
gte_6_18	2,732	7,754	3,622	3,658	1,661	99.89
gte_6_19	41,969	7,415	3,696	3,695	2,787	99.95
total	403,323	441,979	78,708	86,559	45,403	99.57
savings	—	-9.58%	80.49%	78.54%	88.74%	
Total	1,236,272	982,700	264,561	266,579	205,771	99.42
Savings	—	20.51%	78.60%	78.44%	83.36%	

Table A.6: 0-1 knapsack problems: comparison on run time.

Name	Run time (seconds)				
	CPLEX-D	CPLEX-SB d	OUR-SB	LRN-SB	LRN-GSB
gte_2_00	0.32	0.78	2.13	1.04	0.43
gte_2_01	0.61	0.72	1.99	1.02	4.25
gte_2_02	0.65	0.69	1.96	1.29	0.61
gte_2_03	0.85	0.81	3.07	1.28	0.61

Continued on next page

Table A.6 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
gte_2_04	0.57	1.81	1.89	0.89	1.06
gte_2_05	0.73	6.55	1.92	0.96	0.62
gte_2_06	1.62	0.75	2.62	1.02	0.77
gte_2_07	0.74	1.94	2.40	0.90	0.52
gte_2_08	0.73	0.71	2.09	0.99	1.01
gte_2_09	0.68	0.71	2.33	0.98	0.74
gte_2_10	0.64	0.68	2.05	0.84	0.57
gte_2_11	1.17	0.72	2.01	1.03	0.65
gte_2_12	0.40	11.51	1.95	1.01	1.31
gte_2_13	0.43	1.28	3.26	1.07	0.65
gte_2_14	0.73	0.40	4.21	1.04	0.69
gte_2_15	0.76	1.05	2.42	0.72	0.58
gte_2_16	0.54	0.69	1.95	1.26	1.00
gte_2_17	0.47	1.66	2.36	0.74	0.65
gte_2_18	0.75	0.74	2.11	0.93	0.63
gte_2_19	0.07	0.76	2.07	0.95	0.72
total	13.46	34.96	46.79	19.96	18.07
savings	—	-159.73%	-247.62%	-48.29%	-34.25%
gte_3_00	3.81	6.88	15.72	2.02	1.40
gte_3_01	1.68	5.54	16.17	2.48	1.42
gte_3_02	1.49	1.40	14.04	2.04	1.52
gte_3_03	8.86	1.72	14.15	2.02	1.40
gte_3_04	0.19	1.12	14.62	2.26	1.42
gte_3_05	1.65	2.28	14.85	2.04	1.47
gte_3_06	0.59	1.45	13.69	2.04	1.38
gte_3_07	0.92	1.48	16.98	2.17	23.70
gte_3_08	1.49	0.05	13.83	2.19	1.33
gte_3_09	9.10	3.55	14.04	2.28	1.28
gte_3_10	0.21	1.30	14.44	2.24	1.49
gte_3_11	1.39	1.25	14.11	1.77	1.31
gte_3_12	2.17	1.12	13.97	2.05	1.32
gte_3_13	2.22	5.23	14.10	2.06	2.10
gte_3_14	1.64	1.78	14.12	2.19	1.33
gte_3_15	1.20	1.92	11.47	1.67	1.25
gte_3_16	2.02	3.53	17.21	1.94	1.38
gte_3_17	1.61	1.44	13.67	1.03	0.96
gte_3_18	6.01	0.94	14.26	2.06	1.41
gte_3_19	39.87	3.41	14.00	3.10	2.24
total	88.12	47.39	289.44	41.65	51.11
savings	—	46.22%	-228.46%	52.73%	42.00%
gte_4_00	10.64	2.42	21.12	3.46	2.21
gte_4_01	2.58	2.69	22.88	3.88	2.45
gte_4_02	2.35	2.17	22.27	3.39	29.54
gte_4_03	4.37	2.63	21.16	3.49	4.36
gte_4_04	2.90	8.76	20.35	3.29	2.63

Continued on next page

Table A.6 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
gte_4_05	2.51	4.88	21.52	3.49	2.18
gte_4_06	2.42	1.70	21.59	3.38	2.42
gte_4_07	2.32	2.12	21.48	3.30	2.00
gte_4_08	1.52	5.66	21.06	3.49	2.02
gte_4_09	2.57	2.41	21.59	1.24	1.17
gte_4_10	2.66	2.66	20.99	5.35	1.17
gte_4_11	2.45	1.77	21.78	3.71	2.27
gte_4_12	3.65	2.53	22.92	5.33	0.90
gte_4_13	2.19	18.44	38.97	5.03	2.33
gte_4_14	4.02	2.61	22.20	2.22	2.39
gte_4_15	2.53	2.26	35.21	4.32	3.03
gte_4_16	2.89	0.31	22.19	2.49	2.39
gte_4_17	3.40	2.57	21.55	4.78	1.99
gte_4_18	1.01	2.41	21.35	3.56	2.32
gte_4_19	21.66	1.51	21.70	3.54	2.46
total	80.64	72.51	463.88	72.74	72.23
savings	—	10.08%	-475.25%	9.80%	10.43%
gte_5_00	5.51	6.89	31.64	7.44	3.67
gte_5_01	4.49	3.69	42.25	7.05	4.60
gte_5_02	4.34	15.17	32.19	5.73	3.59
gte_5_03	4.67	4.09	31.98	5.99	3.69
gte_5_04	4.00	3.19	42.13	5.44	4.07
gte_5_05	3.45	3.02	30.39	5.46	3.52
gte_5_06	4.15	3.85	29.32	5.34	3.33
gte_5_07	5.71	3.75	32.32	3.43	2.99
gte_5_08	4.88	4.21	30.33	4.51	3.51
gte_5_09	44.49	4.05	31.25	1.10	1.23
gte_5_10	82.34	3.02	32.01	7.53	3.79
gte_5_11	40.71	3.67	44.29	7.37	3.46
gte_5_12	4.18	3.99	47.44	5.20	3.36
gte_5_13	4.01	6.79	30.25	3.57	3.42
gte_5_14	3.94	3.54	28.99	5.60	3.24
gte_5_15	1.29	3.15	33.20	5.67	5.02
gte_5_16	18.52	13.68	30.48	5.53	5.72
gte_5_17	2.98	3.61	57.34	8.98	6.20
gte_5_18	3.43	2.55	30.67	5.45	3.62
gte_5_19	1.89	2.82	29.94	5.94	3.75
total	248.98	98.73	698.41	112.33	75.78
savings	—	60.35%	-180.51%	54.88%	69.56%
gte_6_00	8.02	24.86	40.11	4.51	4.21
gte_6_01	6.62	4.89	52.79	10.72	5.94
gte_6_02	19.27	5.15	42.61	6.36	4.17
gte_6_03	2.63	5.10	40.49	8.70	10.21
gte_6_04	6.54	4.85	43.41	7.77	5.29
gte_6_05	2.85	4.78	39.96	10.34	5.48

Continued on next page

Table A.6 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
gte_6_06	4.41	5.04	41.56	8.34	4.63
gte_6_07	6.58	4.14	70.39	11.51	4.83
gte_6_08	5.53	5.55	39.21	11.04	5.24
gte_6_09	5.20	4.86	41.54	7.36	8.28
gte_6_10	34.04	1.00	39.73	7.65	4.97
gte_6_11	7.48	5.06	40.75	7.59	6.44
gte_6_12	4.49	5.36	39.62	12.39	5.23
gte_6_13	7.57	79.10	39.87	8.92	6.05
gte_6_14	4.43	8.28	40.24	7.79	5.05
gte_6_15	8.25	5.38	39.58	7.56	4.13
gte_6_16	129.80	113.77	39.51	6.59	9.09
gte_6_17	5.72	5.71	56.21	12.38	4.84
gte_6_18	4.90	5.25	42.01	7.75	4.63
gte_6_19	26.77	5.63	40.64	7.69	6.61
total	301.1	303.76	870.23	172.96	115.32
savings	—	-0.88%	-189.02%	42.56%	61.70%
Total	732.3	557.35	2,368.75	419.64	332.51
Savings	—	23.89%	-223.47%	42.70%	54.59%

Table A.7: Set covering problems (all settings except the branching scheme set to default): comparison on number of nodes explored.

Name	Number of nodes explored					Top 5
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
gte_3_00	189	49	71	125	121	53.97
gte_3_01	0	6	0	15	17	62.50
gte_3_02	459	105	163	235	247	64.41
gte_3_03	229	72	123	229	207	66.96
gte_3_04	43	11	15	29	33	80.00
gte_3_05	432	255	398	856	860	65.00
gte_3_06	454	144	231	357	357	56.98
gte_3_07	797	120	217	305	317	62.75
gte_3_08	357	96	145	215	209	48.15
gte_3_09	1,032	328	527	739	733	56.49
gte_3_10	154	39	61	79	91	72.50
gte_3_11	206	72	117	161	155	54.32
gte_3_12	182	53	91	109	109	63.64
gte_3_13	372	99	278	331	259	61.11
gte_3_14	90	24	41	87	75	54.55
gte_3_15	308	151	451	621	380	65.61
gte_3_16	3,094	661	1,687	3,310	1,958	63.22
gte_3_17	162	180	297	350	149	60.75
gte_3_18	653	168	269	385	397	64.25
gte_3_19	25	10	9	21	21	63.64

Continued on next page

Table A.7 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
total	9,238	2,643	5,191	8,559	6,695	62.04
savings	—	71.39%	43.81%	7.35%	27.53%	
gte_4_00	1,899	424	695	1,135	1,207	54.40
gte_4_01	21,937	5,203	9,610	12,951	11,401	62.67
gte_4_02	2,018	649	1,124	2,199	2,492	48.04
gte_4_03	2,059	921	1,660	1,700	1,030	58.51
gte_4_04	1,306	255	301	883	667	50.81
gte_4_05	2,845	656	1,141	1,329	1,429	66.77
gte_4_06	653	243	343	429	469	66.05
gte_4_07	1,599	417	677	1,017	1,079	63.06
gte_4_08	1,167	824	886	1,082	871	62.31
gte_4_09	2,891	729	1,401	1,967	1,997	52.64
gte_4_10	1,308	312	615	791	773	59.09
gte_4_11	316	91	151	211	233	58.49
gte_4_12	2,009	723	975	1,273	1,297	62.17
gte_4_13	334	96	171	315	321	51.90
gte_4_14	629	184	571	716	576	61.52
gte_4_15	605	419	1,255	1,243	974	53.83
gte_4_16	4,949	1,361	2,563	2,835	2,869	67.70
gte_4_17	927	208	283	465	475	56.22
gte_4_18	6,991	1,794	3,285	4,367	4,655	60.39
gte_4_19	6,327	1,965	3,177	4,277	4,211	63.07
total	62,769	17,474	30,884	41,185	39,026	58.98
savings	—	72.16%	50.80%	34.39%	37.83%	
gte_5_00	23,821	6,681	9,512	12,221	11,611	67.89
gte_5_01	5,242	1,614	2,676	3,576	2,587	50.02
gte_5_02	86,293	19,232	34,643	54,551	58,833	47.92
gte_5_03	24,463	7,896	9,575	29,202	8,363	52.49
gte_5_04	25,340	7,976	13,415	29,427	13,484	53.59
gte_5_05	50,420	13,794	43,890	37,584	22,105	53.97
gte_5_06	25,458	5,035	7,595	13,525	11,909	49.37
gte_5_07	23,613	5,768	7,930	12,201	11,901	50.54
gte_5_08	40,940	7,437	9,239	20,675	19,945	47.68
gte_5_09	18,382	5,219	13,144	16,116	8,763	54.40
gte_5_10	6,040	1,418	2,437	2,649	2,573	59.62
gte_5_11	84,832	15,154	23,419	34,019	34,353	50.02
gte_5_12	9,816	2,616	4,569	5,635	5,529	62.78
gte_5_13	45,062	10,122	13,891	27,651	27,971	45.64
gte_5_14	5,031	1,131	2,265	2,589	2,539	57.53
gte_5_15	2,961	494	963	1,375	1,283	60.32
gte_5_16	4,631	1,378	1,951	4,189	3,631	50.07
gte_5_17	48,178	9,118	15,211	19,939	21,947	53.08
gte_5_18	6,636	3,538	3,809	5,671	7,032	54.30
gte_5_19	3,373	858	1,213	1,413	1,455	59.97
total	540,532	126,479	221,347	334,208	277,814	54.06

Continued on next page

Table A.7 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
savings	—	76.60%	59.05%	38.17%	48.60%	
gte_6_00	42,312	10,259	14,377	24,751	22,243	48.47
gte_6_01	29,328	5,845	6,606	26,683	9,410	50.18
gte_6_02	475,541	112,763	130,332	153,365	143,540	50.71
gte_6_03	292,621	51,922	81,139	113,165	114,129	51.57
gte_6_04	157,153	38,415	54,504	61,817	54,355	51.80
gte_6_05	1,000,000	237,913	372,956	583,261	327,250	50.51
gte_6_06	786,699	152,654	247,627	288,791	246,429	49.50
gte_6_07	129,674	36,923	52,717	63,267	80,168	49.83
gte_6_08	22,914	7,018	22,817	16,759	49,319	47.18
gte_6_09	915,036	156,113	261,332	389,466	383,797	49.30
gte_6_10	43,531	7,324	29,507	23,517	15,987	47.51
gte_6_11	357,677	61,959	117,870	233,052	131,020	52.92
gte_6_12	318,621	44,247	95,685	92,456	71,021	45.96
gte_6_13	53,450	12,163	27,238	30,744	28,400	47.67
gte_6_14	161,352	21,305	37,535	53,115	57,937	48.45
gte_6_15	964,732	79,484	133,315	178,729	176,585	45.88
gte_6_16	33,315	6,117	10,941	13,859	13,201	51.01
gte_6_17	35,555	3,825	23,971	34,350	10,231	47.29
gte_6_18	29,981	8,719	12,647	22,948	8,188	54.79
gte_6_19	138,349	36,283	68,913	83,001	81,530	46.62
total	5,987,841	1,091,251	1,802,029	2,487,096	2,024,740	49.36
savings	—	81.78%	69.91%	58.46%	66.19%	
scpnre1	28,695	8,265	14,773	14,971	14,519	55.18
scpnre2	360,101	20,432	47,850	120,035	36,435	54.34
scpnre3	24,363	10,330	22,636	37,510	7,977	50.48
scpnre4	48,924	12,693	34,000	26,354	18,257	49.00
scpnre5	11,714	2,424	3,475	6,827	6,707	48.65
total	473,797	54,144	122,734	205,697	83,895	51.53
savings	—	88.57%	74.10%	56.59%	82.29%	
scpnrf1	28,500	4,850	9,769	13,975	15,081	55.25
scpnrf2	14,850	3,403	10,605	9,475	9,564	51.40
scpnrf3	26,004	5,777	9,859	9,897	7,374	52.25
scpnrf4	49,738	15,290	32,663	25,962	45,739	54.84
scpnrf5	180,911	38,135	134,708	31,066	263,011	57.40
total	300,003	67,455	197,604	90,375	340,769	54.23
savings	—	77.52%	34.13%	69.88%	-13.59%	
Total	7,374,180	1,359,446	2,379,789	3,167,120	2,772,939	55.75
Savings	—	81.56%	67.73%	57.05%	62.40%	

Table A.8: Set covering problems (all settings except the branching scheme set to default): comparison on run time.

Name	Run time (seconds)				
	CPLEX-D	CPLEX-SB d	OUR-SB	LRN-SB	LRN-GSB
gte_3_00	1.03	1.21	1.59	0.83	0.84
gte_3_01	0.64	0.60	0.65	0.54	0.54
gte_3_02	1.08	1.73	2.44	0.83	0.86
gte_3_03	0.86	1.29	1.86	0.72	0.73
gte_3_04	0.91	0.87	1.01	0.77	0.77
gte_3_05	1.20	3.33	4.93	1.46	1.56
gte_3_06	1.17	2.15	3.15	1.00	1.05
gte_3_07	1.27	1.75	2.78	0.85	0.89
gte_3_08	0.99	1.62	2.21	0.79	0.80
gte_3_09	1.70	4.08	5.95	1.40	1.50
gte_3_10	1.18	1.27	1.69	0.95	0.96
gte_3_11	0.89	1.25	1.78	0.74	0.74
gte_3_12	0.95	1.24	1.77	0.75	0.76
gte_3_13	1.28	2.07	4.16	1.15	1.05
gte_3_14	0.74	0.81	1.06	0.60	0.61
gte_3_15	1.28	2.74	5.40	1.34	1.22
gte_3_16	3.72	9.29	20.57	4.40	3.22
gte_3_17	1.03	2.51	3.91	0.99	0.82
gte_3_18	1.47	2.70	3.75	1.13	1.22
gte_3_19	0.82	0.81	0.89	0.68	0.69
total	24.21	43.32	71.55	21.92	20.83
savings	—	-78.93%	-195.54%	9.46%	13.96%
gte_4_00	3.23	7.43	10.31	2.62	2.85
gte_4_01	26.22	88.14	127.15	18.87	19.41
gte_4_02	3.42	11.62	16.68	3.73	4.47
gte_4_03	3.62	16.81	25.26	3.44	2.80
gte_4_04	2.54	5.01	5.03	1.99	1.82
gte_4_05	4.61	10.49	16.10	3.31	3.47
gte_4_06	1.85	4.68	5.68	1.49	1.59
gte_4_07	3.09	6.65	9.61	2.62	2.86
gte_4_08	2.81	13.50	14.51	2.60	2.44
gte_4_09	4.54	12.30	19.83	3.76	4.30
gte_4_10	2.91	5.81	9.55	2.36	2.50
gte_4_11	1.70	2.62	3.50	1.41	1.41
gte_4_12	3.90	12.07	14.47	3.12	3.25
gte_4_13	1.57	2.48	3.41	1.48	1.49
gte_4_14	2.09	4.27	9.42	2.03	1.96
gte_4_15	1.83	6.25	16.46	2.33	2.27
gte_4_16	7.84	25.02	38.04	5.84	6.58
gte_4_17	2.34	4.55	5.14	1.84	2.00
gte_4_18	7.94	25.08	41.59	6.44	7.91
gte_4_19	8.25	30.88	41.67	6.95	7.36

Continued on next page

Table A.8 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
total	96.3	295.66	433.41	78.23	82.74
savings	—	-207.02%	-350.06%	18.76%	14.08%
gte_5_00	35.62	145.97	165.85	25.83	25.84
gte_5_01	9.71	39.71	53.62	8.20	6.93
gte_5_02	121.13	407.12	541.47	85.75	102.61
gte_5_03	38.16	167.96	190.54	49.39	20.16
gte_5_04	40.00	184.60	267.97	52.89	29.88
gte_5_05	81.72	332.36	891.56	69.97	53.03
gte_5_06	35.46	100.46	125.03	25.30	23.56
gte_5_07	34.29	125.11	138.51	22.75	24.59
gte_5_08	43.94	139.87	135.09	32.40	35.19
gte_5_09	26.43	103.40	226.30	27.78	17.39
gte_5_10	11.04	31.33	42.01	7.21	7.28
gte_5_11	108.59	263.03	352.12	48.59	63.21
gte_5_12	16.79	56.41	80.71	14.33	15.18
gte_5_13	57.60	174.67	201.77	39.81	45.70
gte_5_14	8.69	22.46	37.87	5.70	6.51
gte_5_15	5.93	10.65	17.16	4.08	4.08
gte_5_16	7.54	26.12	31.38	8.13	8.16
gte_5_17	71.63	185.25	255.46	38.43	47.75
gte_5_18	11.00	75.40	69.97	11.24	14.86
gte_5_19	7.41	19.26	22.57	4.71	4.93
total	772.68	2,611.14	3,846.96	582.49	556.84
savings	—	-237.93%	-397.87%	24.61%	27.93%
gte_6_00	71.50	237.77	269.94	48.40	53.28
gte_6_01	51.57	158.02	131.88	56.08	24.77
gte_6_02	811.37	3276.67	2808.39	329.69	351.68
gte_6_03	444.64	1150.59	1447.43	208.26	245.13
gte_6_04	300.65	1145.51	1159.07	146.18	140.63
gte_6_05	2090.74	7954.58	9334.73	1343.48	894.80
gte_6_06	1642.13	4574.53	5542.81	631.67	623.79
gte_6_07	219.59	1091.93	1158.55	136.18	191.34
gte_6_08	49.13	193.82	524.24	32.40	114.39
gte_6_09	1418.36	4333.46	5150.20	782.76	878.00
gte_6_10	90.44	215.14	732.85	53.31	40.58
gte_6_11	650.05	1827.87	2567.68	503.31	333.31
gte_6_12	541.22	1147.01	2063.62	179.02	154.35
gte_6_13	97.24	365.06	654.05	67.15	70.05
gte_6_14	255.16	457.85	676.52	93.79	119.40
gte_6_15	1659.98	2087.70	2733.66	381.12	430.00
gte_6_16	63.74	156.69	224.07	33.01	35.14
gte_6_17	71.76	121.19	534.89	82.06	32.28
gte_6_18	63.96	258.78	326.49	55.52	25.62
gte_6_19	248.08	981.12	1430.96	162.82	181.08
total	10,841.31	31,735.29	39,472.03	5,326.21	4,939.62

Continued on next page

Table A.8 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
savings	—	-192.73%	-264.09%	50.87%	54.44%
scpnre1	36.17	152.47	227.91	26.19	26.68
scpnre2	473.95	477.54	963.13	211.44	72.64
scpnre3	40.02	222.76	383.73	61.35	18.44
scpnre4	67.95	262.26	638.79	42.70	34.48
scpnre5	17.96	47.43	57.40	12.95	14.71
total	636.05	1,162.46	2,270.96	354.63	166.95
savings	—	-82.76%	-257.04%	44.24%	73.75%
scpnrf1	37.23	92.17	148.00	22.09	27.10
scpnrf2	21.70	75.45	201.53	16.48	16.36
scpnrf3	33.85	111.19	163.98	17.28	15.45
scpnrf4	70.52	318.81	624.99	40.11	77.69
scpnrf5	225.45	786.69	2148.05	53.71	360.18
total	388.75	1,384.31	3,286.55	149.67	496.78
savings	—	-256.09%	-745.41%	61.50%	-27.79%
Total	12,759.3	37,232.18	49,381.46	6,513.15	6,263.76
Savings	—	-191.80%	-287.02%	48.95%	50.91%

Table A.9: Set packing problems (all settings except the branching scheme set to default): comparison on number of nodes explored.

Name	Number of nodes explored					Top 5
	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
gte_100_00	69	25	57	65	65	40.26
gte_100_01	1,053	284	527	633	681	60.13
gte_100_02	2,384	384	689	1,285	1,018	62.65
gte_100_03	632	124	337	389	449	58.49
gte_100_04	436	97	219	255	293	54.00
gte_100_05	618	89	181	451	297	59.30
gte_100_06	291	54	137	147	153	55.00
gte_100_07	1,259	296	640	906	814	56.08
gte_100_08	237	69	147	167	161	58.94
gte_100_09	1,480	415	871	1,519	1,241	53.50
gte_100_10	333	97	187	241	219	57.14
gte_100_11	6,848	694	2,551	3,948	4,119	58.79
gte_100_12	4,883	587	1,681	2,351	2,178	61.90
gte_100_13	728	240	408	440	569	57.75
gte_100_14	937	177	407	523	541	58.94
gte_100_15	204	61	164	266	194	60.73
gte_100_16	1,615	429	826	1,027	1,007	57.26
gte_100_17	629	136	279	365	365	58.41
gte_100_18	1,048	192	430	720	684	50.40
gte_100_19	505	204	332	239	251	65.19
total	26,189	4,654	11,070	15,937	15,299	57.24

Continued on next page

Table A.9 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
savings	—	82.23%	57.73%	39.15%	41.58%	
gte_150_00	3,994	1,128	1,373	1,517	1,601	71.71
gte_150_01	5,614	1,642	2,557	3,326	2,914	67.70
gte_150_02	6,719	1,776	3,495	4,131	3,877	65.52
gte_150_03	9,236	2,389	4,793	5,317	4,516	64.99
gte_150_04	6,733	1,657	3,504	4,460	3,771	67.53
gte_150_05	6,733	1,128	2,409	3,252	3,828	69.46
gte_150_06	4,179	985	1,791	2,313	2,303	67.76
gte_150_07	8,842	2,470	4,069	5,050	4,933	69.75
gte_150_08	12,999	3,762	4,584	7,171	5,835	65.51
gte_150_09	8,739	1,844	3,616	4,100	4,627	65.08
gte_150_10	10,432	1,731	3,711	4,695	4,229	67.26
gte_150_11	2,695	812	955	1,217	1,129	61.09
gte_150_12	8,334	2,374	3,273	3,833	4,569	69.43
gte_150_13	6,811	2,336	2,271	3,699	4,101	66.59
gte_150_14	17,061	4,423	7,634	8,882	9,534	69.12
gte_150_15	4,804	1,457	2,105	2,883	3,040	70.86
gte_150_16	9,437	2,548	3,260	4,590	4,567	63.83
gte_150_17	6,534	985	2,151	1,734	2,257	69.68
gte_150_18	3,019	906	1,349	1,843	1,735	68.86
gte_150_19	8,212	3,019	3,437	5,193	4,671	68.84
total	151,127	39,372	62,337	79,206	78,037	67.53
savings	—	73.95%	58.75%	47.59%	48.36%	
gte_200_00	11,529	2,654	3,613	4,143	3,880	76.74
gte_200_01	13,042	2,580	4,067	4,449	4,521	75.30
gte_200_02	5,935	1,932	3,065	3,375	3,229	78.37
gte_200_03	10,275	2,896	4,741	5,119	5,295	78.40
gte_200_04	8,510	2,820	4,544	5,123	4,691	75.91
gte_200_05	7,163	1,617	2,285	2,581	2,579	76.72
gte_200_06	10,491	2,034	3,302	4,616	4,305	77.59
gte_200_07	12,420	3,518	5,134	5,902	5,488	79.98
gte_200_08	12,543	3,321	6,063	6,349	6,121	77.09
gte_200_09	13,989	3,672	4,652	5,485	6,601	77.45
gte_200_10	7,178	1,599	2,251	2,743	2,921	76.37
gte_200_11	8,794	2,333	3,838	4,436	4,421	76.62
gte_200_12	14,920	4,123	6,366	6,459	6,607	78.68
gte_200_13	11,265	2,339	3,740	4,520	4,783	79.71
gte_200_14	12,167	3,535	4,189	4,270	5,550	77.92
gte_200_15	10,793	3,109	3,679	3,921	4,409	78.69
gte_200_16	11,926	3,197	5,599	5,989	6,397	76.02
gte_200_17	8,415	2,265	3,509	4,377	4,363	77.20
gte_200_18	8,801	2,601	3,971	3,955	3,887	76.73
gte_200_19	13,832	4,577	6,228	6,630	6,331	77.18
total	213,988	56,722	84,836	94,442	96,379	77.43
savings	—	73.49%	60.35%	55.87%	54.96%	

Continued on next page

Table A.9 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
gte_250_00	17,114	4,172	6,435	7,313	7,410	82.64
gte_250_01	22,247	5,380	10,368	10,880	10,181	81.99
gte_250_02	18,336	4,392	7,993	8,603	8,338	81.64
gte_250_03	13,427	3,233	5,843	6,273	6,099	82.15
gte_250_04	14,130	3,245	5,937	5,847	6,243	81.68
gte_250_05	11,298	2,732	4,719	4,965	4,947	82.02
gte_250_06	17,460	3,748	7,292	7,527	7,688	80.56
gte_250_07	21,971	5,386	9,621	9,955	10,209	80.81
gte_250_08	18,015	4,234	7,439	7,841	7,831	81.05
gte_250_09	20,101	4,413	7,656	7,936	8,064	82.94
gte_250_10	17,919	4,389	7,621	7,161	8,817	81.75
gte_250_11	18,005	3,731	7,523	7,769	7,827	81.63
gte_250_12	12,128	3,045	5,607	5,745	5,881	81.92
gte_250_13	15,681	3,617	6,449	6,779	6,731	81.89
gte_250_14	18,485	4,063	7,697	8,189	8,307	82.96
gte_250_15	16,419	3,577	6,899	7,064	7,236	82.87
gte_250_16	11,576	2,667	4,945	4,969	5,171	81.87
gte_250_17	11,448	2,584	4,323	4,497	4,547	81.15
gte_250_18	22,703	5,797	10,485	11,562	11,596	80.51
gte_250_19	10,175	2,212	3,805	4,313	4,159	81.41
total	328,638	76,617	138,657	145,188	147,282	81.77
savings	—	76.69%	-0.20%	57.81%	55.82%	55.18%
gte_300_00	16,505	3,736	6,414	6,875	6,892	83.90
gte_300_01	25,929	8,203	10,947	11,445	11,538	83.68
gte_300_02	17,527	4,041	7,714	7,892	8,387	83.84
gte_300_03	17,767	3,960	7,307	7,559	7,413	83.80
gte_300_04	13,648	4,489	6,573	6,371	6,477	84.01
gte_300_05	10,595	3,015	4,379	4,569	4,583	84.28
gte_300_06	12,937	3,031	5,830	5,968	5,776	83.47
gte_300_07	15,196	3,893	6,486	7,046	6,878	84.60
gte_300_08	16,705	3,717	7,420	7,912	7,381	83.26
gte_300_09	13,087	2,772	5,567	5,593	5,591	84.26
gte_300_10	17,004	3,624	7,063	7,217	7,093	83.90
gte_300_11	9,332	2,486	4,681	4,649	4,555	84.23
gte_300_12	17,515	3,903	8,137	8,027	8,143	83.00
gte_300_13	14,201	2,955	6,247	6,203	6,453	83.74
gte_300_14	10,999	2,831	4,745	5,001	5,057	83.86
gte_300_15	17,854	3,998	7,711	8,242	8,322	84.51
gte_300_16	19,322	4,359	8,153	9,152	8,053	84.18
gte_300_17	8,098	2,179	4,043	4,165	4,107	83.99
gte_300_18	26,069	6,392	12,005	12,279	12,163	83.77
gte_300_19	14,965	3,924	6,753	6,887	7,201	84.02
total	315,255	77,508	138,175	143,052	142,063	83.91
savings	—	75.41%	56.17%	54.62%	54.94%	

Continued on next page

Table A.9 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB	Accuracy%
Total	1,035,197	254,873	435,075	477,825	479,060	73.58
Savings	—	75.38%	57.97%	53.84%	53.72%	

Table A.10: Set packinging problems (all settings except the branching scheme set to default): comparison on run time.

Name	Run time (seconds)				
	CPLEX-D	CPLEX-SB d	OUR-SB	LRN-SB	LRN-GSB
gte_100_00	0.73	0.74	1.18	0.56	0.59
gte_100_01	1.24	3.44	5.47	0.97	1.05
gte_100_02	2.06	4.47	7.03	1.51	1.38
gte_100_03	1.02	1.93	4.05	0.74	0.80
gte_100_04	1.09	1.82	2.69	0.84	0.87
gte_100_05	0.81	1.30	2.39	0.62	0.55
gte_100_06	1.00	1.29	2.08	0.75	0.76
gte_100_07	1.59	4.15	6.95	1.33	1.29
gte_100_08	0.63	1.31	2.03	0.48	0.48
gte_100_09	1.52	5.27	10.70	1.54	1.37
gte_100_10	0.75	1.55	2.64	0.49	0.52
gte_100_11	4.04	7.68	23.70	2.88	3.57
gte_100_12	3.50	7.08	15.61	1.95	2.15
gte_100_13	1.11	3.12	4.28	0.83	1.03
gte_100_14	1.30	2.54	4.39	0.93	1.03
gte_100_15	0.67	1.10	2.09	0.65	0.58
gte_100_16	1.45	4.76	8.31	1.06	1.13
gte_100_17	1.20	2.49	3.70	0.85	0.91
gte_100_18	1.19	2.51	4.93	0.77	0.83
gte_100_19	1.21	3.03	4.36	0.85	0.88
total	28.11	61.58	118.58	20.6	21.77
savings	—	-119.07%	-321.84%	26.72%	22.55%
gte_150_00	8.95	25.61	37.66	5.06	5.16
gte_150_01	9.45	35.55	60.86	6.55	7.17
gte_150_02	11.88	36.47	75.40	11.30	8.57
gte_150_03	13.72	45.16	93.77	9.90	8.93
gte_150_04	14.32	34.17	69.39	9.56	9.02
gte_150_05	9.89	25.08	54.74	7.67	9.00
gte_150_06	8.18	23.70	45.91	6.16	6.54
gte_150_07	15.45	44.62	94.18	11.05	11.93
gte_150_08	17.31	66.91	84.78	11.42	11.46
gte_150_09	12.67	34.38	72.44	8.57	8.63
gte_150_10	16.29	36.68	78.43	10.63	10.12
gte_150_11	6.39	18.95	24.51	4.33	4.32
gte_150_12	13.65	44.66	70.98	7.59	10.33
gte_150_13	10.18	46.83	50.75	7.17	9.36

Continued on next page

Table A.10 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
gte_150_14	22.69	76.20	128.26	14.91	16.77
gte_150_15	8.85	32.74	49.32	7.25	7.35
gte_150_16	14.02	45.97	65.63	8.67	9.33
gte_150_17	11.27	24.99	62.68	4.90	7.04
gte_150_18	6.39	20.87	32.79	4.65	4.75
gte_150_19	12.62	59.44	69.70	10.45	10.24
total	244.17	778.98	1,322.18	167.79	176.02
savings	—	-219.03%	-441.50%	31.28%	27.91%
gte_200_00	35.22	89.89	168.88	23.53	22.49
gte_200_01	41.75	86.48	170.14	23.14	26.79
gte_200_02	24.82	73.18	151.87	20.61	22.83
gte_200_03	34.78	92.09	192.86	27.38	25.25
gte_200_04	27.75	99.80	215.84	24.81	27.70
gte_200_05	23.47	60.64	123.64	17.10	17.58
gte_200_06	29.86	74.13	177.55	23.82	23.25
gte_200_07	42.23	119.99	216.07	32.12	28.93
gte_200_08	42.36	104.51	241.34	31.13	28.88
gte_200_09	37.59	110.79	192.85	27.51	30.05
gte_200_10	25.23	66.82	119.39	17.61	18.33
gte_200_11	30.37	89.98	183.44	31.65	25.37
gte_200_12	41.08	123.55	235.66	28.81	29.96
gte_200_13	38.01	84.56	189.25	26.27	26.30
gte_200_14	34.67	119.41	206.36	26.30	32.00
gte_200_15	34.96	101.73	186.89	22.30	26.15
gte_200_16	35.10	93.72	205.91	27.86	32.08
gte_200_17	27.23	77.00	150.10	20.77	22.33
gte_200_18	30.98	88.11	177.27	22.10	22.87
gte_200_19	36.68	123.51	225.80	28.92	29.41
total	674.14	1,879.89	3,731.11	503.74	518.55
savings	—	-178.86%	-453.46%	25.28%	23.08%
gte_250_00	68.13	198.69	463.18	48.61	53.51
gte_250_01	71.84	230.35	613.26	62.12	59.44
gte_250_02	64.81	209.25	480.89	51.92	47.38
gte_250_03	56.56	150.38	428.13	44.08	45.82
gte_250_04	60.45	157.25	428.64	41.60	43.55
gte_250_05	54.77	153.04	437.28	39.93	45.28
gte_250_06	64.22	174.22	470.23	49.77	52.29
gte_250_07	81.78	240.71	645.59	62.93	60.32
gte_250_08	66.93	190.83	464.90	48.19	48.35
gte_250_09	68.21	203.76	498.31	53.99	51.20
gte_250_10	66.37	199.93	497.59	45.26	51.17
gte_250_11	78.38	196.38	539.07	55.32	59.00
gte_250_12	50.59	161.67	440.46	43.88	47.91
gte_250_13	68.28	187.91	497.25	51.10	51.00
gte_250_14	69.21	181.35	483.69	52.58	52.23

Continued on next page

Table A.10 – *Continued from previous page*

Name	CPLEX-D	CPLEX-SB	OUR-SB	LRN-SB	LRN-GSB
gte_250_15	79.97	176.15	523.07	49.39	52.62
gte_250_16	53.76	137.21	410.69	40.36	43.54
gte_250_17	51.85	136.79	387.10	39.74	39.37
gte_250_18	73.16	230.68	551.25	61.51	59.34
gte_250_19	62.83	128.76	352.65	43.51	40.99
total	1,312.1	3,645.31	9,613.23	985.79	1,004.31
savings	—	-177.82%	-632.66%	24.87%	23.46%
gte_300_00	122.57	316.91	892.86	89.88	94.97
gte_300_01	120.62	428.06	1034.22	99.83	102.98
gte_300_02	125.73	340.72	1002.58	91.13	104.03
gte_300_03	118.53	283.77	837.30	82.95	90.92
gte_300_04	95.71	308.64	947.53	87.59	79.77
gte_300_05	90.57	243.61	736.21	67.34	77.49
gte_300_06	102.49	268.71	860.52	79.88	83.70
gte_300_07	106.38	309.62	870.77	92.54	87.49
gte_300_08	130.78	308.04	900.72	88.71	86.33
gte_300_09	111.00	284.31	836.13	84.60	85.00
gte_300_10	108.56	261.17	793.62	80.16	85.24
gte_300_11	96.55	239.06	767.32	73.31	75.03
gte_300_12	142.59	306.67	956.05	83.39	91.42
gte_300_13	98.07	248.74	821.59	75.40	78.62
gte_300_14	96.60	260.27	860.17	78.59	78.40
gte_300_15	135.08	344.62	1021.62	104.59	93.30
gte_300_16	124.48	314.98	946.35	94.00	89.02
gte_300_17	100.12	245.16	811.96	75.45	74.00
gte_300_18	151.81	368.15	1064.10	94.52	99.80
gte_300_19	105.80	308.06	939.96	80.28	80.24
total	2,284.04	5,989.27	17,901.58	1,704.14	1,737.75
savings	—	-162.22%	-683.77%	25.39%	23.92%
Total	4,542.56	12,355.03	32,686.68	3,382.06	3,458.4
Savings	—	-171.98%	-619.57%	25.55%	23.87%

APPENDIX B

ILLUSTRATIVE INSTANCES IN CHAPTER 4

B.1 Instance in Proposition 4.2.2

Here we show the complete instance and feasible solution used in Proposition 4.2.2. The complete network is shown in Figure B.1.

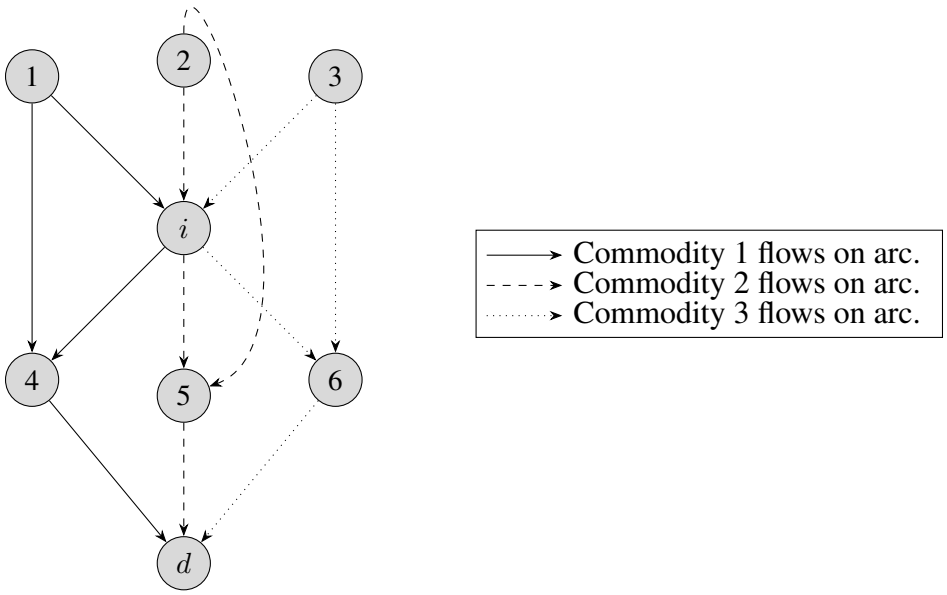


Figure B.1: Complete network for Proposition 4.2.2

Commodities k_1, k_2, and k_3 have origins 1, 2, and 3, respectively, and are all destined

for node d. The solution of interest takes the following form:

$$x_{(1,4),k_1} = \frac{5}{8}, \quad x_{(1,i),k_1} = \frac{3}{8}, \quad x_{(2,i),k_2} = \frac{3}{8}, \quad x_{(2,5),k_2} = \frac{5}{8}, \quad x_{(3,i),k_3} = \frac{3}{8},$$
$$x_{(3,6),k_3} = \frac{5}{8}, \quad x_{(i,4),k_1} = \frac{3}{8}, \quad x_{(i,5),k_2} = \frac{3}{8}, \quad x_{(i,6),k_3} = \frac{3}{8}, \quad x_{(4,d),k_1} = 1,$$
$$x_{(5,d),k_2} = 1, \quad x_{(6,d),k_3} = 1,$$

and all other x_{ak} variables are set to 0. It can be easily checked that the above solution lies in $LP(S^1)$.

B.2 Instance in Proposition 4.2.2

Here we show the calculations for the feasible solution for the instance in Figure 4.3. Recall the given feasible solution:

$$w_{(1,i),d} = \frac{1}{40}, \quad w_{(1,d),d} = \frac{3}{40}, \quad w_{(2,i),d} = \frac{3}{8}, \quad w_{(2,d),d} = \frac{1}{8}, \quad w_{(i,d),d} = \frac{2}{5}$$
$$y_{(1,i),d} = \frac{1}{4}, \quad y_{(1,d),d} = \frac{3}{4}, \quad y_{(2,i),d} = \frac{3}{4}, \quad y_{(2,d),d} = \frac{1}{4}, \quad y_{(i,d),d} = \frac{2}{3}.$$

All other w_{ad} and y_{ad} variables are set to 0 and all n_a variables can be set to any feasible lower bound for each arc.

- For Constraint (4.10):

$$
\begin{aligned}
w_{(1,i),d} + w_{(1,d),d} &= \frac{1}{40} + \frac{3}{40} = q_{k_1} = \frac{1}{10}, \quad \text{for node } o_1, \\
w_{(2,i),d} + w_{(2,d),d} &= \frac{3}{8} + \frac{1}{8} = q_{k_2} = \frac{1}{2}, \quad \text{for node } o_2, \\
w_{(i,d),d} - w_{(1,i),d} - w_{(2,i),d} &= \frac{2}{5} - \frac{1}{40} - \frac{3}{8} = 0, \quad \text{for node } i, \\
-w_{(1,d),d} - w_{(i,d),d} - w_{(2,d),d} &= -\frac{3}{40} - \frac{2}{5} - \frac{1}{8} = -q_{k_1} - q_{k_2} = -\frac{3}{5}, \quad \text{for node } d.
\end{aligned}
$$

- For Constraint (4.11):

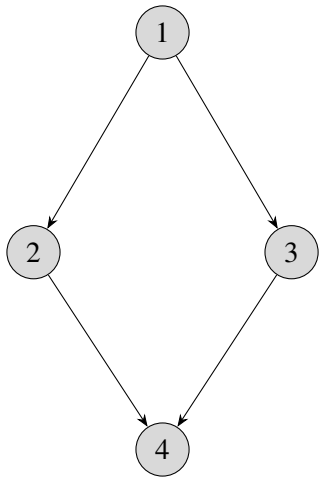
$$
\begin{aligned}
w_{(1,i),d} = \frac{1}{40} \leq \frac{3}{5} y_{(1,i),d} &= \frac{3}{5} \cdot \frac{1}{4} = \frac{3}{20}, \quad \text{for arc } (1,i), \\
w_{(2,i),d} = \frac{3}{8} \leq \frac{3}{5} y_{(2,i),d} &= \frac{3}{5} \cdot \frac{3}{4} = \frac{9}{20}, \quad \text{for arc } (2,i), \\
w_{(1,d),d} = \frac{3}{40} \leq \frac{3}{5} y_{(1,d),d} &= \frac{3}{5} \cdot \frac{3}{4} = \frac{9}{20}, \quad \text{for arc } (1,d), \\
w_{(i,d),d} = \frac{2}{5} \leq \frac{3}{5} y_{(i,d),d} &= \frac{3}{5} \cdot \frac{2}{3} = \frac{2}{5}, \quad \text{for arc } (i,d), \\
w_{(2,d),d} = \frac{1}{8} \leq \frac{3}{5} y_{(2,d),d} &= \frac{3}{5} \cdot \frac{1}{4} = \frac{3}{20}, \quad \text{for arc } (2,d).
\end{aligned}
$$

Constraint (4.5) can easily be checked for each node and the n_a variables can be set to any feasible lower bound for each arc.

B.3 An Simple Illustrative Instance on F3

Here we consider a small illustrative instance that shows the effect of the simple strengthening inequalities, the cut-set inequalities as well as the wheat-stalk inequalities on Formulation F3. The instance has a network with $N = \{1, 2, 3, 4\}$, $A = \{(1,2), (1,3), (2,4), (3,4)\}$,

arc lengths $c_{(1,3)} = 1.5$, $c_{(1,2)} = c_{(2,4)} = c_{(3,4)} = 1$, and four commodities $K = \{1, 2, 3, 4\}$ with parameters given in Figure B.2.



Commodity k	Origin $o(k)$	Destination $d(k)$	Quantity $q(k)$
1	1	4	0.6
2	2	4	0.5
3	3	4	0.4
4	1	3	0.9

Figure B.2: Left: network for the illustrative instance; Right: commodity data for the illustrative instance.

Clearly, only commodity 1 can use arc $(1, 2)$, only commodity 3 can use arc $(3, 4)$, only commodities 1 and 2 can use arc $(2, 4)$ and only commodities 1 and 4 can use arc $(1, 3)$. The Formulation F3 for this instance, after removing constraints that are redundant for its

LP relaxation, is thus

$$\begin{aligned}
\min \quad & n_{(1,2)} + n_{(2,4)} + n_{(3,4)} + 1.5n_{(1,3)} \\
& w_{(1,3),3} = 0.9 && \text{(B.1)} \\
& w_{(1,2),4} + w_{(1,3),4} = 0.6, && \text{(B.2)} \\
& w_{(2,4),4} - w_{(1,2),4} = 0.5, && \text{(B.3)} \\
& w_{(3,4),4} - w_{(1,3),4} = 0.4, && \text{(B.4)} \\
& y_{(1,2),4} + y_{(1,3),4} \leq 1, && \text{(B.5)} \\
& w_{a,4} \leq 1.5y_{a,4}, \quad \forall a \in A, && \text{(B.6)} \\
& w_{(1,3),3} \leq 0.9y_{(1,3),3}, && \text{(B.7)} \\
& n_{(1,2)} \geq w_{(1,2),4}, && \text{(B.8)} \\
& n_{(1,3)} \geq w_{(1,3),4} + w_{(1,3),3}, && \text{(B.9)} \\
& n_{(2,4)} \geq w_{(2,4),4}, && \text{(B.10)} \\
& n_{(3,4)} \geq w_{(3,4),4}, && \text{(B.11)} \\
& n_{(1,2)}, n_{(1,3)}, n_{(2,4)}, n_{(3,4)} \in \mathbb{Z}_{\geq 0}, \\
& w_{(1,3),3}, w_{(1,2),4}, w_{(1,3),4}, w_{(2,4),4}, w_{(3,4),4} \geq 0, \\
& y_{(1,3),3}, y_{(1,2),4}, y_{(1,3),4}, y_{(2,4),4}, y_{(3,4),4} \in \{0,1\}.
\end{aligned}$$

Solving the LP relaxation of this formulation gives an objective value of 3.45. Let $(\tilde{n}, \tilde{y}, \tilde{w})$ denote the solution to the LP relaxation. Then $\tilde{w}_{(1,2),4} = 0.6$, $\tilde{w}_{(1,3),4} = 0$, $\tilde{w}_{(2,4),4} = 1.1$, $\tilde{w}_{(3,4),4} = 0.4$, $\tilde{y}_{(1,2),4} = 0.6$, $\tilde{y}_{(1,3),4} = 0$, $\tilde{y}_{(2,4),4} = 1$, $\tilde{y}_{(3,4),4} = 1$. Note that (B.1) and (B.7) combine to force $\tilde{y}_{(1,3),3} = 1$. The n variables are all fractional: $\tilde{n}_a = \tilde{w}_{a,4}$ for all $a \in \{(1,2),(2,4),(3,4)\}$ and $\tilde{n}_{(1,3)} = \tilde{w}_{(1,3),3} = 0.9$. This LP solution satisfies (4.20) and (4.24), so they do not improve the strength. Besides, all d-cut inequalities are also satisfied. However, for (4.19), $\tilde{w}_{(1,2),4} = 0.6 > 0.6\tilde{y}_{(1,2),4} = 0.24$, and for (4.26), $\tilde{w}_{(1,3),4} = 0 < 0.6\tilde{y}_{(1,3),4} = 0.36$. In addition, more than half of the constraints (4.25) are

violated since $\tilde{n}_{(1,3)} = 0.9 < \tilde{y}_{(1,3),3} = 1$ and $\tilde{n}_{(3,4)} = 0.4 < \tilde{y}_{(3,4),4} = 1$.

Applying (4.19), (4.20), (4.21), (4.24), and (4.26) (in other words, adding all simple strengthening ineqaulities except for (4.25)), doesn't change the LP bound. However, if (4.25) is also added, the bound, improves from 3.45 to 4.25. The corresponding solution is $\tilde{w}_{(1,3),3} = 0.9$, $\tilde{w}_{(1,2),4} = 0$, $\tilde{w}_{(1,3),4} = 0.6$, $\tilde{w}_{(2,4),4} = 0.5$, $\tilde{w}_{(3,4),4} = 1.0$, $\tilde{y}_{(1,3),3} = 1$, $\tilde{y}_{(1,2),4} = 0$, $\tilde{y}_{(1,3),4} = 1$, $\tilde{y}_{(2,4),4} = 1$, $\tilde{y}_{(3,4),4} = 1$, and $n_{(1,2)} = 0$, $n_{(1,3)} = 1.5$, $n_{(2,4)} = 1$, $n_{(3,4)} = 1$. Note $n_{(1,2)} \geq y_{(1,2),4}$ from Constraint (4.25) is facet defining.

The violated cut-set inequalities for this instance are

$$\begin{aligned} n_{(1,3)} &\geq \lceil 0.9 \rceil = 1, \quad \text{cut-set } \{1,2,4\}, \\ n_{(3,4)} &\geq \lceil 0.4 \rceil = 1, \quad \text{cut-set } \{3\}, \text{ and} \\ n_{(1,2)} + n_{(1,3)} &\geq \lceil 0.9 + 0.6 \rceil = 2, \quad \text{cut-set } \{1\}. \end{aligned} \tag{B.12}$$

The first two of them dominate the strengthening constraints (4.25) for the arcs $(1,3)$ and $(3,4)$ respectively. However, none of them dominate $n_{(1,2)} \geq y_{(1,2),4}$, which is Constraint (4.25) for arc $(1,2)$. If we add all the three inequalities in (B.12), we will obtain the same objective value of 4.25.

We now examine the wheat-stalk inequalities. Consider the inequality formed from $\hat{a} = (1,3)$, with $H = \{4\}$, $\kappa_4 = \{1\}$, and $\tau_4 = \{(1,3)\}$. Note $\underline{K}^{\hat{a}} = \{4\}$, and thus, $L_{\hat{a}} = \lceil q_4 \rceil = 1$. The wheat-stalk inequality is

$$n_{(1,3)} \geq 1 + (\lceil 0.6 + 0.9 \rceil - 1)(y_{(1,3),4} - 1 + 1) = 1 + y_{(1,3),4}, \tag{B.13}$$

which dominates the first cut-set inequality, derived from the cut-set $\{1,2,4\}$. Another wheat-stalk inequality can be formed from $\hat{a} = (2,4)$, with $H = \{4\}$, $\kappa_4 = \{1\}$, and $\tau_1 = \{(1,2),(2,4)\}$. Note $\underline{K}^{\hat{a}} = \{2\}$, and thus, $L_{\hat{a}} = \lceil q_2 \rceil = 1$. The wheat-stalk inequality

is

$$n_{(2,4)} \geq 1 + (\lceil 0.6 + 0.5 \rceil - 1)(y_{(1,2),4} + y_{(2,4),4} - 2 + 1) = y_{(1,2),4} + y_{(2,4),4}. \quad \text{(B.14)}$$

Since $\underline{K}_{(2,4)} = \{2\}$, preprocessing would have fixed $y_{(2,4),4} = 1$, so the wheat-stalk inequality becomes

$$n_{(2,4)} \geq 1 + y_{(1,2),4},$$

which dominates the second cut-set inequality, derived from the cut-set $\{3\}$. To have a clear view of the possible strength of these inequalities, we list the LP relaxation values with different strengthening constraints and valid inequalities added in Table B.1.

Table B.1: LP relaxation values with different strengthening constraints and valid inequalities added.

Simple Strengthening Constraints	None	Cut-set (B.12)	Wheat-stalk (B.13), (B.14)	Cut-set & Wheat-stalk (B.12), (B.13), (B.14)
None	3.45	4.25	3.9	4.75
(4.19), (4.20), (4.21), (4.24), (4.26)	3.45	4.25	4.5	5
(4.19), (4.20), (4.21), (4.24), (4.25), (4.26)	4.25	4.5	5	5

We inspect the convex hull of integer feasible solutions to this instance. Firstly, we simplify the formulation by projecting out all variables that can be fixed using equations valid for this set to get an equivalent polyhedron. After observing that $y_{(1,2),4} + y_{(1,3),4} = 1$ and $w_{(1,2),4} = 0.6y_{(1,2),4}$ are implied by the simple strengthening constraints, it's not difficult to see the convex hull of projected solutions is

$$\text{conv}\left(\begin{array}{c} \{(1,1,2,1,1) + (\eta_1,\eta_2,\eta_3,\eta_4,0) : \eta_1,\eta_2,\eta_3,\eta_4 \geq 0\} \\ \bigcup \{(0,2,1,1,0) + (\eta_1,\eta_2,\eta_3,\eta_4,0) : \eta_1,\eta_2,\eta_3,\eta_4 \geq 0\} \end{array}\right),$$

where projected solutions take the form

$$v = (v_1, v_2, v_3, v_4, v_5) = (n_{(1,2)}, n_{(1,3)}, n_{(2,4)}, n_{(3,4)}, y_{(1,2),4}).$$

The solution $v = (1, 1, 2, 1, 1)$ corresponds to the choice of in-tree for $d = 4$ using arc $(1, 2)$, which has objective value 5.5. The solution $v = (0, 2, 1, 1, 0)$ corresponds to the optimal choice of in-tree for $d = 4$ using arc $(1, 3)$, which has objective value 5 and is optimal for this instance.

It can be shown that both wheat-stalk inequalities are facet defining, taking the first inequality in its equivalent form $n_{(1,3)} \geq 1 + y_{(1,3),4} = 1 + (1 - y_{(1,2),4}) = 2 - y_{(1,2),4}$. The third cut-set inequality $n_{(1,2)} + n_{(1,3)} \geq 2$ is *not* a facet. It is implied by the strengthening inequality (4.25) $n_{(1,2)} \geq y_{(1,2),4}$ together with the first wheat-stalk inequality, $n_{(1,3)} \geq 2 - y_{(1,2),4}$: the two inequalities sum to give $n_{(1,2)} + n_{(1,3)} \geq 2$. Thus, for this instance, all the violated cut-set inequalities are implied by other inequalities and both of the given wheat-stalk inequalities are facet-defining.

REFERENCES

[1] K. Aardal, R. Weismantel, and L. A. Wolsey, "Non-standard approaches to integer programming," *Discrete Applied Mathematics*, vol. 123, no. 1, pp. 5–74, 2002.

[2] T. Achterberg, "Constraint integer programming," PhD thesis, TU Berlin, 2007.

[3] T. Achterberg and T. Berthold, "Hybrid branching," in *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, Springer, 2009, pp. 309–311.

[4] T. Achterberg, T. Berthold, T. Koch, and K. Wolter, "Constraint integer programming: A new approach to integrate CP and MIP," in *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, Springer, 2008, pp. 6–20.

[5] T. Achterberg, T. Koch, and A. Martin, "Branching rules revisited," *Operations Research Letters*, vol. 33, no. 1, pp. 42–54, 2005.

[6] T. Achterberg and R. Wunderling, "Mixed integer programming: Analyzing 12 years of progress," in *Facets of combinatorial optimization*, Springer, 2013, pp. 449–481.

[7] A. M. Alvarez, Q. Louveaux, and L. Wehenkel, "A supervised machine learning approach to variable branching in branch-and-bound," in *IN ECML*, Citeseer, 2014.

[8] ——, "A machine learning-based approximation of strong branching," *INFORMS Journal on Computing*, vol. 29, no. 1, pp. 185–195, 2017.

[9] F. Alvelos and J. M. V. de Carvalho, "Comparing branch-and-price algorithms for the unsplittable multicommodity flow problem," in *Proceedings of the International Network Optimization Conference*, 2003, pp. 7–12.

[10] Y. P. Aneja, "An integer linear programming approach to the steiner problem in graphs," *Networks*, vol. 10, no. 2, pp. 167–178, 1980.

[11] D. Applegate, R. Bixby, V. Chvatal, and B. Cook, "Finding cuts in the tsp (a preliminary report)," Center for Discrete Mathematics & Theoretical Computer Science, Tech. Rep., 1995.

[12] A. Atamtürk, "On capacitated network design cut–set polyhedra," *Mathematical Programming*, vol. 92, no. 3, pp. 425–437, 2002.

[13] A. Atamtürk and D. Rajan, "On splittable and unsplittable flow capacitated network design arc–set polyhedra," *Mathematical Programming*, vol. 92, no. 2, pp. 315–333, 2002.

[14] M.-F. Balcan, T. Dick, T. Sandholm, and E. Vitercik, "Learning to branch," *arXiv preprint arXiv:1803.10150*, 2018.

[15] C. Barnhart, C. A. Hane, and P. H. Vance, "Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems," *Operations Research*, vol. 48, no. 2, pp. 318–326, 2000.

[16] E. M. L. Beale and J. A. Tomlin, "Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables," in *Proceedings 5th IFORS Conference, Tavistock*, J. Lawrence, Ed., Wiley, 1970, pp. 447–454.

[17] E. Beale, "Branch and bound methods for mathematical programming systems," in *Discrete Optimization II*, ser. Annals of Discrete Mathematics, P. Hammer, E. Johnson, and B. Korte, Eds., vol. 5, Elsevier, 1979, pp. 201–219.

[18] A. Benhamiche, A. R. Mahjoub, N. Perrot, and E. Uchoa, "Unsplittable non-additive capacitated network design using set functions polyhedra," *Comput. Oper. Res.*, vol. 66, no. C, pp. 105–115, Feb. 2016.

[19] M. Bénichou, J.-M. Gauthier, P. Girodet, G. Hentges, G. Ribière, and O. Vincent, "Experiments in mixed-integer linear programming," *Mathematical Programming*, vol. 1, no. 1, pp. 76–94, 1971.

[20] T. Berthold and D. Salvagnin, "Cloud branching," in *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, Springer, 2013, pp. 28–43.

[21] D. Bertsimas and J. Dunn, "Optimal classification trees," *Machine Learning*, vol. 106, no. 7, pp. 1039–1082, 2017.

[22] M. Boile, S. Theofanis, A. Baveja, and N. Mittal, "Regional repositioning of empty containers: Case for inland depots," *Transportation Research Record*, vol. 2066, no. 1, pp. 31–40, 2008.

[23] B. Brockmüller, O. Günlück, L. A. Wolsey, *et al.*, "Designing private line networks-polyhedral analysis and computation," Université catholique de Louvain, Center for Operations Research and . . ., Tech. Rep., 1996.

[24] H. Chang, H. Jula, A. Chassiakos, and P. Ioannou, "A heuristic solution for the empty container substitution problem," *Transportation Research Part E: Logistics and Transportation Review*, vol. 44, no. 2, pp. 203–216, 2008.

[25] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[26] S. Chopra, I. Gilboa, and S. T. Sastry, "Source sink flows with capacity installation in batches," *Discrete Applied Mathematics*, vol. 85, no. 3, pp. 165–192, 1998.

[27] M. Chouman, T. G. Crainic, and B. Gendron, "The impact of filtering in a branch-and-cut algorithm for multicommodity capacitated fixed charge network design," *EURO Journal on Computational Optimization*, vol. 6, no. 2, pp. 143–184, 2018.

[28] V. Chvátal, "Hard knapsack problems," *Operations Research*, vol. 28, no. 6, pp. 1402–1411, 1980.

[29] T. G. Crainic, "Service network design in freight transportation," *European Journal of Operational Research*, vol. 122, pp. 272–288, 2000.

[30] H. Crowder, E. L. Johnson, and M. Padberg, "Solving large-scale zero-one linear programming problems," *Operations Research*, vol. 31, no. 5, pp. 803–834, 1983.

[31] H. Crowder and M. W. Padberg, "Solving large-scale symmetric travelling salesman problems to optimality," *Management Science*, vol. 26, no. 5, pp. 495–509, 1980.

[32] G. Dahl, A. Martin, and M. Stoer, "Routing through virtual paths in layered telecommunication networks," *Operations Research*, vol. 47, no. 5, pp. 693–702, 1999.

[33] P. J. Dejax and T. G. Crainic, "Survey paper—a review of empty flows and fleet management models in freight transportation," *Transportation science*, vol. 21, no. 4, pp. 227–248, 1987.

[34] G. Di Liberto, S. Kadioglu, K. Leo, and Y. Malitsky, "Dash: Dynamic approach for switching heuristics," *European Journal of Operational Research*, vol. 248, no. 3, pp. 943–953, 2016.

[35] E. D. Dolan and J. J. More, "Benchmarking Optimization Software with Performance Profiles," *Mathematical Programming*, vol. 91, pp. 201–213, 2002.

[36] Y. Du and R. Hall, "Fleet sizing and empty equipment redistribution for center-terminal transportation networks," *Management Science*, vol. 43, no. 2, pp. 145–157, 1997.

[37] A. L. Erera, J. C. Morales, and M. Savelsbergh, "Robust optimization for empty repositioning problems," *Operations Research*, vol. 57, no. 2, pp. 468–483, 2009.

[38] A. Erera, M. Hewitt, M. Savelsbergh, and Y. Zhang, "Improved load plan design through integer programming based local search," *Transportation Science*, vol. 47, Aug. 2013.

[39] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, IEEE, 1975, pp. 184–193.

[40] M. Fischetti, "Facets of two steiner arborescence polyhedra," *Mathematical Programming*, vol. 51, no. 1-3, pp. 401–419, 1991.

[41] B. Fortz, L. Gouveia, and M. Joyce-Moniz, "Models for the piecewise linear unsplittable multicommodity flow problems," *European Journal of Operational Research*, vol. 261, no. 1, pp. 30–42, 2017.

[42] A. Frangioni and B. Gendron, "0–1 reformulations of the multicommodity capacitated network design problem," *Discrete Applied Math.*, vol. 157, no. 6, pp. 1229–1241, 2009.

[43] V. Gabrel, A. Knippel, and M. Minoux, "Exact solution of multicommodity network optimization problems with general step cost functions," *Oper. Res. Lett.*, vol. 25, no. 1, pp. 15–23, Aug. 1999.

[44] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," *arXiv preprint arXiv:1906.01629*, 2019.

[45] J.-M. Gauthier and G. Ribière, "Experiments in mixed-integer linear programming using pseudo-costs," *Mathematical Programming*, vol. 12, no. 1, pp. 26–47, 1977.

[46] B. Gavish and K. Altinkemer, "Backbone network design tools with economic tradeoffs," *INFORMS Journal on Computing*, vol. 2, pp. 236–252, Aug. 1990.

[47] M. X. Goemans and Y.-S. Myung, "A catalog of steiner tree formulations," *Networks*, vol. 23, no. 1, pp. 19–28, 1993.

[48] R. E. Gomory, "Outline of an algorithm for integer solutions to linear programs and an algorithm for the mixed integer problem," in *50 Years of Integer Programming 1958-2008*, Springer, 2010, pp. 77–103.

[49] M. Hewitt, "Enhanced dynamic discretization discovery for the continuous time load plan design problem," *Transportation Science*, vol. 53, no. 6, pp. 1731–1750, 2019.

[50] A. I. Jarrah, E. Johnson, and L. C. Neubert, "Large-scale, less-than-truckload service network design," *Operations Research*, vol. 57, no. 3, pp. 609–625, 2009.

[51] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*, Springer, 1972, pp. 85–103.

[52] E. B. Khalil, P. Le Bodic, L. Song, G. L. Nemhauser, and B. N. Dilkina, "Learning to branch in mixed integer programming.," in *AAAI*, 2016, pp. 724–731.

[53] T. Koch and A. Martin, "Solving steiner tree problems in graphs to optimality," *Networks: An International Journal*, vol. 32, no. 3, pp. 207–232, 1998.

[54] B. Krishnamoorthy and G. Pataki, "Column basis reduction and decomposable knapsack problems," *Discrete Optimization*, vol. 6, no. 3, pp. 242–270, 2009.

[55] T. L. Magnanti and R. Wong, "Network design and transportation planning: Models and algorithms," *Transportation Science*, vol. 18, pp. 1–55, Feb. 1984.

[56] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica: Journal of the Econometric Society*, pp. 497–520, 1960.

[57] H. Lenstra, Jr., "Integer programming with a fixed number of variables," *Mathematics of Operations Research*, vol. 8, pp. 538–548, 1983.

[58] J. T. Linderoth and M. W. Savelsbergh, "A computational study of search strategies for mixed integer programming," *INFORMS Journal on Computing*, vol. 11, no. 2, pp. 173–187, 1999.

[59] A. Lodi and G. Zarpellon, "On learning and branching: a survey," *Top*, vol. 25, no. 2, pp. 207–236, 2017.

[60] Y. Long, L. H. Lee, and E. P. Chew, "The sample average approximation method for empty container repositioning with uncertainties," *European Journal of Operational Research*, vol. 222, no. 1, pp. 65–75, 2012.

[61] A. Marcos Alvarez, L. Wehenkel, and Q. Louveaux, "Online learning for strong branching approximation in branch-and-bound," University of Liege, Tech. Rep., 2016.

[62] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell, "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning," *Discrete Optimization*, vol. 19, pp. 79–102, 2016.

[63] G. Pataki and M. Tural, "Basis reduction methods," in *Wiley Encyclopedia of Operations Research and Management Science*, J. J. Cochran, L. A. Cox Jr., P. Keskinocak, J. P. Kharoufeh, and J. C. Smith, Eds., 2011.

[64] D. Pisinger, "Where are the hard knapsack problems?" *Computers & Operations Research*, vol. 32, no. 9, pp. 2271–2284, 2005.

[65] W. B. Powell and I. A. Koskosidis, "Shipment routing algorithms with tree constraints," *Transportation Science*, vol. 26, no. 3, pp. 230–245, 1992.

[66] C. Raack, A. M. Koster, S. Orlowski, and R. Wessäly, "Capacitated network design using general flow-cutset inequalities," 2007.

[67] C. Raack, A. M. Koster, and R. Wessäly, "On the strength of cut-based inequalities for capacitated network design polyhedra," 2007.

[68] Y. Tang, S. Agrawal, and Y. Faenza, "Reinforcement learning for integer programming: Learning to cut," *arXiv preprint arXiv:1906.04859*, 2019.

[69] É. Tardos, "A strongly polynomial minimum cost circulation algorithm," *Combinatorica*, vol. 5, no. 3, pp. 247–255, 1985.

[70] B. Thiongane, J. Cordeau, and B. Gendron, "Formulations for the nonbifurcated hop-constrained multicommodity capacitated fixed-charge network design problem," *Computers & Operations Research*, vol. 53, pp. 1–8, 2015.

[71] N. Wieberneit, "Service network design for freight transportation: A review," *OR Spectrum*, vol. 30, pp. 77–112, Jan. 2008.

[72] D. Wojtczak, "On strong np-completeness of rational problems," in *International Computer Science Symposium in Russia*, Springer, 2018, pp. 308–320.

[73] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao, "Adapting boosting for information retrieval measures," *Information Retrieval*, vol. 13, no. 3, pp. 254–270, 2010.

[74] Y. Yang, Y. Ridousane, N. Boland, A. Erera, and M. Savelsbergh, "Substitution-based equipment balancing in service networks with multiple equipment types," *Optimization Online 7564*, 2020.

VITA

Yu Yang was born on July 21, 1994, in China. After obtaining his B.S. degree from the School of Mathematical Sciences, Peking University in June 2016, he continued to pursue a Ph.D. in Operations Research at the H. Milton Stewart School of Industrial & Systems Engineering, Georgia Institute of Technology. He completed his Ph.D. study in July 2020. He will join the Department of Industrial and Systems Engineering, University of Florida, as an assistant professor in August 2020.