Branch and Price for Sub-modular Bin Packing

Liding Xu, Claudia d'Ambrosio, Sonia Vanier, Emiliano Traversi

Branch and Price for Sub-modular Bin Packing

Liding Xu, Claudia D'Ambrosio, Sonia Vanier

LIX CNRS, École Polytechnique, Institut Polytechnique de Paris, 91128 Palaiseau, France, liding.xu@polytechnique.edu, dambrosio@lix.polytechnique.fr, sonia.vanier@polytechnique.edu

Emiliano Traversi

LIPN CNRS, Université Sorbonne Paris Nord, 93430 Villetaneuse, France, emiliano.traversi@lipn.univ-paris13.fr

The submodular bin packing (SMBP) problem aims to pack items into a minimal number of bins for which the capacity utilization function is submodular. The SMBP is equivalent to the random-constrained and robust bin packing problem under various conditions. However, due to the combinatorial and nonlinear nature of the underlying optimization problem, it is difficult to solve the SMBP. In this paper, we propose a branch-and-price algorithm to solve this problem. The resulting price subproblems are submodular Knapsack problems, and we propose a tailored exact branch-and-cut algorithm based on piecewise linear relaxation to solve them. To speed up column generation, we develop a hybrid pricing strategy that can replace the exact pricing algorithm with a fast heuristic pricing algorithm. We test our algorithms on instances from the literature. The computational results show the efficiency of our branch-and-price algorithm and the proposed pricing techniques. .

Key words: branch and price; sub-modular bin packing; sub-modular knapsack; piece-wise linear function

1. Introduction

Bin packing (BP) is an important combinatorial optimization problem with applications in various fields, including call centers, healthcare, container shipping, and cloud computing. These applications are typically modeled as BP problems that aim to pack items into a minimum number of bins, with a capacity constraint on each bin. Submodular Bin Packing (SMBP) is a nonlinear variant of the classical linear BP. The SMBP differs from the classical linear BP in the expression of the capacity utilization functions: The capacity utilization function in the classical linear BP is linear with respect to the items, while the capacity

utilization function in SMBP is a linear term plus a square root term, and this function is submodular with respect to the set of items (see Atamtürk and Narayanan (2008)).

In many practical applications of BPs, item sizes are not known before the packaging decision is made. Therefore, uncertainty must be considered in the modeling. We will show that the SMBP formulates BPs with uncertainty under various conditions. Recently, Cohen et al. (2019) sstudied the resource allocation problem for cloud services. The authors model this problem as an SMBP problem and solve it using approximation algorithms. In this paper, we study exact algorithms to solve the SMBP via the Dantzig-Wolfe decomposition approach and exact algorithms to solve the pricing submodular Knapsack problems. Heterogeneous optimization approaches have been used in the literature to model BPs under uncertainty, including stochastic optimization, chance-constrained optimization, robust optimization, and nonlinear optimization models. The SMBP is a nonlinear optimization model with connections to the other three optimization models. In the stochastic optimization model, the objective function is an expected value function. The stochastic BP (SBP) models item sizes as random variables. If capacity constraint violations can be accounted for as a penalty cost, the optimization objective is to minimize the number of utilized bins plus the expected penalty cost (Denton et al. (2010)). In SBP, the two costs must be weighted in a balanced manner to form a single objective. The chance constrained and robust optimization approaches model uncertainty in the problem constraints directly, so capacity constraints are not violated in these models. The chance constraint is a well-known tool for modeling constraints on random variables (Charnes and Cooper (1963)). The BP with chance constraints (BPCC) (Shylo et al. (2013)) considers the BP problem, where the item sizes follow a multivariate distribution and the items in each bin must satisfy the capacity constraint with a certain probability. Robust optimization, in particular distributionally robust optimization, considers the worst case of chance constraints within a given family of distributions (see Ghaoui et al. (2003)). More generally than BPCC, distributionally robust BP (DRBP) allows item sizes to belong to a family of distributions with common properties. Capacity constraints are robustly satisfied with respect to the entire family of distributions (Zhang et al. (2020), Cohen et al. (2019)).

TThe sample-based average approximation method (SAA) is a common approach to solve stochastic, chance constrained and robust optimization problems (Luedtke and Ahmed (2008), Bertsimas et al. (2018)). It approximates the problem as a two- or multi-stage

mixed-integer linear program and computes approximate solutions that converge to an optimal solution in a probabilistic sense. The scalability and accuracy of this approach depend on the number of samples. Some BPCC and DRBP problems are actually equivalent to SMBPs under various conditions, so these problems can be solved using deterministic methods.

In Cohen et al. (2019) it is shown that BPCC is equivalent to SMBP when the item sizes follow independent Gaussian distributions. Furthermore, if the distributions of the item sizes have the same mean values and the same diagonal covariance matrix, then DRBP is equivalent to SMBP. If only the first two moments of the distribution are known, then the BPCC can be reformulated as an SMBP problem according to Zhang et al. (2018). The SMBP is also valuable from a practical point of view, as it provides an upper bound for general independent distributions over bounded intervals (note that the SMBP becomes a relaxation, see Cohen et al. (2019)).

The SMBP is a binary nonlinear optimization problem due to the nonlinear submodular capacity utilization function. Zhang et al. (2018) show that any SMBP can be reformulated as a binary second-order conic program (BSOCP), and the reformulation can be strengthened by the extended polymatroid inequalities of Atamtürk and Narayanan (2008). The authors use a branch-and-cut algorithm to solve the SMBP. As with linear BPs, it is challenging to scale the branch-and-cut algorithm for SMBPs. The Dantzig-Wolfe (DW) decomposition is a well-known approach to solving large linear BPs: a linear BP is reformulated into a set-cover formulation based on enumerating all feasible packing patterns; then its continuous relaxation is solved using a column generation approach (Gilmore and Gomory (1961)).The branch-and-price algorithm integrates column generation with the branch-and-bound algorithm and is the state-of-the-art exact method for solving linear BPs (Wei et al. (2020b), Delorme et al. (2016)). TTo solve the SMBP efficiently, we use the DW decomposition and develop a branch-and-price algorithm. In our DW decomposition of the SMBP, we use the classical set-cover reformulation for the linear BP (Gilmore and Gomory (1961)), but with nonlinear pricing problems. The submodular Knapsack problems have linear objective functions, but the capacity utilisation functions are submodular. As with the linear BP, solving the pricing problems involves the most computational effort of the branch-and-price solvers, so many efforts have been made in the past to develop algorithms (Sadykov and Vanderbeck (2013), Wei et al. (2020a)). The pricing problem is

more difficult for SMBPs because there is still no pseudopolynomial algorithm (such as dynamic programming) to solve the submodular Knapsack problems. Therefore, solving pricing problems is crucial to the performance of our branch-and-price algorithm. There are several papers in the literature on exact algorithms for variants of the classical Knapsack problem (Cacchiani et al. (2022)), e.g., the quadratic Knapsack (Caprara et al. (1999), Furini and Traversi (2019)), the multidimensional Knapsack (Puchinger et al. (2010)), the quadratic multi-knapsack (Bergman (2019), Olivier et al. (2021)). The quadratic Knapsack has a nonlinear objective function, while the submodular Knapsack has a nonlinear submodular function in the constraint. As far as we know, there is no tailored exact algorithmic framework and implementation for the submodular Knapsack problem either. Moreover, the submodular Knapsack problem is important in its own right as it models the chance-constrained Knapsack problem (Goyal and Ravi (2010)).

We propose a non-convex Mixed Binary Quadratically Constrained Programming (MBQCP) formulation for the submodular Knapsack problem. Based on this formulation, we construct the Piece-Wise Linear (PWL) relaxation and combine the relaxation with cutting planes to form an exact PWL relaxation-based Branch-and-Cut (PWL-BC) algorithm. PWL functions have been used to approximate or relax non-convex mixed-integer nonlinear programming (MINLP) problems (Geißler et al. (2012)). A PWL function is linear in each partition of its domain and can be modeled by a Mixed Integer Linear Programming (MILP) formulation (Vielma et al. (2010)). In our experiments, the PWL-BC algorithm is faster than the commercial solver CPLEX and it optimally solves most pricing problems where CPLEX still has a large dual gap. To further speed up the PWL-BC algorithm, we also investigate adaptive PWL relaxation in our experiments. We also propose several strategies to accelerate the convergence of the branch-and-price algorithm, i.e., improve primal and dual bounds. Wei et al. (2020b) and Gleixner et al. (2020) incorporate the Farley bound (see Farley (1990), Vance et al. (1994)) into their algorithms to obtain an early valid dual bound before the termination of the column generation procedure. Namely, the formula for the Farley bound imposes a condition on whether an exact pricing algorithm can improve the current dual bound. If the condition is not satisfied, we do not need an exact pricing algorithm, but can use a fast heuristic pricing algorithm. Therefore, a hybrid pricing strategy is used to speed up column generation in our branch- and-price algorithms. We also adapt a primary column selection primal heuristic from Lübbecke and Puchert (2012).

This heuristic checks whether each new column can be combined with other generated columns to form a feasible solution. There are few publicly available instances for the SMBP problem. Cohen et al. (2019) test their approximation algorithms on real instances from data centers that are not available to the public due to confidentiality constraints. To compare the proposed algorithms, we generate instances according to their description. Finally, by combining the proposed techniques, our branch-and-price algorithm solves more instances and closes more dual gaps than CPLEX. In summary, our contribution in this work is threefold. First, we apply the DW decom- position for the SMBP and develop several techniques for the branch-and-price algorithm (including primal heuristics and the hybrid pricing strategy). Second, for the submodular Knapsack problem, we propose the MBQCP formulation and PWL relaxation and compare them with the formulation and relaxation used by existing solution programs. We then develop an adapted PWL-BC algorithm. Finally, we perform computational experiments on a large number of instances to evaluate the proposed algorithms. The source code and benchmark will be published on our project website.

1.1. Literature Review

his work refers to different areas of literature. We can mention: (i) solution methods for SBPs, BPCCs and DRBPs; (ii) applications of branch-and-price algorithms for MINLPs; (iii) primal heuristics specialized for column generation; (iv) existing approximations and polyhedral results for the submodular Knapsack problem; (v) the outer approximation and the PWL approximation. The surgery scheduling problem (also known as operating room scheduling) is a common application of the SBP problem in healthcare, where the surgery duration (item size) is assumed to be stochastic. A number of works (Denton et al. (2010), Batun et al. (2011)) model stochastic surgery scheduling as a stochastic two-stage mixed-integer programming problem: the objective is to minimize the fix and the expected penalty for overtime. In some works (Cardoen et al. (2010), Deng et al. (2019)), only the expected penalty is considered in the models. Shylo et al. (2013) appear to be the first to consider BPCC in surgery scheduling. Assuming that the operation duration follows a multivariate normal distribution, the authors reformulate the problem as a semidefinite program. Decomposition methods are already used together with the SSA method to solve SBPs/BPCCs, focusing either on two/multi-level structures of stochastic programs or on BP problem structure. Denton et al. (2010), Batun et al. (2011)use Bender

decomposition to solve SBPs. Zhang et al. (2020) apply DW decomposition to scenario-based subproblems of BPCC (obtained using the SAA method) and solve the subproblems using the branch-and-price algorithm. In addition, Zhang et al. (2020) consider the DRBP where the distributions of item sizes are ambiguous, i.e., the family of distributions is unknown or at best partially known. The authors approximate the problem by an MILP and solve it using the branch-and-price algorithm. As far as we know, the DW decomposition is still applied to MILPs coming from the approximation or relaxation of BPs with uncertainty, but not yet for an exact nonlinear programming model. Recently, DW decomposition and the branch-and-price algorithm have been used to solve MINLPs (see Allman and Zhang (2021)), such as recursive circle packing (RCP) problems (Gleixner et al. (2020)), binary quadratic problems (Ceselli et al. (2022)),nd facility location with general nonlinear facility cost functions (Ni et al. (2021)). There may be several ways to divide a MINLP into main and subproblems, so that a MINLP may admit different DW decompositions. Ceselli et al. (2022) study the strengths of different DW decompositions for binary quadratic problems. In most cases, after applying the DW decomposition to the compact MINLP formulation, the master problem is a MILP and the pricing problems are MINLPs. Allman and Zhang (2021) directly use a commercial MINLP solver, i.e., BARON (Tawarmalani and Sahinidis (2005)), to solve the pricing problems. In our experiments, we also try to solve the pricing problems using a commercial solver, i.e., CPLEX. However, commercial solvers may not explore the structures of MINLPs. Since pricing problems can be solved in thousands of iterations, Gleixner et al. (2020)shows that any improvement in the pricing algorithm can speed up the convergence of column generation. There are several ways to improve the performance of the branch-and-price algorithm, such as fast primal heuristics during column generation and fast pricing algorithms. Primal heuristics improve the primal bounds of the branch-and-price algorithms. Joncour et al. (2010) propose a constructive approach to create feasible solutions from scratch (namely, column selection) using only knowledge of previously generated columns. Column selection heuristics include greedy or relaxation-based approaches. The greedy heuristic is so fast that it can be invoked during column generation. Column selection heuristics are implemented in the generic branch-and-price solver GCG (Gamrath and Lübbecke (2010), Lübbecke and Puchert (2012)). Our implementation is similar to the greedy column selection approach, but we enforce the last column in the feasible solution. The submodular Knapsack problem is studied in

several ways, we refer to Goyal and Ravi (2010) for approximation algorithms Atamtürk and Narayanan (2009) for polyhedral analysis. The submodular Knapsack problem can be reformulated as a binary second-order conic program (BSOCP) (Atamtürk and Narayanan (2008)), which can be solved by general-purpose solvers. Most solvers implement the LP outer approximation-based branch-and-cut LP-B&C) algorithm (Coey et al. (2020)) to solve the BSOCP or general mixed integer second order conic programming (MISOCP) problems, such as `CPLEX` (Bliek et al. (2014)) and `SCIP` (Berthold et al. (2012)). The LP outer approximation is sometimes referred to as the polyhedral outer approximation. Ben-Tal and Nemirovski (2001) prove that any second-order conic program (SOCP) is polynomially reducible to a linear program, and their result justifies the error analysis of the polyhedral outer approximation for SOC -representable sets. Moreover, this result implies the convergence rate of the LP -BC algorithm based on the polyhedral outer approximation for MISOCPs/BSOCPs.

D'Ambrosio et al. (2012), Geißler et al. (2012) propose exact approaches to solve some nonconvex MINLPs using PWL relaxations. For example, D'Ambrosio et al. (2012) obtain a convex MINLP relaxation for nonconvex MINLPs with separable nonconvex functions. The authors distinguish between convex and concave parts and then convexify the concave parts by PWL functions. Moreover, D'Ambrosio et al. (2019) propose perspective reformulations/cuts to strengthen the resulting convex MINLP relaxation. In this study, the original formulation for the submodular Knapsack problem is a convex MINLP and its nonlinear function includes all problem variables, which is difficult to approximate in high-dimensional problems. We reformulate this formulation into a nonconvex MINLP. This nonconvex MINLP has a concave quadratic constraint and the nonlinear term is just a univariate quadratic function on a relaxed variable. We relax the quadratic function into a PWL function (Vielma et al. (2010)), and obtain a MILP relaxation. The resulting relaxation leads to an algorithm with better performance. Our approach is counterintuitive because it is generally assumed that convex MINLP formulations perform better than non-convex MINLP formulations. In our case, we will show that the quadratic function can be approximated in a "dimension-free" way, since the nonlinearity is concentrated on a single variable.

1.2. Outline

This paper is organized as follows. In 2, we describe the SMBP and give its BSOCP and set-cover formulations. In 3, we introduce the key components of our branch-and-price algorithm: the branching rule, column generation, dual bound computing, initial columns, and primal heuristic. 4, we focus on solving the pricing problem: the heuristic pricing algorithm, reformulations of the pricing problem, PWL relaxation, the exact pricing algorithm, and the hybrid pricing strategy. In 5, we show the computational results of the proposed algorithms for instances generated from the literature and analyze their performance. In 6, we conclude this paper with a conclusion and future research directions.

2. Problem Description and Formulations

In this section, we present existing formulations for the SMBP and propose a new formulation. The SMBP problem considers a finite set $\mathcal{N}$ of items and a finite set $\mathcal{M}$ of potential bins, each of which has identical capacity. The problem is to (i) determine a minimum number of bins to pack all items; (ii) allocate items to bins such that all submodular capacity constraints are satisfied.

2.1. Compact Formulations

We begin by describing two compact formulations for the SMBP. To facilitate the description, we first define the following notation:

Set notation

- $\mathcal{N} := \{1, \ldots, n\}$: the index set of items;
- $\mathcal{M} := \{1, \ldots, m\}$: the index set of potential bins.

Now we can set the model parameters and define the submodular capacity utilisation function.

Sub-modular capacity usage function

Given a ground set $\mathcal{N}$, a set function $f : 2^{\mathcal{N}} \to \mathbb{R}$ is *sub-modular*, if

$$\gamma_i(T_1) \geq \gamma_i(T_2), \qquad \forall T_1 \subset T_2 \subset \mathcal{N} \setminus \{i\}, \forall i \in \mathcal{N},$$

where $\gamma_i(T) := f(T \cup \{i\}) - f(T)$ is the incremental function of f on $T \subset \mathcal{N}$.

Every subset of $\mathcal{N}$ can be indicated by a binary vector $x \in \{0,1\}^{\mathcal{N}}$. The sub-modular capacity usage function f in the SMBP is defined as follows (Cohen et al. (2019)):

$$f(x) := \sum_{i \in \mathcal{N}} a_i x_i + \sigma \sqrt{\sum_{i \in \mathcal{N}} b_i x_i}, \tag{1}$$

where $a_i, b_i \in \mathbb{R}_+$ $(i \in \mathcal{N})$ and $\sigma \in \mathbb{R}_+$. The function f becomes a linear function by setting $\sigma = 0$.

The following variables are used to model the SMBP:

Decision variables

- $y_j = \begin{cases} 1, & \text{if bin } j \text{ is used} \\ 0, & \text{otherwise} \end{cases}$, for $j \in \mathcal{M}$;
- $v_{ij} = \begin{cases} 1, & \text{if item } i \text{ is assigned to bin } j \\ 0, & \text{otherwise} \end{cases}$, for $i \in \mathcal{N}, j \in \mathcal{M}$.

Using the above notation, the SMBP has the following compact and non-convex MINLP formulation:

$$\min \quad \sum_{j \in \mathcal{M}} y_j, \tag{2a}$$

$$\text{s. t.} \quad \sum_{i \in \mathcal{N}} a_i v_{ij} + \sigma \sqrt{\sum_{i \in \mathcal{N}} b_i v_{ij}} \leq c y_j, \qquad \forall j \in \mathcal{M}, \tag{2b}$$

$$\sum_{j \in \mathcal{M}} v_{ij} = 1, \qquad \forall i \in \mathcal{N}, \tag{2c}$$

$$v_{ij} \in \{0, 1\}, \qquad \forall i \in \mathcal{N}, j \in \mathcal{M}, \tag{2d}$$

$$y_j \in \{0, 1\}, \qquad \forall j \in \mathcal{M}. \tag{2e}$$

Regarding the continuous relaxation of the above formulation, constraint (2b) is non-convex, so formulation (2) is a non-convex MINLP.

Zhang et al. (2018) gives a convex MINLP, more precisely, BSOCP reformulation for (2) by replacing (2b) with an equivalent constraint:

$$\sum_{i \in \mathcal{N}} a_i v_{ij} + \sigma \sqrt{\sum_{i \in \mathcal{N}} b_i v_{ij}^2} \leq c y_j, \qquad \forall j \in \mathcal{M}, \tag{3}$$

for which the continuous relaxation is representable by second order cones and thus convex. For any feasible solution $v_{ij} \in \{0,1\}$ we have $v_{ij}^2 = v_{ij}$, so the constraints (3) and (2b) are equivalent.

Then, the compact and convex BSOCP formulation is

$$\min \quad \sum_{j \in \mathcal{M}} y_j, \tag{4a}$$

$$\text{s. t.} \quad (3),\ (2c),\ (2d),\ (2e). \tag{4b}$$

The BSOCP formulation can be solved with commercially available solvers, such as `CPLEX` [1] (Bonami and Tramontani (2015)) and `SCIP` (Gamrath et al. (2020)). When $\sigma = 0$, the BSOCP formulation becomes the conventional compact formulation for the linear BP. Indeed, `CPLEX` and `SCIP` solve the BSOCP problem in a non-compact way, since they use the LP -B&C algorithm, which linearizes the BSOCP problem into a MILP model with numerous cutting planes. As in the linear BP case, we find in experiments (Section 5) that the compact BSOCP formulation is not scalable for large models.

2.2. Set Cover Formulation

In this section, we propose a new set-cover formulation for the SMBP. The formulation is derived in a similar way as the DW decomposition of the classical linear BP (Delorme et al. (2016)). This formulation can be solved efficiently by a branch-and-price algorithm.

A column p is defined by a binary vector as $(d_{1p}, d_{2p}, \ldots, d_{np})$, where $d_{ip} = 1$ if item i is contained in the column p. A column is called *feasible* if the combination of its items can fit into a bin, i.e., satisfies the submodular capacity constraint (2b). The quantity coverage formulation is based on enumerating all feasible columns, the number of which can be exponential to the number of items. Next, we define the following notation:

Set notation

- $\mathcal{P}$: the set of all feasible columns.

Decision variables

[1] From version 12.6.2, CPLEX can solve the SOCP problems represented in certain special forms, i.e., second-order cones and twisted second-order cones. The algorithm uses the outer approximation of the BSOCP problem, which is a LP -B&C algorithm. The constraint (3) can be translated into these forms

- $\lambda_p = \begin{cases} 1, & \text{if column } p \text{ is used by the solution} \\ 0, & \text{otherwise} \end{cases}$ for $p \in \mathcal{P}$.

We obtain the following set cover formulation for the SMBP:

$$\min \quad \sum_{p \in \mathcal{P}} \lambda_p, \tag{5a}$$

$$\text{s. t.} \quad \sum_{p \in \mathcal{P}} d_{ip} \lambda_p \geq 1, \qquad \forall i \in \mathcal{N}, \tag{5b}$$

$$\lambda_p \in \{0, 1\}, \qquad \forall p \in \mathcal{P}. \tag{5c}$$

The set cover constraint (5b) specifies that each item i ($i \in \mathcal{N}$) is contained in at least one bin. As far as we know, the set cover formulation for the SMBP has not yet been proposed and solved in the literature.

The two compact formulations (2) and (4) are MINLPs, but the set-cover formulation (5) is an MILP. Moreover, the number of nonlinear constraints in the compact formulations is equal to the number of potential bins. The nonlinearity of the set-cover formulation is *de facto* 'hidden' in the pricing subproblems (Section 3.2), and each pricing subproblem has only one nonlinear constraint.

The following theorem shows that the set-cover formulation is stronger than the BSOCP formulation:

PROPOSITION 1. *The linear relaxation of the set cover formulation is tighter than the continuous SOCP relaxation of the BSOCP formulation.*

The proof can be found in Section A of the appendices.

3. Branch and Price

It is challenging to solve the set cover formulation with an exponential number of binary variables. In this section, we present an exact branch-and-price algorithm to solve the set-cover 315 formulation for the SMBP. The branch-and-price algorithm integrates column generation with the branch-and-bound algorithm to efficiently solve the LP relaxation. In the following subsections, we describe the important steps of our branch-and-price algorithm: the branch rule, column generation, initial columns, dual bound computation, and primal heuristics.

3.1. Branching Rule

Our branch-and-price algorithm uses the Ryan/Foster branching rule (Foster and Ryan (1976)).

The branching rule selects a pair of items $i_1 \in \mathcal{N}$ and $i_2 \in \mathcal{N}$ that must either be packed together or not packed together.

We denote by

- $\mathcal{S}$: the set of item pairs that are forced to be packed together such that if a column p respects $\mathcal{S}$, then for $i_1, i_2 \in \mathcal{S}$, $d_{i_1 p} = d_{i_2 p}$;
- $\mathcal{D}$: the set of item pairs that are not allowed to be packed together such that if a column p respects $\mathcal{D}$, then for $i_1, i_2 \in \mathcal{D}$, $d_{i_1 p} + d_{i_2 p} \leq 1$.

Indeed, $(\mathcal{S}, \mathcal{D})$ exactly describes the branching decisions made for each node of the search tree. We denote by

$$\mathcal{P}_{\mathcal{S},\mathcal{D}} := \{p \in \mathcal{P} \,|\, \forall (i_1, i_2) \in \mathcal{S}\; d_{i_1 p} = d_{i_2 p} \wedge \forall (i_1, i_2) \in \mathcal{D}\; d_{i_1 p} + d_{i_2 p} \leq 1\}$$

the set of feasible columns respecting branching constraints induced by $(\mathcal{S}, \mathcal{D})$. We refer to $\mathcal{P}_{\mathcal{S},\mathcal{D}}$ as the $(\mathcal{S}, \mathcal{D})$-feasible columns.

At each node of the search tree, the set cover problem (5) is restricted to the branching decision set $(\mathcal{S}, \mathcal{D})$, i.e., it follows as

$$\min \quad \sum_{p \in \mathcal{P}_{\mathcal{S},\mathcal{D}}} \lambda_p, \tag{6a}$$

$$\text{s. t.} \quad \sum_{p \in \mathcal{P}_{\mathcal{S},\mathcal{D}}} d_{ip} \lambda_p \geq 1, \qquad \forall i \in \mathcal{N}, \tag{6b}$$

$$\lambda_p \in \{0, 1\}, \qquad \forall p \in \mathcal{P}_{\mathcal{S},\mathcal{D}}. \tag{6c}$$

The above problem (6) is called the *master problem*, and its LP relaxation is called the *master LP problem*.

3.2. Column Generation

We present a column generation method to solve the master problem LP.

The column generation procedure starts with a subset of $(\mathcal{S}, \mathcal{D})$-realisable columns of the master problem LP, adds columns, and solves the constrained LP iteratively. Given

a subset $\mathcal{P}'_{\mathcal{S},\mathcal{D}}$ of $\mathcal{P}_{\mathcal{S},\mathcal{D}}$, the corresponding restricted LP problem, namely the *Restricted Master LP* (RMLP) problem, is

$$\min \quad \sum_{p\in\mathcal{P}'_{\mathcal{S},\mathcal{D}}} \lambda_p, \tag{7a}$$

$$\text{s. t.} \quad \sum_{p\in\mathcal{P}'_{\mathcal{S},\mathcal{D}}} d_{ip}\lambda_p \geq 1, \qquad \forall i \in \mathcal{N}, \tag{7b}$$

$$\lambda_p \geq 0, \qquad \forall p \in \mathcal{P}'_{\mathcal{S},\mathcal{D}}. \tag{7c}$$

After solving the RMLP, let π_i be the dual variable associated with the i-th constraint (7b). The reduced cost for a column $p \in \mathcal{P}_{\mathcal{S},\mathcal{D}}$ is $r_p := 1 - \sum_{i\in\mathcal{N}} \pi_i d_{ip}$. If there is a column $p \in \mathcal{P}_{\mathcal{S},\mathcal{D}} \setminus \mathcal{P}'_{\mathcal{S},\mathcal{D}}$ whose reduced cost r_p is negative, then adding p to $\mathcal{P}'_{\mathcal{S},\mathcal{D}}$ could reduce the target value of the RMLP. Otherwise, the solution for the RMLP is also optimal for the master problem LP. The column with the most negative reduced cost is determined by solving a pricing problem. The details of the pricing algorithms can be found in Section 4.

Before the column generation procedure is applied to the current node, the items that can only be packed together are combined into the set $\mathcal{S}$ using a preprocessing process. Let the new item set be $\mathcal{N}'$, a', b' be the merged parameters, and the new conflict relation be $\mathcal{D}'$. Preprocessing leads to a smaller pricing problem, which can be formulated as follows: *sub-modular knapsack problem with conflicts*:

$$\max \quad \sum_{i\in\mathcal{N}'} \pi'_i x_i, \tag{8a}$$

$$\text{s. t.} \quad \sum_{i\in\mathcal{N}'} a'_i x_i + \sigma\sqrt{\sum_{i\in\mathcal{N}'} b'_i x_i} \leq c, \tag{8b}$$

$$x_{i_1} + x_{i_2} \leq 1, \qquad \forall (i_1, i_2) \in \mathcal{D}', \tag{8c}$$

$$x_i \in \{0,1\}, \qquad \forall i \in \mathcal{N}'. \tag{8d}$$

If the optimal value $\sum_{i\in\mathcal{N}'} \pi'_i x_i > 1$, then the corresponding column has negative reduced cost and is added to the RMLP. Otherwise, the solution of the RMLP is an optimal solution of the master LP, and the current node is solved.

3.3. Initial Columns

We initialize the branch-and-price algorithm with a set $\mathcal{P}'$ of feasible columns. These feasible columns are computed by an approximation algorithm that also provides the number of feasible bins and a warm start for the compact formulation (5). The approximate solution guarantees that the number is at most equal to a fixed ratio to the optimal number of bins.

The approximate algorithm, namely the greedy min-utilization algorithm, greedily allocates items to bins so that the capacity is used as little as possible.

The algorithm manages the following quantities:

- $\mathcal{L}$: a list of existing bins, which is initially empty;
- Γ: a set of unpacked items, which is initially $\mathcal{N}$.

For each existing bin $p \in \mathcal{L}$, its load is expressed by a binary array $(d_{1p}, d_{2p}, \ldots, d_{np})$. If $d_{ip} = 1$ $(i \in \mathcal{N})$, then the item i is packed by p. Therefore, we also treat p as a subset of items in $\mathcal{N}$.

The algorithm updates the load of the existing bins and adds new bins one by one. At each iteration, the algorithm goes through the following steps:

1. for each existing bin $p \in \mathcal{L}$, computes the sum of a and b of packed items respectively, i.e., $A_p := \sum_{i \in \mathcal{N}} d_{ip} a_i$ and $B_p := \sum_{i \in \mathcal{N}} d_{ip} a_i$;

2. for each existing bin $p \in \mathcal{L}$ and each unpacked item $i \in \Gamma$, if the bin p can accommodate the item i, then computes the incremental capacity usage $\gamma_i(p) := (A_p + a_i + \sigma\sqrt{B_p + b_i}) - (A_p + \sigma\sqrt{B_p})$, otherwise, sets $\gamma_i(p) := \infty$;

3. for each unpacked item $i \in \Gamma$, let $p^i := \arg\min_{p \in \mathcal{L}} \gamma_i(p)$ be the bin with the minimum increment capacity usage with respect to i;

4. among the unpacked items Γ, let $i^\star := \arg\min_{i \in \Gamma} \gamma_i(p^i)$ be the item that has the least increment capacity usage;

5. adds the item $i^\star$ to the corresponding bin p^{i^*} by setting $d_{i^\star p^{i^*}} = 1$;

6. if step 4 fails, i.e., existing bins cannot accommodate any unpacked item, adds a new empty bin p' to $\mathcal{L}$, sets $d_{ip'} = 0$ for $i \in \mathcal{N}$, and goes to step 1.

The greedy min-use algorithm can find a solution that uses at most $\frac{8}{3}$ times the number of bins of the optimal solution.

PROPOSITION 2 **(Cohen et al. (2019))**. *The greedy min-usage algorithm is a $\frac{8}{3}$-ratio approximation algorithm.*

3.4. Dual Bound Computing

For an optimization problem, a dual bound certifies the optimality of a solution. In the branch-and-price environment, a local dual bound at each node of the search tree is a lower bound on the optimum of the master problem (6). The local dual bound is used by the algorithm to fathom the node or select branch nodes.

The optimum of the master problem LP is a local dual bound. However, to converge to this optimum, the column generation procedure usually needs to solve a large number of pricing problems. Namely, at each iteration of the column generation procedure, another local dual bound is available. This bound is referred to in the literature as *Farley bound*. The following lemma illustrates how this bound can be computed.

LEMMA 1 **(Farley (1990), Vance et al. (1994))**. *Let v_{MP} be the optimum of the master LP, let v_{RMLP} be the optimum of the RMLP, let v_{price} be a dual bound for the pricing problem* (8)*, and let $v_{\mathrm{F}} := \frac{v_{\mathrm{RMLP}}}{v_{\mathrm{price}}}$ be the Farley bound. Then, $v_{\mathrm{F}} \leq v_{\mathrm{MP}}$, and thus v_{F} is a local dual bound.*

The computation of the Farley bound requires a dual bound on the pricing problem, obtained using an exact pricing algorithm. The branch-and-price algorithm holds a local lower bound v_{ld} at each node of the search tree. After solving each pricing problem, the branch-and-price algorithm updates v_{ld} according to the following rule:

$$v_{\mathrm{ld}} = \max\{v_{\mathrm{F}}, v_{\mathrm{ld}}\}.$$

3.5. Primal Heuristic

For some difficult SMBP instances, the branch-and-price algorithm may be too computationally intensive, since in the worst case an exponential number of columns could be generated. In this case, the root node RMLP would not converge to an optimum in a reasonable time.

On the other hand, each generated column could be included in a feasible main solution, Therefore, a tailored heuristic is needed to help find primal solutions as soon as a new column is generated.

We propose a primary column selection heuristic similar to the greedy column selection heuristic in Joncour et al. (2010), Lübbecke and Puchert (2012). The column selection is invoked as soon as a column is created.

Unlike the random rounding heuristic, this heuristic uses the coverage constraint structure in (5).

We consider the case of the root knot, the other cases are similar. Similar to the greedy minimal consumption algorithm, the heuristic preserves the following sets:

- $\mathcal{L}$: a list of existing bins, which is initially empty;
- Γ: a set of unpacked items, which is initially $\mathcal{N}$.

Let $\mathcal{P}'$ be the set of generated columns. When a column p is generated, the heuristic has the following steps:

1. adds p to $\mathcal{L}$, and, for each i with $d_{ip} = 1$, removes i from Γ;
2. finds a generated column $p' \in \mathcal{P}'$ such that it packs a maximal number of items in Γ, and adds p' to $\mathcal{L}$;
3. if all items are packed, i.e., $\Gamma = \emptyset$, outputs the solution $\mathcal{L}$, otherwise goes to step 2.

Our column selection heuristic does not require the LP information, but the information of the generated columns. It also differs from classical column selection because classical column selection does not force a column p in the solution.

The remainder of this paper is devoted to efficient methods for solving pricing subproblems, since pricing requires the most computational effort in column generation.

4. Pricing Algorithms

In this section we present solution methods for the pricing problem. The proposed algorithms can be implemented as a stand-alone solver for the submodular Knapsack problem.

We first present a fast heuristic pricing algorithm. We then present two formulations of the submodular Knapsack problem (with conflicts): a convex BSOCP formulation and a non-convex MBQCP formulation. The convex BSOCP formulation is solved in our experiments for a comparative study. The PWL method is a way to relax/approximate a nonlinear function by linear functions in its subdomain. We derive a PWL relaxation of the MBQCP formulation and develop an exact PWL-based branch-and-cut algorithm (PWL-B&C) for the pricing problem.

To speed up column generation, we also present a hybrid pricing strategy that can replace the exact pricing algorithm with a fast heuristic pricing algorithm.

4.1. Heuristic Algorithm

We propose a fast heuristic, the fixing-greedy heuristic. This heuristic is used by the hybrid pricing strategy to speed up the column generation procedure.

The fixing-greedy heuristic is based on the best-fit-greedy algorithm. The best-fit-greedy algorithm adds an item per iteration only if it does not conflict with the previously added items, as long as the capacity is not exceeded. The heuristic keeps

- Δ: the set of items added to the bin, which is initially empty.

At each iteration, the best-fit greedy heuristic has the following steps:

1. computes the sum of a'_i and the sum of b'_i of added items, i.e., $A := \sum_{i\in\Delta} a'_i$ and $B := \sum_{i\in\Delta} b'_i$;
2. for each unadded item $i \in \mathcal{N}' \setminus \Delta$, computes the incremental capacity usage $\gamma_i(\Delta) := (A + a'_i + \sigma\sqrt{B + b'_i}) - (A + \sigma\sqrt{B})$;
3. for each unadded item i, computes the profit-over-usage ratio $r_i := \frac{c'_i}{\gamma_i(\Delta)}$;
4. adds the item with the maximum r_i until none of the items can be added anymore.

The fixing-greedy heuristic enforces, for each time, a item in $\mathcal{N}'$ to be in the solution, runs the best-fit greedy algorithm, and outputs the best solution.

4.2. BSOCP Formulation

The Binary Second Order Conic Programming (BSOCP) formulation for the pricing problem (8) is similar to the BSOCP formulation for the SMBP in (2).

The BSOCP formulation is:

$$\max \quad \sum_{i\in\mathcal{N}'} \pi'_i x_i, \tag{9a}$$

$$\text{s. t.} \quad \sum_{i\in\mathcal{N}'} a'_i x_i + \sigma\sqrt{\sum_{i\in\mathcal{N}'} b'_i x_i^2} \le c, \tag{9b}$$

$$x_{i_1} + x_{i_2} \le 1, \qquad \forall (i_1, i_2) \in \mathcal{D}', \tag{9c}$$

$$x_i \in \{0, 1\}, \qquad \forall i \in \mathcal{N}'. \tag{9d}$$

Where (9b) can be represented by second order conic constraints. The BSOCP formulation is a convex MINLP formulation.

In this section, we analyze the polyhedral outer approximation of the BSOCP formulation and show that a finite number of cutting planes is sufficient to define an exact MILP reformulation of the BSOCP formulation.

To simplify the presentation, we use the following notation:

- the left-hand side of (9b):

$$f(x) := \sum_{i\in\mathcal{N}'} a'_i x_i + \sigma\sqrt{\sum_{i\in\mathcal{N}'} b'_i x_i^2};$$

- the binary set defined by (9b):

$$\mathcal{C} := \{x \in \{0,1\}^{\mathcal{N}'} : f(x) \leq c\};$$

- the continuous relaxation of $\mathcal{C}$:

$$\overline{\mathcal{C}} := \{x \in [0,1]^{\mathcal{N}'} : f(x) \leq c\}.$$

Since f is convex, $\overline{\mathcal{C}}$ is convex. We also note that the convex hull of $\mathcal{C}$ is a polytope.

A set $\mathcal{O}$ is said to be a polyhedral outer approximation of $\mathcal{C}$, if $\mathcal{O}$ is a polyhedron and $\mathcal{C} \subset \mathcal{O}$.

A polyhedral outer approximation can be constructed as follows. Define a linearization of f at some $\hat{x}$ in the domain of f by $\mathcal{L}^f_{\hat{x}}(x) \coloneqq f(\hat{x}) + \nabla f(\hat{x})^\top (x - \hat{x})$. Since f is convex, $\mathcal{L}^f_{\hat{x}}$ is an under-estimator of f, i.e., $\mathcal{L}^f_{\hat{x}}(x) \leq f(x)$ for any x. Hence, $\mathcal{L}^f_{\hat{x}}(x) \leq c$ is a linear inequality valid for $f(x) \leq c$.

The following theorem indicates that it suffices to describe $\mathcal{C}$ with a finite number of valid inequalities and binary constraints.

THEOREM 1. *Given a point $\hat{x} \in \{0,1\}^{\mathcal{N}'}$, the following inequality is valid for $\mathcal{C}$ and $\overline{\mathcal{C}}$:*

$$\sum_{i \in \mathcal{N}'} a'_i x_i + \frac{\sigma}{\sqrt{\sum_{i \in \mathcal{N}'} b'_i \hat{x}_i}} \sum_{i \in \mathcal{N}'} b'_i \hat{x}_i x_i \leq c. \tag{10}$$

Moreover,

1. *if $\hat{x} \notin \mathcal{C}$, the valid inequality is violated by $\hat{x}$;*
2. *Let*

$$\mathcal{O} = \left\{ x \in [0,1]^{\mathcal{N}} : \sum_{i \in \mathcal{N}'} a'_i x_i + \frac{\sigma}{\sqrt{\sum_{i \in \mathcal{N}'} b'_i \hat{x}_i}} \sum_{i \in \mathcal{N}'} b'_i \hat{x}_i x_i \leq c,\ \forall \hat{x} \in \{0,1\}^{\mathcal{N}'} \setminus \mathcal{C} \right\}.$$

Then, $\mathcal{C} = \mathcal{O} \cap \{0,1\}^{\mathcal{N}'}$.

Proof. Since function f is sub-modular and hence convex, it follows that

$$\mathcal{L}^f_{\hat{x}}(x) \leq f(x) \leq c.$$

Moreover,

$$\begin{aligned}
&\mathcal{L}^f_{\hat{x}}(x)\\
=&f(\hat{x}) + \nabla f(\hat{x})^\top (x - \hat{x})\\
=&\sum_{i\in\mathcal{N}'} a'_i x_i + \sigma\sqrt{\sum_{i\in\mathcal{N}'} b'_i \hat{x}_i^2} + \frac{\sigma}{\sqrt{\sum_{i\in\mathcal{N}'} b'_i \hat{x}_i^2}} \sum_{i\in\mathcal{N}'} b'_i \hat{x}_i (x_i - \hat{x}_i)\\
=&\sum_{i\in\mathcal{N}'} a'_i x_i + \sigma\sqrt{\sum_{i\in\mathcal{N}'} b'_i \hat{x}_i^2} + \frac{\sigma}{\sqrt{\sum_{i\in\mathcal{N}'} b'_i \hat{x}_i^2}} \sum_{i\in\mathcal{N}'} b'_i \hat{x}_i x_i - \frac{\sigma}{\sqrt{\sum_{i\in\mathcal{N}'} b'_i \hat{x}_i^2}} \sum_{i\in\mathcal{N}'} b'_i \hat{x}_i \hat{x}_i\\
=&\sum_{i\in\mathcal{N}'} a'_i x_i + \frac{\sigma}{\sqrt{\sum_{i\in\mathcal{N}'} b'_i \hat{x}_i}} \sum_{i\in\mathcal{N}'} b'_i \hat{x}_i x_i
\end{aligned}$$

where the last equation follows from the fact that $\hat{x}$ is binary.

Therefore, inequality (10) in the statement is valid for $\mathcal{C}$.

The left hand side of inequality (10) evaluated at $\hat{x}$ is $\sum_{i\in\mathcal{N}'} a'_i \hat{x}_i + \sigma\sqrt{\sum_{i\in\mathcal{N}'} b'_i \hat{x}_i}$ which is by hypothesis is at least c, so $\hat{x}$ violates the inequality.

Assume $x^* \in \{0,1\}^{\mathcal{N}'}$.

If $x^* \notin \mathcal{C}$, then x^* violates the $\mathcal{L}^f_{x^*}(x) \le c$ which is a facet defining inequality of $\mathcal{O}$, then $x^* \notin \mathcal{O}$. Hence, $x^* \in \mathcal{O}$ implies that $x^* \in \mathcal{C}$.

If $x^* \in \mathcal{C}$, since $\mathcal{O}$ is a polyhedral outer approximation of $\mathcal{C}$, x^* must be in $\mathcal{O}$. Therefore, $\mathcal{C} = \mathcal{O} \cap \{0,1\}^{\mathcal{N}'}$. $\square$

Define the *generating set* of the cuts in Theorem 1 by

$$\mathcal{X} := \{\hat{x} \in \{0,1\}^{\mathcal{N}'} : \hat{x} \notin \mathcal{C}\}, \tag{11}$$

and define the following *cut coefficient set* generated by $\mathcal{X}$

$$\Theta := \left\{ \theta \in \mathbb{R}^{\mathcal{N}'} : \exists \hat{x} \in \mathcal{X}\ \forall i \in \mathcal{N}\ \theta_i = a'_i + \frac{\sigma}{\sqrt{\sum_{i\in\mathcal{N}'} b'_i \hat{x}_i}} b'_i \hat{x}_i \right\}. \tag{12}$$

The BSOCP formulation is equivalent to the following MILP formulation

$$\begin{aligned}
\max \quad & \sum_{i\in\mathcal{N}'} \pi'_i x_i, && && \text{(13a)}\\
\text{s. t.} \quad & \theta^\top x \le c, && \forall \theta \in \Theta && \text{(13b)}\\
& x_{i_1} + x_{i_2} \le 1, && \forall (i_1, i_2) \in \mathcal{D}', && \text{(13c)}\\
& x_i \in \{0,1\}, && \forall i \in \mathcal{N}'. && \text{(13d)}
\end{aligned}$$

However, $\mathcal{X}$ (and hence Θ) is unknown before the search space is explored, and its cardinality may be exponential. In practice, the cuts corresponding to Θ can only be separated *lazily*, i.e., a cut is added until a point $\hat{x} found in \mathcal{X}$. This finite family of cuts cannot be used by a standard solver, but it is a key component for constructing our PWL-B&C algorithm in Section 4.5.

The following lemma explains the approximation error of the polyhedral outer approximation $\mathcal{O}$ w.r.t. $\overline{\mathcal{C}}$.

LEMMA 2 **(Ben-Tal and Nemirovski (2001))**. *Let $\epsilon > 0$, then there exists a method to construct a polyhedral outer approximation $\mathcal{O}$ of $\overline{\mathcal{C}}$ with additional $\mathcal{O}(1)|\mathcal{N}'|\log(\frac{1}{\epsilon})$ variables and constraints, such that the relative ℓ_∞ approximation error $\max_{x\in\mathcal{O}} |\sum_{i\in\mathcal{N}'} a'_i x_i + \sigma\sqrt{\sum_{i\in\mathcal{N}'} b'_i x_i^2} - c|/c$ is at most ϵ.*

Note that the approximation error of the polyhedral outer approximation depends on the number of variables.

4.3. MBQCP Formulation

We present a non-convex Mixed Binary Quadratically Constrained Programming (MBQCP) formulation for the submodular Knapsack problem (with conflicts). Although we do not use this formulation to solve the price subproblems, this formulation inspires PWL relaxation and the PWL-B&C algorithm. Here, we introduce a slack variable w to define the sum $\sum_{i\in\mathcal{N}'} a'_i x_i$. Then our MBQCP formulation becomes the following non-convex MINLP program:

$$\max \quad \sum_{i\in\mathcal{N}'} \pi'_i x_i, \tag{14a}$$

$$\text{s. t.} \quad \sum_{i\in\mathcal{N}'} a'_i x_i = w, \tag{14b}$$

$$\sigma^2 \sum_{i\in\mathcal{N}'} b'_i x_i \le (c-w)^2, \tag{14c}$$

$$x_{i_1} + x_{i_2} \le 1, \qquad \forall (i_1, i_2) \in \mathcal{D}', \tag{14d}$$

$$x_i \in \{0,1\}, \qquad \forall i \in \mathcal{N}', \tag{14e}$$

$$w \in [0, c]. \tag{14f}$$

Although the program contains a concave quadratic constraint (14c), the nonlinearity is only a univariate quadratic function compared to the $|\mathcal{N}'|$-dimensional nonlinear SOC function f in (9b).

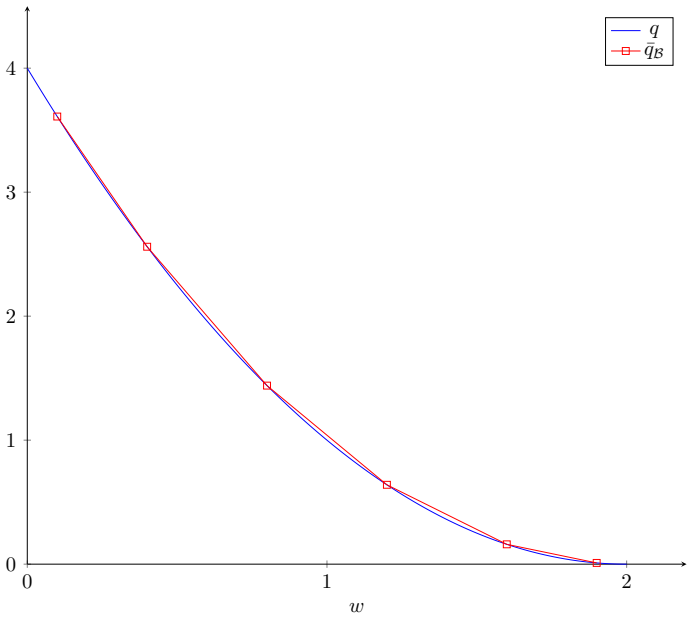

Figure 1 **Graphs of the quadratic and its PWL over-estimator**

4.4. PWL Relaxation

A Piece-Wise Linear (PWL) function is linear on each element of a given partition of its domain. We derive an MILP relaxation of the MBQCP formulation (14) based on PWL functions and refer to this new MILP relaxation as the PWL relaxation. The approximation error of the optimal PWL relaxation is discussed in this section. Denote by $q(w) := (c-w)^2$ the univariate quadratic function. We denote a value of the slack variable w in the constraint (14c) as a *breakpoint*. Given an ordered set of breakpoints $\mathcal{B} = (w_1, w_2, \ldots, w_h)$ such that $w_k \in [\underline{w}, \overline{w}]$ $(k \in [h])$, $w_1 = \underline{w}$ and $w_h = \overline{w}$, the following function is a PWL approximation of q over the domain $[\underline{w}, \overline{w}]$:

$$\bar{q}_{\mathcal{B}}(w) := \frac{q(w_k) - q(w_{k-1})}{w_k - w_{k-1}}(w - w_{k-1}) + q(w_{k-1}), \text{ for } w_{k-1} \le w \le w_k, 2 \le k \le h.$$

Note that $\bar{q}_{\mathcal{B}}$ is an *over-estimator* of q due to the convexity of q.

We call $\mathcal{B}$ a breakpoint set in $[\underline{w}, \overline{w}]$, and $\bar{q}_{\mathcal{B}}$ its induced PWL function. Note that we consider the two bounds $\underline{w}$ and $\overline{w}$ as breakpoints here.

Figure 1 shows the graphs of a quadratic function and its PWL over-estimator, where 495
$\underline{w} = 0.1$, $\overline{w} = 1.9$, $c = 2$ and $\mathcal{B} = \{0.1, 0.4, 0.8, 1.2, 1.6, 1.9\}$.

Replacing $\sigma^2 \sum_{i \in \mathcal{N}'} b_i' x_i \le q(w)$ with $\sigma^2 \sum_{i \in \mathcal{N}'} b_i' x_i \le \bar{q}_{\mathcal{B}}(w)$ in the constraint (14c), we obtain the following PWL relaxation of the MBQCP formulation (14):

$$\max \quad \sum_{i\in\mathcal{N}'} \pi_i' x_i, \tag{15a}$$

$$\text{s. t.} \quad \sum_{i\in\mathcal{N}'} a_i' x_i = w, \tag{15b}$$

$$\sigma^2 \sum_{i\in\mathcal{N}'} b_i' x_i \le \bar{q}_{\mathcal{B}}(w), \tag{15c}$$

$$x_{i_1} + x_{i_2} \le 1, \qquad \forall (i_1, i_2) \in \mathcal{D}', \tag{15d}$$

$$x_i \in \{0,1\}, \qquad \forall i \in \mathcal{N}', \tag{15e}$$

$$w \in [0, c]. \tag{15f}$$

Modeling PWL functions The graphs of PWL functions have several MILP formulations, see Vielma et al. (2010). In this paper we consider the logarithmic model. We denote by z the auxiliary binary variables introduced in the MILP formulation of $\bar{q}_{\mathcal{B}}$. MILP solvers such as `CPLEX` can automatically formulate $\bar{q}_{\mathcal{B}}$ and add auxiliary variables z in the internal data structure.

With a fixed cardinality of $\mathcal{B}$, we aim to minimize the approximation error of the PWL relaxation, and the approximation error is expressed as $p-$norm of the difference between the approximation function and the objective function.

Definition 1. Given a set $\mathcal{B} \subset [\underline{w}, \overline{w}]$ of breakpoints, the ℓ_p approximation error of $\bar{q}_{\mathcal{B}}$ with respect to q over $[\underline{w}, \overline{w}]$ is defined as $\ell_p(\bar{q}_{\mathcal{B}}, q) := (\int_{\underline{w}}^{\overline{w}} |\bar{q}_{\mathcal{B}}(w) - q(w)|^p \, dw)^{\frac{1}{p}}$.

Given an integer h, denote by $\mathbb{B}^h$ the family of breakpoint sets of cardinality h in $[\underline{w}, \overline{w}]$, the *breakpoint selection problem* aims to find a set $\mathcal{B} \in \mathbb{B}^h$ to minimize the ℓ_p error:

$$\min_{\mathcal{B}\in\mathbb{B}^h} \ell_p(\bar{q}_{\mathcal{B}}, q) \tag{16}$$

Geißler et al. (2012) Use a convex program to compute the ℓ_∞-approximation error for general nonconvex functions. Berjón et al. (2015) develop an error analysis that yields asymptotically tight bounds to quantify the ℓ_2-approximation error.

An optimal solution to the breakpoint selection problem under the ℓ_∞-approximation error is an equidistant partition of $[\underline{w}, \overline{w}]$.

THEOREM 2. [2] *Given* $\mathcal{B} \in \mathbb{B}^h$,

$$\ell_\infty(\bar{q}_\mathcal{B}, q) = \max_{w \in [\underline{w}, \overline{w}]} |\bar{q}_\mathcal{B}(w) - q(w)| = \max_{2 \le k \le h} \frac{(w_k - w_{k-1})^2}{4}.$$

Furthermore, let $w_k = \underline{w} + \frac{k-1}{h-1}(\overline{w} - \underline{w})$ *for* $1 \le k \le h$, *which yields the minimum* ℓ_∞*-approximation error* $\frac{(\overline{w}-\underline{w})^2}{4(h-1)^2}$ *for the break point selection problem* (16).

The proof can be found in Section B of the appendices. The approximation error decreases with the quadratic rate with respect to h. The relative ℓ_∞-approximation error is defined as

$$\frac{\ell_\infty(\bar{q}_\mathcal{B}, q)}{(\overline{w} - \underline{w})^2}.$$

We have the following result on the relative approximation error of the PWL relaxation.

COROLLARY 1. *Let* $\epsilon > 0$, *then there exists an MILP formulation of PWL function* $\bar{q}_\mathcal{B}$ *induced by* $\mathcal{B}$ *with* $\mathcal{O}(1)\log(\frac{1}{\epsilon})$ *binary variables and* $\mathcal{O}(1)\frac{1}{\sqrt{\epsilon}}$ *continuous variables and constraints, such that the relative* ℓ_∞*-approximation error is at most* ϵ.

Proof. For the logarithmic model of PWL function, given h breakpoints from the equidistant partition, the relative ℓ_∞-approximation error is $\frac{(\overline{w}-\underline{w})^2}{4(h-1)^2(\overline{w}-\underline{w})^2} = \frac{1}{4(h-1)^2}$ with $\log(h-1)$ binary variables and $h-1$ continuous variables and constraints (Vielma et al. (2010)), the result follows. □

Next, we summarize the approximation errors of all the proposed relaxations to their corresponding formulations. Note that we do not consider the integrality of the binary variable x'. Comparing Lemma 2 and Corollary 1, the approximation error of the PWL relaxation (15) to the MBQCP formulation (14) is independent of the number of variables, while the approximation error of the polyhedral outer approximation to the BSOCP formulation (9) depends on this number.

For a large (possibly exponential) number of breakpoints, the PWL relaxation can also be transformed into an exact reformulation of the MBQCP formulation.

COROLLARY 2. *Let* $\mathcal{B}^* = \{w \in [0, c] : \exists x \in \{0,1\}^{\mathcal{N}'}, w = \sum_{i \in \mathcal{N}'} a'_i x_i\}$. *Then, the PWL relaxation* (15) *derived from* $\mathcal{B}^*$ *is an exact reformulation of the MBQCP formulation* (15).

[2] After the revision process, the authors were made aware of essentially the same result appearing as Lemma 3 in Bärmann et al. (2021) – the two lemmas were proved independently of one another.

Proof. It suffices to prove that given a binary point (x, w) satisfying the PWL relaxation, it is also feasible for the MBQCP formulation.

It is obvious that $w \in \mathcal{B}^*$. Note that for any $w' \in \mathcal{B}^*$ we have $\bar{q}_{\mathcal{B}^*}(w') = q(w')$. Hence, $\bar{q}_{\mathcal{B}^*}(w) = q(w)$.

Since (x, w) satisfies the constraint (15c) in the PWL relaxation, we have $\sigma^2 \sum_{i \in \mathcal{N}'} b'_i x_i \leq \bar{q}_{\mathcal{B}^*}(w) = q(w)$. Hence, (x, w) satisfies the corresponding constraint (14c) in the MBQCP formulation. □

4.5. Exact PWL-B&C Algorithm

The approximation error of the PWL relaxation is dimensionless, but only for a small number of breakpoints it is not exact. Instead of adding many breakpoints, the finite number of cuts induced by the set Θ in (12) suffices to make the PWL relaxation exact. We propose a combined formulation and a branch-and-cut algorithm based on the PWL relaxation (PWL-B&C) to solve it.

$$\max \quad \sum_{i \in \mathcal{N}'} \pi'_i x_i, \tag{17a}$$

$$\text{s. t.} \quad \sum_{i \in \mathcal{N}'} a'_i x_i = w, \tag{17b}$$

$$\sigma^2 \sum_{i \in \mathcal{N}'} b'_i x_i \leq \bar{q}_{\mathcal{B}}(w), \tag{17c}$$

$$\theta^\top x \leq c, \qquad \forall \theta \in \Theta \tag{17d}$$

$$x_{i_1} + x_{i_2} \leq 1, \qquad \forall (i_1, i_2) \in \mathcal{D}', \tag{17e}$$

$$x_i \in \{0, 1\}, \qquad \forall i \in \mathcal{N}', \tag{17f}$$

$$w \in [0, c]. \tag{17g}$$

IThe combined formulation (17) combines the MILP formulation (13) with the (redundant) PWL relaxation, so this formulation is an exact formulation for submodular Knapsack problems with conflicts.

We show in experiments that this formulation with redundant constraints (17b) and (17c) can be solved much faster than the standard BSOCP formulation (9). In practice, only a few cuts in (17d) are required to separate.

Our algorithm consists of three main steps: tightening the bounds, constructing the PWL relaxation (breakpoints), and the PWL B&C algorithm. First, bound tightening

is an upstream procedure used to tighten the bounds on the breakpoints for all pricing problems. Then, the PWL relaxation (breakpoints) is constructed for all pricing problems, and this is also a pre-solving procedure. The construction depends on the number of items, the size of the items, and the capacity. Finally, based on the PWL relaxation, the PWL -B&C algorithm is adapted to the LP -B&C algorithm (Coey et al. (2020)).

Bound tightening The bound tightening procedure is called before the branch-and-price algorithm to reduce the boundaries of the breakpoints $\mathcal{B}$ into $[0, c]$.

Considering a pricing problem at a node of the search tree, we find that if $w = \sum_{i\in\mathcal{N}'} a'_i x_i$ is small, $q(w) = (c-w)^2$ is larger than $\sigma^2 \sum_{i\in\mathcal{N}'} b'_i x_i$, so the capacity constraint (14c) is not active. Thus, there is no need to overestimate q when w is small. More precisely, there is a $\underline{w} \in [0, c]$ such that for any binary solution $x \in \{0,1\}^{\mathcal{N}'}$, let $w = \sum_{i\in\mathcal{N}'} a'_i x_i$, if $w \le \underline{w}$, then $\sigma^2 \sum_{i\in\mathcal{N}'} b'_i x_i \le q(\underline{w})$. Since q is not increasing, $q(w) \ge q(\underline{w}) \ge \sigma^2 \sum_{i\in\mathcal{N}'} b'_i x_i$. The point $\underline{w}$ is called *lower breakpoint*, the submodular capacity constraint (14c) is never violated for $w \in [0, \underline{w}]$. We can start by overestimating q starting from the maximum lower breakpoint computed from the following convex MBQCP problem:

$$
\begin{aligned}
\underline{w} := \max \quad & w, \\
\text{s. t.} \quad & \sum_{i\in\mathcal{N}'} a'_i x_i = w, \\
& \sigma^2 \sum_{i\in\mathcal{N}'} b'_i x_i \ge (c-w)^2, \\
& x_{i_1} + x_{i_2} \le 1, \qquad \forall (i_1, i_2) \in \mathcal{D}', \\
& x_i \in \{0,1\}, \qquad \forall i \in \mathcal{N}'.
\end{aligned}
\tag{18}
$$

Similarly, we can define the *upper breakpoint*. There exists some upper breakpoint $\overline{w} \in [0, c]$, such that for any binary solution $x \in \{0,1\}^{\mathcal{N}'}$, if $\sum_{i\in\mathcal{N}'} a'_i x_i + \sigma\sqrt{\sum_{i\in\mathcal{N}'} b'_i x_i^2} \le c$, then $\sum_{i\in\mathcal{N}'} a'_i x_i \le \overline{w}$. The minimum upper breakpoint can be computed from the following BSOCP problem:

$$
\begin{aligned}
\overline{w} := \max \quad & \sum_{i\in\mathcal{N}'} a'_i x_i, \\
\text{s. t.} \quad & \sum_{i\in\mathcal{N}'} a'_i x_i + \sigma\sqrt{\sum_{i\in\mathcal{N}'} b'_i x_i^2} \le c, \\
& x_{i_1} + x_{i_2} \le 1, \qquad \forall (i_1,i_2)\in\mathcal{D}', \\
& x_i \in \{0,1\}, \qquad \forall i \in \mathcal{N}'.
\end{aligned}
\tag{19}
$$

We solve the above two programs at the root node and obtain the bound $[\underline{w}_r, \overline{w}_r]$ for breakpoints. Since the feasible sets of the other nodes are a subset of the set of the root node, the above programs at other nodes are more strict than those at the root node. It follows for $\underline{w}, \overline{w}$ of any other node that $\underline{w}_r \le \underline{w}$ and $\overline{w} \le \overline{w}_r$. We then set $\underline{w} = \underline{w}_r$ and $\overline{w} = \overline{w}_r$ for all nodes.

construction of breakpoints To determine the number of breakpoints $\mathcal{B}$, we run a greedy heuristic algorithm that tries to maximize the number of elements in a bin. We take h as the solution value given by the heuristic algorithm and assign breakpoints h equidistantly in $[w, \overline{w}]$. Note that for a fixed number of breakpoints, the equidistant partition gives the best approximation error according to Theorem 2. We also add a breakpoint corresponding to $w = 0$.

PWL-B&C algorithm The main steps of the PWL-B&C algorithm are described in Algorithm 1. Recall that the problem (8) is a maximization problem. Algorithm 1 maintains a lower bound L (initially $-\infty$), a set of active nodes $\mathscr{N}$ of the search tree, a pool of cuts $\mathscr{C}$, and an established solution x^*.

A node (l, u, U) is characterized by the finite variable boundary vectors l and u and the upper bound U of the node. The upper bound U is firstly inherited from its parent node and secondly computed via the LP relaxation. We use a MILP solver, i.e. `CPLEX`, to construct the MILP formulation of the PWL function $q_{\mathcal{B}}$, and denote z as the additional binary variables to model $\bar{q}_{\mathcal{B}}$ (see Section 4.4). The variables z are constructed internally by the MILP solver, and we assume that the PWL function is forced when z is set to binary.

We denote by $\mathscr{M}_{\mathcal{B}}(\mathscr{C}, l, u, U)$ the MILP relaxation restricted to finite bounds (l, u) for (x, z) at a node of the search tree. The MILP relaxation $\mathscr{M}_{\mathcal{B}}(\mathscr{C}, l, u, U)$ consists of the PWL relaxation (15), cuts from $\mathscr{C}$, and other cuts added by the MILP solver.

Algorithm 1: PWL-B&C algorithm

Input: a sub-modular knapsack with conflicts in the formulation (17), and the set $\mathcal{B}$ of breakpoints;
Output: a primal solution x^* and a dual upper bound (dual gap);
initialize MILP $\mathscr{M}_{\mathcal{B}}(\mathscr{C}, l_0, u_0, \infty)$ as the PWL relaxation (15);
initialize cut pool $\mathscr{C}$ to $\emptyset$;
initialize node list $\mathscr{N}$ of $\mathscr{M}_{\mathcal{B}}(\mathscr{C}, l_0, u_0, \infty)$ with root node (l_0, u_0);
initialize incumbent solution $x^* = 0$, lower bound L to $-\infty$;
while $\mathscr{N}$ *contains nodes* **do**
    remove a node (l, u) from $\mathscr{N}$;
    solve LP relaxation of $\mathscr{M}_{\mathcal{B}}(\mathscr{C}, l, u, U)$;
    if *LP is infeasible* **then**
        continue ; ▷ fathomed by infeasibility
    get an LP optimal solution $(\hat{x}, \hat{z})$;
    set U to $\sum_{i \in \mathcal{N}'} \pi_i \hat{x}_i$;
    if *upper bound* $U \leq L$ **then**
        continue ; ▷ fathomed by bound
    end
    if $(\hat{x}, \hat{z})$ *is binary* **then**
        if $\hat{x}$ *satisfies capacity constraint* (8b) **then**
            update L to U;
            set x^* to $\hat{x}$;
            continue ; ▷ fathomed by integrality
        else
            add separation cut to $\mathscr{C}$ by Theorem 1;
            add the node $\mathscr{M}_{\mathcal{B}}(\mathscr{C}, l, u, U)$ to $\mathscr{N}$;
            continue ; ▷ reoptimization after cut added
        end
    end
    add branch nodes to $\mathscr{N}$ using $(\hat{x}, \hat{z})$ (fractional) and U;
end

The node set $\mathscr{N}$ initially contains the root node (l^0, u^0), where $l^0, u^0 \in \mathbb{R}^I$ are the finite initial global bounds on variables (x, z). On Line 8 of Algorithm 1, the main loop removes a node (l, u, U) from $\mathscr{N}$. Line 9 solves the LP relaxation of $\mathscr{M}_{\mathcal{B}}(\mathscr{C}, l, u, U)$ by the MILP solver.

If the LP -relaxation $\mathscr{M}_{\mathcal{B}}(\mathscr{C}, l, u, U)$ is infeasible, Line 11 immediately fathoms the node by infeasibility; otherwise, the upper bound U of a node is set to the optimal value of LP on Line 13. The upper bound U of the node means that any feasible solution to the combined formulation (17) that satisfies the bounds of the node for (x, z) has a target value of at most U. Since LP is a relaxation of the combined formulation (17), any feasible solution to the combined formulation (17) that satisfies the bounds of the node for x has a target value of at most U.

Line 15 fathoms the node by Bound if U is not better than the established value L. If $\hat{x}$ is binary and feasible, its solution value U should be at least the lower bound L. Line 19 updates the lower bound L to U, Line 20 stores the new established solution $\hat{x}$, and Line 21 fathoms the node since $\hat{x}$ is an optimal binary solution with respect to the bounds (l, u). If $\hat{z}$ is binary and feasible, then the PWL function is implicitly enforced by the integrality of $\hat{z}$. The condition (17d) is a lazy condition that is disconnected if $\hat{x}$ is binary. Line 23 adds this condition to the cut pool $\mathscr{C}$, Line 24 adds the current node for re-optimization, and Line 25 discards $\hat{x}$ by the cut in the next optimization iteration. Finally, $(\hat{x}, \hat{z})$ must be fractional on Line 28, the algorithm branches using the information from fractionality and U.

4.6. Hybrid Pricing Strategy

The heuristic pricing algorithm in Section 4.1 is fast, but it cannot guarantee the dual upper bound that yields the Farley bound in Lemma 1. The exact pricing algorithm is slow but yields the dual upper bound for pricing. The hybrid pricing strategy first calls the heuristic pricing algorithm to decide whether calling the exact pricing algorithm can improve the local dual bound of the master problem.

In fact, for a heuristic solution, the exact algorithm is required only under a certain condition. The condition is given by the following theorem.

PROPOSITION 3. *Let v_{heur} be the solution value of the heuristic pricing algorithm, let v_{RMLP} be the optimum of RMLP* (7)*, and let v_{ld} be the current local dual bound for the*

master problem. If $\frac{v_{\text{RMLP}}}{v_{heur}} \leq v_{\text{ld}}$*, the exact algorithm cannot yield a better local dual bound than* v_{ld}*.*

Proof. Let v_{popt} be the optimum for the pricing problem (8), then $v_{heur} \leq v_{popt}$. It follows that $\frac{v_{\text{RMLP}}}{v_{popt}} \leq \frac{v_{\text{RMLP}}}{v_{heur}} \leq v_{\text{ld}}$. However, v_{popt} is the smallest pricing dual bound v_{price}, so $\frac{v_{\text{RMLP}}}{v_{popt}}$ is the greatest Farley bound according to Lemma 1. Therefore even if the pricing algorithm is solved to optimality, we cannot obtain a better bound than v_{ld}. □

If the condition $\frac{v_{\text{RMLP}}}{v_{heur}} \leq v_{\text{ld}}$ holds, one can get rid of the exact pricing algorithm, and use the solution from the fixing-greedy heuristic in Section 4.1. The hybrid pricing strategy is outlined in Algorithm 2.

Algorithm 2: Hybrid pricing strategy

1 **Input:** a pricing problem (8) with the objective coefficients π', v_{RMLP} the optimum of RMLP (7), v_{ld} the local dual bound of the master problem;
2 **Output:** a generated column x^*, and the updated local dual bound v_{ld};
3 call the heuristic pricing algorithm with the objective coefficients π' ; ▷ run heuristic first
4 let x, v_{heur} be the heuristic solution and its value;
5 **if** $\frac{v_{\text{RMLP}}}{v_{heur}} \leq v_{\text{ld}}$ *and* $1 - \sum_{i \in \mathcal{N}'} \pi'_i \bar{x}_i < 0$ **then**
6 $\quad x^* \leftarrow x$; ▷ heuristic solution
7 **else**
8 $\quad$ call the exact pricing Algorithm 1 ; ▷ exact pricing
9 $\quad$ let $\tilde{x}, v_{\text{price}}$ be the primal solution and the dual bound;
10 $\quad x^* \leftarrow \tilde{x}$;
11 $\quad v_{\text{ld}} = \max\{v_{\text{ld}}, \frac{v_{\text{RMLP}}}{v_{\text{price}}}\}$; ▷ update the local dual bound
12 **end**

In Line 3, the heuristic algorithm is called first. If $\frac{v_{\text{RMLP}}}{v_{heur}} \leq v_{\text{ld}}$, the exact pricing is not needed. If the heuristic solution x has a negative reduced cost, the strategy outputs it in Line 6. Otherwise, the strategy calls the exact algorithm in Line 8.

5. Computational Experiments

In this section, we present the computational experiments we use to test the effectiveness of our branch-and-price algorithms for the SMBP. In particular, we test different configurations of branch-and-price algorithms to evaluate the proposed techniques. The source code and benchmarks are publicly available on the project website https://github.com/lidingxu/cbp. There we also provide a bash file that you can use to reproduce the experiments on Linux systems.

5.1. Benchmarks

We produce benchmarks as described in Cohen et al. (2019). The authors test their approximation algorithms on benchmarks from real cloud datacenters of `Google`, which are not accessible due to confidentiality. We therefore create new instances using the same generation method.

The authors generate SMBPs to model BPs with the chance constraint

$$\mathbb{P}(\sum_{i\in\mathcal{N}} \mu_i x_{ij} \leq c y_j) \geq \alpha, \tag{20}$$

or BPs with the distributionally robust constraint

$$\inf_{\mu\sim\mathscr{D}} \mathbb{P}(\sum_{i\in\mathcal{N}} \mu_i x_{ij} \leq c y_j) \geq \alpha, \tag{21}$$

where $\mathscr{D}$ is a family of distributions, and μ_i $(i \in \mathcal{N})$ is the nominal size of item i. For different risk levels α, they propose three data generation methods (cases) to construct the data a, b, σ in the SMBP (4), i.e., the Gaussian case,the Hoeffding inequality case, and the distributionally robust approximation case.

As for them, we set the capacity of each bin to 72 (the number of cores of the servers), the risk level $\alpha \in \{0.6, 0.7, 0.8, 0.9, 0.95, 0.99\}$.

We set the number of items (i.e., jobs) $|\mathcal{N}| \in \{100, 400, 1000\}$ to obtain three benchmarks with different sizes: `CloudSmall`, `CloudMedium` and `CloudBig`. There are three generation methods and six risk levels. For each combination of generation methods and risk levels, we generate six instances with different random seeds. As a result, we have $108 = 6\ times6\ times3$ instances in a benchmark. The interested reader can find the detailed generation method in Section C of the appendices.

5.2. Experimental Setups

In this section we describe the setup of the experiments, including the development environment, the implementation of the algorithms, and the solution statistics.

Development environment The experiments are conducted on a computer with Intel Core i7-6700K CPU @ 4.00GHZ and 16GB main memory. We use `SCIP` 7.0.3 (Gamrath et al. (2020)) as a branch-and-price (B&P) framework to solve the set cover formulation (5). We use ILOG `CPLEX` 20.1 as:

- an LP solver to solve the LP relaxation of RMLP (7);
- a BSOCP solver to solve BSOCP formulations of the SMBP (2) and the submodular knapsack problem with conflicts (9);
- an MILP solver used by the PWL-B&C Algorithm 1;

`CPLEX`'s parameters are set by default, except that we disable its parallelism.

Solver implementation We implement five solvers for the SMBP in this work according to the proposed techniques. Four of them are branch-and-price solvers.

These solvers are as follows:

1. a `CPLEX`-based solver for solving the compact BSOCP formulation (2), abbreviated as CPLEX-BSOCP.

2. a `CPLEX`-based solver for solving the BSOCP formulation (9) of the pricing problem, abbreviated as B&P-BSOCP.

3. a branch-and-price solver that uses the PWL-B&C algorithm to solve the combined formulation (17) of the pricing problem, abbreviated as B&P-PWL. The PWL-B&C algorithm calls `CPLEX` to formulate PWL functions and solve the resulting MILP, and cuts are added as *lazy* constraints.

4. a branch-and-price solver that uses the hybrid pricing strategy in Algorithm 2, abbreviated as B&P-hybrid. The hybrid pricing strategy uses the PWL-B&C algorithm as an exact pricing subroutine.

5. a branch-and-price solver that uses the hybrid pricing strategy in Algorithm 2 and the column selection heuristic in Section 3.5, abbreviated as B&P-hybrid*.

We use the approximation algorithm in Section 3.3 to find an initial feasible solution that serves as a warm start for all solvers. The time limit for each solver is 3600 CPU seconds.

If the column generation procedure at the root node does not finish after 3500 CPU seconds, it is halted, giving `SCIP` 100 CPU seconds to invoke its own primary heuristic.

For the pricing problems, we set the same time limit for the exact algorithms ($|\mathcal{N}|$ $times$0.015 CPU seconds) and the same tolerance for relative gaps (the default of `CPLEX`).

Performance metrics and statistical tests In order to evaluate the solver performance in different instances, we compare shifted geometric means (SGMs) (see Achterberg et al. (2008)) of performance metrics.

The SGM of values $v_1, ..., v_N \geq 0$ with shift $s \geq 0$ is defined as

$$\left(\prod_{i=1}^{N}(v_i + s)\right)^{1/N} - s.$$

Given an SMBP problem instance, let $\underline{v}$ be a dual lower bound and $\overline{v}$ be a primal upper bound found by a solver. The relative dual gap in percentage is defined as:

$$\delta_d := \frac{\overline{v} - \underline{v}}{\overline{v}} \times 100.$$

A smaller relative dual gap indicates better performance.

Let v^a be the value of the solution found by the greedy minimal algorithm, which is communicated to all solvers as a warm start. The closed primary bound is defined as:

$$\delta_p := \frac{v^a - \overline{v}}{\max(\overline{v} - \underline{v}^*, 1e^{-6})} \max equation v100,$$

where $\underline{v}^*$ is the largest dual bound found among all solvers. A larger closed primary gap means better performance.

We plot the following performance metrics for each instance tested by each solver and compute the SGMs of the benchmarks:

1. t: the total running time in CPU seconds, with a shifted value set to 1;
2. δ_d%: the relative dual gap in percentage, with a shifted value set to 1%;
3. δ_p%: the closed primal bound in percentage, with a shifted value set to 1%;
4. #N: the number of nodes of the search tree, with a shifted value set to 1;
5. #C: the number of columns generated, with a shifted value set to 1;
6. E%: the percentage of columns generated by the exact pricing algorithm, with a shifted value set to 1%;

7. τ%: the relative dual gap in percentage of a pricing problem solved by an exact algorithm, with a shifted value set to 1%;

8. t_p%: the ratio between pricing time and total solving time in percentage, with a shifted value set to 1%.

Metrics (1)-(4) refer to master problems and are available to all solvers. Metrics (5)-(8) refer to pricing problems and are not available for the CPLEX-BSOCP, while metric (6) is 100% for the B&P-PWL and B&P-BSOCP.

We will discuss the computational results, which are divided into two parts. In the first part, we compare five solvers: the CPLEX-BSOCP, B&P-BSOCP, B&P-PWL, B&P-Hybrid, and B&P-Hybrid*. The second part is based on the observations and analysis of the first part, in which we test whether non-equidistant (adaptive) breakpoints can improve the performance of the B&P-Hybrid* solver.

5.3. Comparative Analysis of Main Results

The main computational results are summarized in Table 1, for detailed results we refer to Section D in the appendices. For each benchmark, we report the SGM statistics of the performance metrics, the number of instances solved (#S) and the number of instances with improved primary bounds (#I).

Next, we analyze the main computational results by comparing the solvers.

Compact formulation vs. set cover formulation For all benchmarks, the B&P hybrid is the best B&P solver in terms of dual gap, total time, and number of instances solved. Therefore, we compare the B&P hybrid with the CPLEX-BSOCP.

We first analyze the results for the small benchmark. The average total time and average dual gap of the CPLEX-BSOCP are 4.34 and 7.86 times that of the B&P hybrid, respectively. The CPLEX-BSOCP solves 18 instances, and this performance is the worst among all solvers. In contrast, the B&C hybrid solves 65 instances. However, the closed primary bound of the CPLEX-BSOCP is 1.93 times larger than that of the B&P hybrid. CPLEX-BSOCP improves the initial approximate solutions for 13 more instances than the B&P hybrid. We analyze the performance of the compact BSOCP formulation (2) and the set-cover formulation (5). We focus on the performance statistics of the master problems.

The CPLEX-BSOCP outperforms all other solvers in terms of finding good primal solutions because `CPLEX` has powerful internal heuristic algorithms.

Benchmarks	Solvers	Master statistics						Pricing statistics			
		t	$\delta_d\%$	$\delta_p\%$	#N	#S	#I	#C	E%	$\tau\%$	$t_p\%$
`CloudSmall` ($\lvert\mathcal{N}\rvert = 100$)	CPLEX-BSOCP	1452	15.8	0.0	26601	18	0	-	-	-	-
	B&P-BSOCP	2129	11.4	0.9	21	20	17	1373	100	3.56	99
	B&P-PWL	633	2.4	2.7	66	61	32	1869	100	0.01	99
	B&P-Hybrid	330	2.0	3.4	127	65	36	3485	18	0.01	96
	B&P-Hybrid*	335	2.1	4.1	114	63	41	3204	18	0.01	84
`CloudMedium` ($\lvert\mathcal{N}\rvert = 400$)	CPLEX-BSOCP	3600	100.0	0.0	0	0	0	-	-	-	-
	B&P-BSOCP	3600	39.0	0.1	2	0	4	861	100	0.39	98
	B&P-PWL	3600	17.2	0.4	1	0	10	3372	100	0.01	91
	B&P-Hybrid	3600	11.8	0.6	12	0	15	6879	9	0.04	73
	B&P-Hybrid*	3600	11.2	3.0	12	0	49	6797	9	0.03	71
`CloudBig` ($\lvert\mathcal{N}\rvert = 1000$)	CPLEX-BSOCP	3600	100.0	0.0	0	0	0	-	-	-	-
	B&P-BSOCP	3600	59.6	0.0	2	0	0	741	100	0.04	89
	B&P-PWL	3600	43.1	0.2	1	0	6	2105	100	0.01	63
	B&P-Hybrid	3600	34.2	0.4	1	0	11	4257	4	0.01	8
	B&P-Hybrid*	3600	35.3	0.6	1	0	25	4088	4	0.01	7

Table 1 Aggregated statistics of the main computational results

In the medium and big benchmarks, none of the instances are solved. Note that 400 and 1000 items are very large for the classical linear BPs.

The average number of nodes and the average dual gap of the CPLEX-BSOCP are 0 and 100 %, and the CPLEX-BSOCP even cannot finish the root node procedure. Because by default `CPLEX` adds SOC cuts for nonlinear constraints (2b) and reoptimizes the LP relaxation iteratively, the resulting size of LP relaxation is too large to solve efficiently. The B&P-Hybrid (or the B&P-Hybrid*) is still able to prove a dual gap for the approximation solution or find better primal solutions.

The set-covering formulation is more scalable than the compact formulation.

LP -B&C vs. PWL-B&C As for the exact pricing algorithms, we compare the LP -B&C algorithm implemented by `CPLEX` and our PWL-B&C algorithm. The comparisons

are mainly related to the ability of the pricing algorithms to prove the dual bounds in limited time. We focus on the problem statistics of the B&P-BSOCP and the B&P-PWL.

For the small, medium, and large benchmarks, the average dual gaps of the B&P-BSOCP are 4.17, 2.26, and 1.38 times as large as those of the B&P-PWL, respectively. For the small benchmark, the average total time and average number of solved instances of the B&P-BSOCP are 3.35 and 0.33 times as large as those of the B&P-PWL.

Most time is spent on pricing problems. For the small, medium, and large benchmarks, the average number of generated columns of the B&P-PWL is 1.36, 3.91, and 2.83 times that of the B&P-BSOCP, respectively; the average double gap of pricing problems of the B&P-BSOCP is 445.36, 48.875, and 5.25 times that of the B&P-PWL, respectively.

`CPLEX` can solve "simple" pricing problems, but when "difficult" pricing problems occur after generating many columns, it cannot prove tight dual bounds. We find that for the B&P-BSOCP, the average dual gap in pricing decreases when the instance size increases. This is because for the large instance, only a small number of columns are generated before the time limit and the "hard" pricing problems do not arise.

In contrast, the B&P-PWL solves almost all pricing problems up to optimality (about 0.01%) in a finite time.

Exact pricing vs. hybrid pricing We compare the exact pricing strategy (B&P-PWL) with the hybrid pricing strategy (B&P-Hybrid).

For the small benchmark, the average total time of the B&P-PWL is 1.92 times that of the B&P-Hybrid, and the B&P-Hybrid solves 4 instances more. For the small, medium, and large benchmarks, the average dual gaps of the B&P-PWL are 1.2, 1.46, and 1.26 times as large as those of the B&P-Hybrid, respectively. This is because most of the slow exact pricing procedures are replaced by the fast heuristic pricing procedures in the B&P hybrid. Therefore, more columns can be generated.

For the small, medium, and large benchmarks, the average number of columns generated by the B&P hybrid is 1.86, 2.04, and 2.02 times that of the B&P PWL, respectively; the average percentage of exact columns generated by the B&P hybrid is 18.92%, 9.19%, and 4.747%, respectively. Note that the price-dual gap of the B&P hybrid is slightly larger than that of the B&P-PWL because more "hard" columns are generated.

The results show that the hybrid pricing strategy is a scalable approach for large instances. The B&P hybrid also has better performance in terms of the closed primary bound and the number of improved instances. This is because the B&P hybrid generates more columns and the `SCIP` heuristics have more chances to improve the primary solutions.

Effects of column selection heuristics We analyze the effects of the column selection heuristic, for which we compare the B&P hybrid and the B&P hybrid*.

In terms of dual task and total time for the small benchmark, the B&P hybrid performs slightly better than the B&P hybrid*. We focus on the quality of the primary solution in the statistics of the main problem. For the small benchmark, the B&P hybrid and the B&P hybrid* improve 36 and 41 instances, respectively. The CPLEX-BSOCP improves 49 instances, and the average closed primary bound of the CPLEX-BSOCP is 1.57 times that of the B&P hybrid*. The CPLEX-BSOCP is the best solver in the primary tasks for the small benchmark.

For the medium and large benchmarks, the B&P hybrid* improves 49 and 21 instances, and the B&P hybrid improves 15 and 11 instances. For the medium and large benchmarks, the average closed primary barriers of the B&P hybrid* are 4.75 and 1.61 times greater than those of the B&P hybrid, respectively.

The column selection heuristic is useful for finding better primary solutions.

Summary of the analysis The B&P hybrid is the best solver in the dual tasks. For the exact pricing algorithms, PWL-B&C outperforms `CPLEX` in all instances, and consequently PWL-B&C yields a substantial improvement on the dual tasks for the B&P algorithms. This result suggests that PWL-B&C can also be an independent solver for submodular Knapsack problems. The hybrid pricing strategy speeds up column generation. The column selection heuristic can find better solutions than the approximate solutions.

As for benchmarks, `CloudSmall` is a suitable testbed for comparing solvers, `CloudMedium` is suitable for testing the pricing algorithms, and `CloudBig` is still too big to handle.

5.4. Non-equidistant Breakpoints

According to Theorem 2 in Section 4.4, the optimal breakpoints under the error ℓ_∞ form an equidistant partition of $[\underline{w}, \overline{w}]$. In this section, we investigate whether adaptive non-equidistant breakpoints can improve the B&P hybrid*. There are many possibilities for non-equidistant breakpoints, and we propose a simple approach below.

From the previous experiments, we first made the following observations. When the PWL-B&C algorithm solves a pricing problem by generating many cuts, the problem appears to be a "hard" pricing problem. As a result, exact pricing is slow and numerically unstable, so that `CPLEX` sometimes issues the warning "Advanced basis is not built". This phenomenon usually occurs at the end of column generation.

Therefore, the number of cuts generated should be reduced. This can be achieved by adjusting the breakpoints, as the following example shows.

When a pricing problem is solved by the PWL-B&C algorithm, let $\mathcal{X}_0$ be the set of all $\hat{x}$ in line 23 of Algorithm 1 that generate cuts. We note that $\mathcal{X}_0$ is a subset of the generating set $\mathcal{X}$ in (11), and we denote $\mathcal{X}_0$ as the *sub generating set*. We change the breakpoints of the PWL relaxation to $\mathcal{B}_0 := \{w \in \mathbb{R} : w = \sum_{i \in \mathcal{N}'} a_i \hat{x}_i, \hat{x} \in \mathcal{X}_0\}$. Since the PWL relaxation is now exactly in $\mathcal{B}_0$ (i.e., $\bar{q}_{\mathcal{B}_0}(w) = q(w)$ for $w\mathcal{B}_0$), when the PWL-B&C algorithm is re-executed, the submodular capacity constraint (9b) is satisfied for $x\mathcal{X}_0$. Therefore, the cuts generated by $\mathcal{X}_0$ are not added to line 23 of Algorithm 1.

Then we note that the hybrid pricing strategy uses many heuristic pricing iterations between two exact pricing iterations. For such two pricing iterations, let $\mathcal{X}_1$ and $\mathcal{X}_2$ be the respective subsets, and $\mathcal{B}_1 := \{w \in \mathbb{R} : w = \sum_{i \in \mathcal{N}'} a_i \hat{x}_i, \hat{x} \in \mathcal{X}_1\}$ and $\mathcal{B}_2 := \{w \in \mathbb{R} : w = \sum_{i \in \mathcal{N}'} a_i \hat{x}_2, \hat{x} \in \mathcal{X}_2\}$. We note that the points of $\mathcal{B}_1$ and $\mathcal{B}_2$ can be very far apart.

From the second observation, it is difficult to reuse the information from the previous exact iterations and predict the correct breakpoints of the current iteration.

Finally, we also note that the partial generation set will converge to a midpoint during the one run of the PWL-B&C algorithm.

We use the following non-equidistant breakpoint strategy. We compute the midpoint of $\mathcal{B}$ from a "warm-up" PWL-B&C algorithm with the equidistant breakpoints.

In the "warm-up" stage, we record the sub generating set $\mathcal{X}'$ as an ordered list, and assign each element of $\mathcal{X}'$ an order according to the time that this element is added to $\mathcal{X}'$. Let $w_{-1} = \sum_{i \in \mathcal{N}'} a_i \hat{x}_i$, where $\hat{x}$ is the last element in $\mathcal{X}'$. Once Line 23 of Algorithm 1 is executed, let $w = \sum_{i \in \mathcal{N}'} a_i \hat{x}_i$. If $\frac{w_c - w_{-1}}{\overline{w} - \underline{w}} \leq 0.1$ (the convergence criteria is satisfied), then the "warm-up" PWL-B&C algorithm terminates and outputs $w_c := w$. Otherwise, we add $\hat{x}$ to $\mathcal{X}'$ and continue the algorithm.

In the "restart" stage, we rerun the PWL-B&C algorithm with the following non-equidistant breakpoints. The number of new breakpoints is the same as the number of old

breakpoints. Let $i_c := \lceil \frac{w-\underline{w}}{\overline{w}-\underline{w}} h \rceil$, $S_l := \sum_{1 \le j \le i_c - 1} j$, and $S_u := \sum_{1 \le j \le h - i_c} j$, we generate the non-equidistant breakpoints centered at w_c as follows:

$$w_i := \begin{cases} w_c - \frac{\sum_{1 \le j \le i_c - i} j}{S_l}(w_c - \underline{w}), & 1 \le i < i_c, \\ w_c, & i = i_c, \\ w_c + \frac{\sum_{1 \le j \le i - i_c} j}{S_u}(\overline{w} - w_c), & h \ge i > i_c. \end{cases}$$

The set $\mathcal{B} = \{w_i\}$ is centered at w_c with high-density.

Next, we implement a solver, namely B&P-Hybrid**, which modifies B&P-Hybrid* by using the two-stage PWL-B&C algorithm for exact pricing. The two solvers are tested on the `CloudMedium` benchmark, since none of the instances in this benchmark can be solved by any solver and the time to solve RMLPs in Section 3.2 is not very long.

For a detailed comparison, the results in Table 2 show the average SGM statistics of instances with the same risk level and generation method. As for the notation of the instances, "G" denotes the instances of the Gaussian distribution, "H" denotes the instances of the Hoeffding inequality, and "D" denotes the instances of the distributionally robust case.

We first focus on the price statistics. The average number of columns of the B&P hybrid** is 1.08 times that of the B&P hybrid*; the average double pricing gap and the average exact pricing time of the B&P hybrid* are 3 times and 1.04 times that of the B&P hybrid**, respectively. Although the two-stage PWL-B&C algorithm takes more time, the time to solve difficult price instances is reduced due to the refined non-equidistant breakpoints. The two-stage PWL-B&C algorithms solve all pricing problems up to optimality (0.01%), except for the distributionally robust case and $\alpha = 0.99$.

Next, we focus on the master statistics. The average number of improved instances and the closed primary bound of the B&P hybrid** are 1.1 and 1.2 times larger than those of the B&P hybrid*, respectively. This is because more columns are generated and therefore the primary heuristic has more chances to find better solutions. The average dual gap of the B&P hybrid* is 1.02 times larger than that of the B&P hybrid**.

We conclude that the non-equidistant breakpoints lead to a more efficient solution of the pricing problems and a marginal improvement of the master problems.

Case	α	B&P-Hybrid*									B&P-Hybrid**								
		Master					Pricing				Master					Pricing			
		δ_d%	δ_p%	#N	#S	#I	#C	E%	τ%	t_p%	δ_d%	δ_p%	#N	#S	#I	#C	E%	τ%	t_p%
G	0.6	8.6	0.0	32	0	0	8054	5	0.06	55	8.5	0.0	34	0	0	7783	5	0.01	54
	0.7	10.9	0.0	100	0	0	12105	4	0.01	50	10.9	0.0	106	0	0	12577	3	0.01	47
	0.8	9.7	4.4	42	0	3	9597	5	0.02	55	9.3	8.4	64	0	4	10685	5	0.01	52
	0.9	11.2	11.0	24	0	5	7190	9	0.01	73	11.2	11.0	26	0	5	7553	9	0.01	72
	0.95	13.7	15.0	8	0	6	6150	11	0.01	81	12.5	18.9	8	0	6	6084	14	0.01	80
	0.99	13.3	2.9	2	0	3	6786	9	0.01	74	12.7	5.0	4	0	4	7454	11	0.01	70
H	0.6	10.5	0.0	21	0	0	6258	8	0.1	70	10.3	0.0	42	0	0	7592	7	0.01	64
	0.7	10.6	0.0	32	0	0	7005	7	0.07	64	10.4	0.0	37	0	0	8120	7	0.01	57
	0.8	11.6	0.0	10	0	0	5715	10	0.09	75	11.4	0.0	33	0	0	7446	8	0.01	65
	0.9	10.8	0.0	9	0	0	6275	9	0.05	73	10.6	0.0	17	0	0	7071	10	0.01	69
	0.95	10.7	0.0	23	0	0	6570	9	0.02	71	10.6	0.0	30	0	0	7360	9	0.01	68
	0.99	12.7	0.0	27	0	0	7067	9	0.02	72	12.3	0.7	42	0	1	8074	8	0.01	67
D	0.6	10.3	21.8	13	0	6	6962	9	0.01	70	10.3	21.8	20	0	6	8015	8	0.01	67
	0.7	13.6	15.1	19	0	6	6991	10	0.01	78	13.6	15.1	25	0	6	7261	10	0.01	77
	0.8	14.8	7.5	2	0	5	5848	11	0.01	80	14.0	9.1	2	0	5	6292	14	0.01	77
	0.9	14.8	2.3	2	0	3	6844	8	0.02	78	14.2	6.7	2	0	5	6584	8	0.01	78
	0.95	16.5	17.6	2	0	6	5817	14	0.01	87	17.1	14.8	2	0	6	5698	13	0.01	87
	0.99	3.9	77.4	2	0	6	4126	18	0.09	95	4.2	76.2	3	0	6	4060	17	0.07	95
All		11.2	3.0	12	0	49	6797	9	0.03	71	11.0	3.6	17	0	54	7343	9	0.01	68

Table 2 **Master and pricing problem statistics of the B&P-Hybrid* and B&P-Hybrid** for `CloudMedium`**

6. Conclusion

In this work, we study exact branch-and-price algorithms for SMBPs. We develop the PWL-B&C algorithm for pricing submodular Knapsack problems. We find that the branch-and-price algorithm is a promising method for solving SMBPs, as in linear BPs. Our branch-and-price algorithms can solve more instances than `CPLEX`.

The proposed PWL-B&C algorithm is more efficient than `CPLEX` for sub-modular Knapsack problems. The PWL-B&C algorithm can also be extended to solve the multiple sub-modular Knapsack problems. For general MINLP problems, if a nonlinear constraint can be reformulated into a linear part and a univariate nonlinear part, then the univariate nonlinear part can be convexified by the PWL relaxation. Our hybrid pricing strategy is applicable to the column generation procedure, where the main problems are in set-cover formulations, as long as there are fast pricing heuristics. This pricing strategy is useful for large instances. For example, consider the high capacity vehicle routing problem, for which an exact pricing algorithm is difficult.

We compare the equidistant and non-equidistant breakpoints for the PWL-B&C algorithm, and the non-equidistant breakpoints can speed up the PWL-B&C algorithm given a suitable

partition. The future study can investigate a more accurate partition than the one created by the "warm-up" phase of the PWL-B&C algorithm.

ACKNOWLEDGMENT The authors would like to thank Leo Liberti and and Sandra Ulrich Ngueveu for discussion with the authors.

Appendix A: Proof of Proposition 1

Proof. Let

$$F_j := \{(v_{1j}, \cdots, v_{nj}, y_j) \in \{0,1\}^n \times \{0,1\} : \sum_{i\in\mathcal{N}} a_i v_{ij} + \sigma\sqrt{\sum_{i\in\mathcal{N}} b_i v_{ij}} \le c y_j\}$$

be the feasible set of the $j-$th constraint in the BSOCP formulation. Therefore, the feasible set of the BSCOP formulation is $F = \prod_{j\in\mathcal{M}} F_j$.

Let $\overline{F}_j$ be the continuous relaxation of F_j, and

$$\overline{F}_j = \{(v_{1j}, \cdots, v_{nj}, y_j) \in [0,1]^n \times [0,1] : \sum_{i\in\mathcal{N}} a_i v_{ij} + \sigma\sqrt{\sum_{i\in\mathcal{N}} b_i v_{ij}} \le c y_j\}.$$

Therefore, the feasible set of the continuous relaxation of the BSCOP formulation is $\overline{F} = \prod_{j\in\mathcal{M}} \overline{F}_j$.

On the other hand, the points of F_j are zero vector and $(p,1)$ $(p \in \mathcal{P})$. Therefore, its convex hull is

$$\text{conv}(F_j) = \{(v_{1j}, \cdots, v_{nj}, y_j) \in [0,1]^n \times [0,1] : \exists \lambda_p \in [0,1]^{\mathcal{P}}, \sum_{p\in\mathcal{P}} \lambda_p = y_j, v = \sum_{p\in\mathcal{P}} d_p \lambda_p\}.$$

We note that $\overline{F}_j$ is also a convex relaxation of F_j, hence $F_j \subset \text{conv}(F_j) \subset \overline{F}_j$.

The optimum of the continuous relaxation of the BSOCP formulation is $\min_{(v,y)\in\overline{F}, v \text{ satisfies (2c)}} \sum_{j\in\mathcal{M}} y_j$.

An optimal solution of the LP relaxation of the set cover formulation satisfies $\sum_{p\in\mathcal{P}} d_{ip}\lambda_p = 1$ $(i \in \mathcal{N})$, and the optimal value is exactly the same as $\min_{(v,y)\in\prod_{j\in\mathcal{M}} \text{conv}(F_j), v \text{ satisfies (2c)}} \sum_{j\in\mathcal{M}} y_j$. Since $\prod_{j\in\mathcal{M}} \text{conv}(F_j) \subset \overline{F}$, the result follows. □

Appendix B: Proof of Theorem 2

Proof. Since $\bar{q}_{\mathcal{B}}$ and q have the same value at $w \in \{w_1, \ldots, w_h\}$, it follows that the ℓ_∞-norm is the maximum value of ℓ_∞-norms over individual sub intervals:

$$\ell_\infty(\bar{q}_{\mathcal{B}}, q) = \max_{w\in[\underline{w},\overline{w}]} |\bar{q}_{\mathcal{B}}(w) - q(w)| = \max_{2\le k\le h} \max_{w\in[w_{k-1},w_k]} |\bar{q}_{\mathcal{B}}(w) - q(w)|.$$

Let $w \in [w_{k-1}, w_k]$, then

$$\begin{aligned}
&|\bar{q}_{\mathcal{B}}(w) - q(w)| \\
=&\frac{q(w_k) - q(w_{k-1})}{w_k - w_{k-1}}(w - w_{k-1}) + q(w_{k-1}) - (c-w)^2 \\
=&(w - w_{k-1})(w_k - w).
\end{aligned}$$

We have

$$\begin{aligned}
&\max_{w\in[w_{k-1},w_k]} |\bar{q}_{\mathcal{B}}(w) - q(w)| \\
=&\max_{w\in[w_{k-1},w_k]} (w - w_{k-1})(w_k - w) \\
=&\frac{(w_k - w_{k-1})^2}{4}.
\end{aligned}$$

The maximum value is at $w = \frac{w_{k-1}+w_k}{2}$.

It follows that (16) is equivalent to:

$$\min_{\underline{w}=w_1\le\ldots\le w_h=\overline{w}} \max_{2\le k\le h} \frac{(w_k - w_{k-1})^2}{4}.$$

Therefore, the optimal solution is an equidistant partition of $[\underline{w}, \overline{w}]$, and the results follow. □

Appendix C: Benchmark Generation

Next, we briefly review the method to generate an instance. We call the distribution of μ_i the *target distribution* for item i. We assume that every μ_i follows the same target distribution. This target distribution is unknown in Cohen et al. (2019) except for its quantiles in Table 3.

Given α and $\mathcal{N}$, we generate an SMBP instance as follows:

1. sample μ_i ($i \in \mathcal{N}$) according to Table 3;
2. sample a and b from μ and σ, using one of the following cases:

- Gaussian case;
- Hoeffding's inequality case;
- distributionally robust approximation case.

Table 3 Example distribution of item size

Item sizes	1	2	4	8	16	32	72
% Items	36.3	13.8	21.3	23.1	3.5	1.9	0.1

We first illustrate the approach of sampling μ. We approximate the target distribution by a normalized histogram such that its quantile distribution is the same as in Table 3. A histogram consists of intervals divided from the entire range $[0, 72]$, and each interval has endpoints of two consecutive quantiles of Table 3. The histogram gives a discrete non-parametric estimation of the target distribution. To obtain a nominal item size μ_i ($i \in \mathcal{N}$) sampled as from a continuous distribution, we apply a two-stage sampling. It has two steps:

1. sample an interval $[d_1, d_2]$ from the histogram;
2. sample a nominal item size μ_i from $[d_1, d_2]$ uniformly.

Second, we construct a truncated Gaussian, which is defined by its lower and upper bounds $\underline{A}$ and $\overline{A}$, its mean μ', and its standard deviation σ'. To obtain these parameters, for each $i \in \mathcal{N}$, we:

1. sample $\underline{A}_i \in [0.3, 0.6]$ and $\overline{A}_i \in [0.7, 1.0]$ uniformly;
2. sample scale parameter $s_i \in [0.1, 0.5]$;
3. compute the mean μ'_i and the standard variation σ'_i of the truncated Gaussian with lower bound $\underline{A}_i$, upper bound $\overline{A}_i$ and scale parameter s_i.

With the above parameters, we generate the data a, b, σ of the SMBP (4). There are three cases, which correspond to different assumptions on the uncertainty or probability distribution.

For the Gaussian case:

1. let $\sigma = \Phi^{-1}(\alpha)$, where Φ is the cumulative distribution function of the Gaussian distribution;

2. for $i \in \mathcal{N}$:
   (a) $a_i = \mu_i' \mu_i$;
   (b) $b_i = (\sigma_i' \mu_i)^2$.

For the Hoeffding's inequality case:

1. let $\sigma = \sqrt{-0.5 \ln(1-\alpha)}$;
2. for $i \in \mathcal{N}$:
   (a) $a_i = \mu_i' \mu_i$;
   (b) $b_i = ((\overline{A}_i - \underline{A}_i)\mu_i)^2$.

For the distributionally robust approximation case:

1. let $\sigma = \sqrt{\alpha/(1-\alpha)}$;
2. for $i \in \mathcal{N}$:
   (a) $a_i = \mu_i' \mu_i$;
   (b) $b_i = (\sigma_i' \mu_i)^2$.

For all the above cases, if there exists $i \in \mathcal{N}$ such that a_i, b_i are too large to fit a bin (usually for large α, σ), then we rescale a_i, b_i to fit the bin.

Appendix D: Detailed Results

Master problem statistics are summarized in Table 4, Table 5, and Table 6. Pricing problem statistics are summarized in Table 7, Table 8, and Table 9.

As for the case notation, "G" denotes the instances of the Gaussian distribution case, "H" denotes the instances of the Hoeffding inequality case, and "D" denotes the instances of the distributionally robust case.

For each benchmark, we report the SGM statistics of performance metrics, the number of solved instances (#S), and the number of instances with improved primal bounds (#I).

We also divide the instances in each benchmark into small subsets and report SGM statistics of these subsets. In these subsets, instances have the same risk level and case.

Case	α	CPLEX-BSOCP						B&P-BSOCP						B&P-PWL						B&P-Hybrid						B&P-Hybrid*					
		t	δ_d%	δ_p%	#N	#S	#I	t	δ_d%	δ_p%	#N	#S	#I	t	δ_d%	δ_p%	#N	#S	#I	t	δ_d%	δ_p%	#N	#S	#I	t	δ_d%	δ_p%	#N	#S	#I
G	0.6	38	1.6	0.0	70	4	0	211	1.7	0.0	7	4	0	43	0.0	1.2	2	6	1	47	0.6	0.0	7	5	0	42	0.6	0.0	6	5	0
	0.7	1199	10.6	0.0	42897	1	0	1049	6.0	0.0	10	2	0	145	1.6	0.0	9	4	0	80	1.5	0.0	20	4	0	84	1.5	0.0	19	4	0
	0.8	3600	21.6	0.0	215412	0	0	3600	19.3	0.0	21	0	0	3161	10.0	0.0	287	1	0	2792	9.7	0.0	1177	1	0	2917	9.8	0.0	918	1	0
	0.9	3600	24.6	0.0	236150	0	0	3600	23.3	0.0	25	0	0	624	1.7	3.7	79	4	2	221	1.5	3.7	112	4	2	271	1.5	3.7	126	4	2
	0.95	3600	30.3	0.0	125945	0	0	3600	23.3	0.0	16	0	0	1160	3.6	3.7	140	3	2	1190	5.1	6.5	743	2	3	836	1.4	34.1	465	4	5
	0.99	3600	35.6	0.0	85454	0	0	3600	27.4	0.0	25	0	0	1952	5.5	6.3	266	2	3	1624	10.1	0.8	1110	1	1	828	5.4	6.3	442	2	3
H	0.6	535	3.5	0.0	6828	3	0	2300	6.4	1.2	66	2	1	777	1.7	1.2	35	4	1	166	0.6	3.7	41	5	2	253	1.7	1.2	43	4	1
	0.7	532	6.0	0.0	3565	2	0	572	3.1	0.0	6	3	0	71	0.0	0.0	1	6	0	18	0.0	0.0	1	6	0	19	0.0	0.0	1	6	0
	0.8	178	3.3	0.0	2159	3	0	1728	7.6	0.0	37	2	0	572	0.7	9.0	83	5	3	528	3.8	1.2	277	3	1	587	3.8	1.2	247	3	1
	0.9	352	3.4	0.0	1807	3	0	3600	20.4	0.0	76	0	0	1506	6.1	0.0	160	2	0	413	1.6	3.7	167	4	2	455	1.6	3.7	162	4	2
	0.95	734	6.5	0.0	23189	2	0	990	6.5	0.0	17	2	0	329	1.5	0.0	18	4	0	119	0.6	1.2	17	5	1	203	1.4	1.2	27	4	1
	0.99	3600	20.9	0.0	345213	0	0	3600	19.7	0.0	16	0	0	253	1.8	1.2	31	4	1	215	1.5	1.2	91	4	1	222	2.9	0.9	81	3	1
D	0.6	3600	24.0	0.0	187687	0	0	3600	20.2	0.0	18	0	0	419	1.7	1.2	34	4	1	145	0.6	7.8	55	5	3	135	1.5	3.1	46	4	2
	0.7	3600	29.0	0.0	153161	0	0	3600	23.1	2.3	21	0	2	1234	1.6	9.0	141	4	3	458	1.6	9.0	276	4	3	476	1.6	9.0	246	4	3
	0.8	3600	34.7	0.0	95858	0	0	3600	22.3	0.0	20	0	0	3600	14.4	0.9	463	0	1	2153	2.8	9.0	1542	3	3	2476	4.6	7.8	1305	2	3
	0.9	3600	44.6	0.0	41259	0	0	3600	25.8	2.4	27	0	2	3146	8.8	6.6	384	1	3	2389	9.0	3.0	1260	1	2	2459	7.4	25.0	1111	1	5
	0.95	3600	62.1	0.0	29645	0	0	3600	15.4	46.9	20	0	6	3321	6.3	31.8	414	1	5	2332	3.5	66.3	664	2	6	2028	3.8	59.1	741	2	6
	0.99	3600	77.5	0.0	10352	0	0	1218	0.6	96.4	33	5	6	117	0.0	100.0	90	6	6	30	0.0	100.0	32	6	6	27	0.0	100.0	32	6	6
All		1452	15.8	0.0	26601	18	0	2129	11.4	0.9	21	20	17	633	2.4	2.7	66	61	32	330	2.0	3.4	127	65	36	335	2.1	4.1	114	63	41

Table 4 Master problem statistics of `CloudSmall` with 108 instances ($|\mathcal{N}| = 100$)

Case	α	CPLEX-BSOCP						B&P-BSOCP						B&P-PWL						B&P-Hybrid						B&P-Hybrid*					
		t	δ_d%	δ_p%	#N	#S	#I	t	δ_d%	δ_p%	#N	#S	#I	t	δ_d%	δ_p%	#N	#S	#I	t	δ_d%	δ_p%	#N	#S	#I	t	δ_d%	δ_p%	#N	#S	#I
G	0.6	3600	100.0	0.0	0	0	0	3600	20.6	0.0	2	0	0	3600	14.1	0.0	2	0	0	3600	8.5	0.0	32	0	0	3600	8.6	0.0	32	0	0
	0.7	3600	100.0	0.0	0	0	0	3600	30.1	0.0	2	0	0	3600	16.3	0.0	2	0	0	3600	10.9	0.0	122	0	0	3600	10.9	0.0	100	0	0
	0.8	3600	100.0	0.0	0	0	0	3600	36.4	0.0	2	0	0	3600	15.9	0.0	2	0	0	3600	11.3	0.0	46	0	0	3600	9.7	4.4	42	0	3
	0.9	3600	100.0	0.0	0	0	0	3600	42.2	0.0	2	0	0	3600	19.1	0.0	2	0	0	3600	13.0	0.0	18	0	0	3600	11.2	11.0	24	0	5
	0.95	3600	100.0	0.0	0	0	0	3600	48.4	0.0	2	0	0	3600	18.8	0.0	2	0	0	3600	14.8	0.8	9	0	1	3600	13.7	14.8	8	0	6
	0.99	3600	100.0	0.0	0	0	0	3600	47.6	0.0	2	0	0	3600	22.3	0.0	2	0	0	3600	14.1	0.0	2	0	0	3600	13.3	2.9	2	0	3
H	0.6	3600	100.0	0.0	0	0	0	3600	27.7	0.0	2	0	0	3600	16.4	0.0	2	0	0	3600	10.5	0.0	22	0	0	3600	10.5	0.0	21	0	0
	0.7	3600	100.0	0.0	0	0	0	3600	29.0	0.0	2	0	0	3600	18.0	0.0	2	0	0	3600	10.5	0.0	30	0	0	3600	10.6	0.0	32	0	0
	0.8	3600	100.0	0.0	0	0	0	3600	31.5	0.0	2	0	0	3600	17.2	0.0	2	0	0	3600	11.6	0.0	9	0	0	3600	11.6	0.0	10	0	0
	0.9	3600	100.0	0.0	0	0	0	3600	33.3	0.0	2	0	0	3600	17.0	0.0	2	0	0	3600	10.9	0.0	7	0	0	3600	10.8	0.0	9	0	0
	0.95	3600	100.0	0.0	0	0	0	3600	35.1	0.0	2	0	0	3600	17.6	0.0	2	0	0	3600	10.7	0.0	24	0	0	3600	10.7	0.0	23	0	0
	0.99	3600	100.0	0.0	0	0	0	3600	39.3	0.0	2	0	0	3600	19.4	0.0	2	0	0	3600	12.7	0.0	27	0	0	3600	12.7	0.0	27	0	0
D	0.6	3600	100.0	0.0	0	0	0	3600	42.9	0.0	2	0	0	3600	20.2	0.9	2	0	1	3600	12.9	0.0	10	0	0	3600	10.3	21.8	13	0	6
	0.7	3600	100.0	0.0	0	0	0	3600	48.7	0.0	2	0	0	3600	21.3	0.0	2	0	0	3600	15.3	0.6	12	0	1	3600	13.6	15.0	19	0	6
	0.8	3600	100.0	0.0	0	0	0	3600	48.4	0.0	2	0	0	3600	22.7	0.0	2	0	0	3600	15.1	0.9	2	0	1	3600	14.8	7.4	2	0	5
	0.9	3600	100.0	0.0	0	0	0	3600	51.2	0.0	2	0	0	3600	21.0	0.0	2	0	0	3600	15.4	0.0	2	0	0	3600	14.8	2.3	2	0	3
	0.95	3600	100.0	0.0	0	0	0	3600	55.9	0.0	2	0	0	3600	22.3	2.6	2	0	3	3600	17.4	13.7	2	0	6	3600	16.5	17.6	2	0	6
	0.99	3600	100.0	0.0	0	0	0	3600	57.4	3.3	2	0	4	3600	4.0	78.4	2	0	6	3600	4.2	76.4	3	0	6	3600	3.9	77.4	2	0	6
All		3600	100.0	0.0	0	0	0	3600	39.0	0.1	2	0	4	3600	17.2	0.4	1	0	10	3600	11.8	0.6	12	0	15	3600	11.2	3.0	12	0	49

Table 5 Master problem statistics of `CloudMedium` with 108 instances ($|\mathcal{N}| = 400$)

Case	α	CPLEX-BSOCP						B&P-BSOCP						B&P-PWL						B&P-Hybrid						B&P-Hybrid*					
		t	δ_d%	δ_p%	#N	#S	#I	t	δ_d%	δ_p%	#N	#S	#I	t	δ_d%	δ_p%	#N	#S	#I	t	δ_d%	δ_p%	#N	#S	#I	t	δ_d%	δ_p%	#N	#S	#I
G	0.6	3600	100.0	0.0	0	0	0	3600	38.9	0.0	2	0	0	3600	38.7	0.0	2	0	0	3600	32.6	0.0	2	0	0	3600	32.6	0.0	2	0	0
	0.7	3600	100.0	0.0	0	0	0	3600	41.7	0.0	2	0	0	3600	38.7	0.0	2	0	0	3600	34.9	0.0	2	0	0	3600	35.2	0.0	2	0	0
	0.8	3600	100.0	0.0	0	0	0	3600	51.2	0.0	2	0	0	3600	39.1	0.0	2	0	0	3600	35.0	0.0	2	0	0	3600	35.2	0.0	2	0	0
	0.9	3600	100.0	0.0	0	0	0	3600	64.1	0.0	2	0	0	3600	47.3	0.0	2	0	0	3600	41.3	0.2	1	0	1	3600	41.9	0.9	1	0	3
	0.95	3600	100.0	0.0	0	0	0	3600	71.3	0.0	2	0	0	3600	47.7	0.0	1	0	0	3600	41.5	0.2	1	0	1	3600	41.6	1.2	1	0	4
	0.99	3600	100.0	0.0	0	0	0	3600	79.8	0.0	2	0	0	3600	45.0	0.0	1	0	0	3600	38.4	0.2	1	0	1	3600	38.9	0.4	1	0	2
H	0.6	3600	100.0	0.0	0	0	0	3600	44.5	0.0	2	0	0	3600	41.6	0.0	2	0	0	3600	36.4	0.0	1	0	0	3600	36.8	0.0	2	0	0
	0.7	3600	100.0	0.0	0	0	0	3600	45.7	0.0	2	0	0	3600	41.0	0.0	2	0	0	3600	35.5	0.0	2	0	0	3600	35.7	0.0	2	0	0
	0.8	3600	100.0	0.0	0	0	0	3600	47.5	0.0	2	0	0	3600	43.1	0.0	2	0	0	3600	37.5	0.0	2	0	0	3600	36.1	0.0	1	0	0
	0.9	3600	100.0	0.0	0	0	0	3600	50.3	0.0	2	0	0	3600	41.9	0.0	2	0	0	3600	36.6	0.0	1	0	0	3600	36.7	0.0	2	0	0
	0.95	3600	100.0	0.0	0	0	0	3600	52.1	0.0	2	0	0	3600	43.3	0.0	2	0	0	3600	37.3	0.0	1	0	0	3600	37.0	0.0	1	0	0
	0.99	3600	100.0	0.0	0	0	0	3600	58.5	0.0	2	0	0	3600	46.1	0.0	2	0	0	3600	41.3	0.0	1	0	0	3600	41.3	0.0	1	0	0
D	0.6	3600	100.0	0.0	0	0	0	3600	62.2	0.0	2	0	0	3600	48.0	0.0	1	0	0	3600	41.4	0.0	1	0	0	3600	41.7	0.0	1	0	0
	0.7	3600	100.0	0.0	0	0	0	3600	69.6	0.0	2	0	0	3600	46.8	0.0	1	0	0	3600	42.4	0.0	1	0	0	3600	42.2	0.5	1	0	2
	0.8	3600	100.0	0.0	0	0	0	3600	77.1	0.0	2	0	0	3600	45.3	0.0	1	0	0	3600	39.6	0.0	1	0	0	3600	39.4	0.5	1	0	2
	0.9	3600	100.0	0.0	0	0	0	3600	82.2	0.0	2	0	0	3600	44.1	0.0	2	0	0	3600	35.7	0.0	1	0	0	3600	36.7	0.0	1	0	0
	0.95	3600	100.0	0.0	0	0	0	3600	84.5	0.0	2	0	0	3600	49.0	0.0	2	0	0	3600	27.7	1.6	1	0	2	3600	29.8	7.0	2	0	6
	0.99	3600	100.0	0.0	0	0	0	3600	84.9	0.0	2	0	0	3600	32.9	19.9	2	0	6	3600	7.9	61.0	2	0	6	3600	13.1	49.6	2	0	6
All		3600	100.0	0.0	0	0	0	3600	59.6	0.0	2	0	0	3600	43.1	0.2	1	0	6	3600	34.2	0.4	1	0	11	3600	35.3	0.6	1	0	25

Table 6 Master problem statistics of `CloudBig` with 108 instances ($|\mathcal{N}| = 1000$)

Case	α	B&P-BSOCP				B&P-PWL				B&P-Hybrid				B&P-Hybrid*			
		#C	E%	τ%	t_p%	#C	E%	τ%	t_p%	#C	E%	τ%	t_p%	#C	E%	τ%	t_p%
G	0.6	519	100	0.1	99	226	100	0.01	99	440	33	0.02	97	389	33	0.02	94
	0.7	836	100	2.1	99	517	100	0.01	99	838	27	0.02	97	808	28	0.02	89
	0.8	1784	100	5.71	99	7573	100	0.01	99	25935	12	0.01	94	21026	12	0.01	70
	0.9	1711	100	11.67	99	1874	100	0.01	99	2904	16	0.01	96	3260	15	0.01	87
	0.95	1717	100	12.59	99	2887	100	0.01	99	14654	12	0.01	95	8706	13	0.01	81
	0.99	1699	100	16.48	99	5800	100	0.01	99	20729	12	0.01	94	10076	11	0.01	73
H	0.6	2449	100	0.29	99	1869	100	0.02	99	1511	20	0.03	96	2084	19	0.02	86
	0.7	523	100	4.14	99	204	100	0.02	99	179	35	0.01	98	194	34	0.01	98
	0.8	1594	100	0.29	99	2149	100	0.01	99	7162	13	0.01	94	6128	13	0.01	76
	0.9	2663	100	0.9	99	3782	100	0.01	99	4476	16	0.01	95	4117	18	0.01	86
	0.95	897	100	1.48	99	942	100	0.02	99	861	32	0.03	97	1293	33	0.02	91
	0.99	1722	100	8.05	99	940	100	0.01	99	2420	21	0.01	96	2274	20	0.01	81
D	0.6	1821	100	6.4	99	1306	100	0.01	99	2099	16	0.01	95	1798	16	0.01	86
	0.7	1728	100	12.51	99	3303	100	0.01	99	7434	10	0.01	95	6730	10	0.01	81
	0.8	1697	100	15.03	99	7952	100	0.01	99	23859	12	0.01	95	20313	13	0.01	75
	0.9	1686	100	17.62	99	6862	100	0.01	99	20794	15	0.01	96	18002	15	0.01	83
	0.95	1703	100	16.3	99	5850	100	0.01	99	7948	37	0.01	98	8224	32	0.01	95
	0.99	601	100	10.74	99	730	100	0.01	99	603	23	0.01	97	606	22	0.01	96
All		1373	100	3.56	99	1869	100	0.01	99	3485	18	0.01	96	3204	18	0.01	84

Table 7 Pricing problem statistics of `CloudSmall` with 108 instances ($|\mathcal{N}| = 100$)

Case	α	B&P-BSOCP				B&P-PWL				B&P-Hybrid				B&P-Hybrid*			
		#C	E%	τ%	t_p%	#C	E%	τ%	t_p%	#C	E%	τ%	t_p%	#C	E%	τ%	t_p%
G	0.6	2060	100	0.01	94	3117	100	0.01	88	9368	4	0.06	54	8054	5	0.06	55
	0.7	1413	100	0.02	97	3622	100	0.01	87	12327	4	0.02	54	12105	4	0.01	50
	0.8	943	100	0.11	98	3847	100	0.01	88	11031	5	0.02	61	9597	5	0.02	55
	0.9	814	100	0.36	99	3277	100	0.01	91	6595	9	0.01	74	7190	9	0.01	73
	0.95	658	100	0.77	99	3200	100	0.01	93	6212	11	0.01	81	6150	11	0.01	81
	0.99	589	100	2.04	99	3816	100	0.01	92	6622	10	0.01	76	6786	9	0.01	74
H	0.6	1503	100	0.02	97	2985	100	0.01	90	6343	8	0.11	70	6258	8	0.1	70
	0.7	1281	100	0.03	97	3114	100	0.01	90	7692	7	0.07	64	7005	7	0.07	64
	0.8	1246	100	0.03	98	3346	100	0.01	91	5947	9	0.09	75	5715	10	0.09	75
	0.9	1043	100	0.06	98	3409	100	0.01	90	6472	9	0.05	73	6275	9	0.05	73
	0.95	867	100	0.13	98	3145	100	0.01	91	6513	9	0.02	74	6570	9	0.02	71
	0.99	840	100	0.22	99	3199	100	0.01	91	7448	9	0.02	73	7067	9	0.02	72
D	0.6	794	100	0.31	99	3189	100	0.01	91	6691	9	0.01	67	6962	9	0.01	70
	0.7	681	100	0.57	99	3191	100	0.01	93	6069	11	0.01	80	6991	10	0.01	78
	0.8	609	100	1.55	99	3674	100	0.01	93	5792	11	0.01	81	5848	11	0.01	80
	0.9	552	100	2.35	99	3331	100	0.01	94	6703	9	0.02	78	6844	8	0.02	78
	0.95	519	100	2.7	99	3129	100	0.01	96	5757	15	0.01	90	5817	14	0.01	87
	0.99	448	100	16.35	99	4386	100	0.01	97	4162	18	0.12	97	4126	18	0.09	95
All		861	100	0.39	98	3372	100	0.01	91	6879	9	0.04	73	6797	9	0.03	71

Table 8 Pricing problem statistics of `CloudMedium` with 108 instances ($|\mathcal{N}| = 400$)

Case	α	B&P-BSOCP				B&P-PWL				B&P-Hybrid				B&P-Hybrid*			
		#C	E%	τ%	t_p%	#C	E%	τ%	t_p%	#C	E%	τ%	t_p%	#C	E%	τ%	t_p%
G	0.6	1786	100	0.01	55	1923	100	0.01	47	2748	6	0.01	5	2766	6	0.01	5
	0.7	1315	100	0.01	79	1903	100	0.01	53	2953	6	0.01	5	2932	6	0.01	5
	0.8	1038	100	0.01	90	1977	100	0.01	56	3182	5	0.01	6	3155	5	0.01	6
	0.9	760	100	0.02	96	2130	100	0.01	60	4433	3	0.01	5	4360	3	0.01	5
	0.95	629	100	0.03	98	2223	100	0.01	61	4724	3	0.01	6	4597	3	0.01	5
	0.99	409	100	0.08	99	2340	100	0.01	64	5005	3	0.01	6	4795	3	0.01	6
H	0.6	1422	100	0.01	76	1624	100	0.01	67	2968	5	0.01	10	2960	5	0.01	10
	0.7	1357	100	0.01	80	1689	100	0.01	66	3058	6	0.01	9	3017	6	0.01	9
	0.8	1242	100	0.01	84	1723	100	0.01	67	3214	5	0.01	8	3174	5	0.01	8
	0.9	1091	100	0.01	88	1694	100	0.01	66	3134	5	0.01	9	3070	5	0.01	9
	0.95	1017	100	0.01	90	1822	100	0.01	63	3316	5	0.01	8	3244	5	0.01	8
	0.99	877	100	0.01	94	1916	100	0.01	63	3780	4	0.01	7	3686	4	0.01	7
D	0.6	788	100	0.02	96	2114	100	0.01	60	4225	4	0.01	6	4145	4	0.01	5
	0.7	673	100	0.02	97	2158	100	0.01	60	4302	4	0.01	5	4130	4	0.01	5
	0.8	498	100	0.04	99	2210	100	0.01	63	4842	3	0.01	6	4634	3	0.01	6
	0.9	328	100	0.2	99	2461	100	0.01	69	6319	3	0.01	7	5800	3	0.01	7
	0.95	254	100	0.53	99	2208	100	0.01	82	9854	3	0.01	16	8816	3	0.01	12
	0.99	186	100	3.13	99	5373	100	0.01	84	12413	5	0.01	32	9778	4	0.01	17
All		741	100	0.04	89	2105	100	0.01	63	4257	4	0.01	8	4088	4	0.01	7

Table 9 Pricing problem statistics of `CloudBig` with 108 instances ($|\mathcal{N}| = 1000$)

References

Achterberg T, Berthold T, Koch T, Wolter K (2008) Constraint integer programming: A new approach to integrate cp and mip. Perron L, Trick MA, eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 6–20 (Berlin, Heidelberg: Springer Berlin Heidelberg).

Allman A, Zhang Q (2021) Branch-and-price for a class of nonconvex mixed-integer nonlinear programs. *Journal of Global Optimization* 1–20.

Atamtürk A, Narayanan V (2008) Polymatroids and mean-risk minimization in discrete optimization. *Operations Research Letters* 36(5):618–622.

Atamtürk A, Narayanan V (2009) The submodular knapsack polytope. *Discrete Optimization* 6(4):333–344.

Bärmann A, Burlacu R, Hager L, Kleinert T (2021) On piecewise linear approximations of bilinear terms: Structural comparison of univariate and bivariate mixed-integer programming formulations .

Batun S, Denton BT, Huschka TR, Schaefer AJ (2011) Operating room pooling and parallel surgery processing under uncertainty. *INFORMS Journal on Computing* 23(2):220–237.

Ben-Tal A, Nemirovski A (2001) On polyhedral approximations of the second-order cone. *Mathematics of Operations Research* 26(2):193–205.

Bergman D (2019) An exact algorithm for the quadratic multiknapsack problem with an application to event seating. *INFORMS Journal on Computing* 31(3):477–492.

Berjón D, Gallego G, Cuevas C, Morán F, García N (2015) Optimal piecewise linear function approximation for gpu-based applications. *IEEE Transactions on Cybernetics* 46(10):2584–2595.

Berthold T, Heinz S, Vigerske S (2012) Extending a cip framework to solve miqcps. Lee J, Leyffer S, eds., *Mixed Integer Nonlinear Programming*, 427–444 (New York, NY: Springer New York).

Bertsimas D, Gupta V, Kallus N (2018) Robust sample average approximation. *Mathematical Programming* 171(1):217–282.

Bliek C, Bonami P, Lodi A (2014) Solving mixed-integer quadratic programming problems with IBM-CPLEX: A progress report. *Proceedings of the twenty-sixth RAMP Symposium*, 16–17.

Bonami P, Tramontani A (2015) Recent improvement to misocp in cplex. Technical report.

Cacchiani V, Iori M, Locatelli A, Martello S (2022) Knapsack problems-an overview of recent advances. part ii: Multiple, multidimensional, and quadratic knapsack problems. *Computers & Operations Research* 105693.

Caprara A, Pisinger D, Toth P (1999) Exact solution of the quadratic knapsack problem. *INFORMS Journal on Computing* 11(2):125–137.

Cardoen B, Demeulemeester E, Beliën J (2010) Operating room planning and scheduling: A literature review. *European journal of operational research* 201(3):921–932.

Ceselli A, Létocart L, Traversi E (2022) Dantzig–wolfe reformulations for binary quadratic problems. *Mathematical Programming Computation* 1–36.

Charnes A, Cooper WW (1963) Deterministic equivalents for optimizing and satisficing under chance constraints. *Operations Research* 11(1):18–39.

Coey C, Lubin M, Vielma JP (2020) Outer approximation with conic certificates for mixed-integer convex problems. *Mathematical Programming Computation* 12(2):249–293.

Cohen MC, Keller PW, Mirrokni V, Zadimoghaddam M (2019) Overcommitment in cloud services: Bin packing with chance constraints. *Management Science* 65(7):3255–3271.

D'Ambrosio C, Frangioni A, Gentile C (2019) Strengthening the sequential convex MINLP technique by perspective reformulations. *Optimization Letters* 13(4):673–684.

D'Ambrosio C, Lee J, Wächter A (2012) An algorithmic framework for minlp with separable non-convexity. Lee J, Leyffer S, eds., *Mixed Integer Nonlinear Programming*, 315–347 (New York, NY: Springer New York).

Delorme M, Iori M, Martello S (2016) Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research* 255(1):1–20.

Deng Y, Shen S, Denton B (2019) Chance-constrained surgery planning under conditions of limited and ambiguous data. *INFORMS Journal on Computing* 31(3):559–575.

Denton BT, Miller AJ, Balasubramanian HJ, Huschka TR (2010) Optimal allocation of surgery blocks to operating rooms under uncertainty. *Operations Research* 58(4 PART 1):802–816.

Farley AA (1990) Note on bounding a class of linear programming problems, including cutting stock problems. *Operations Research* 38(5):922–923.

Foster BA, Ryan DM (1976) An integer programming approach to the vehicle scheduling problem. *Operational Research Quarterly (1970-1977)* 27(2):367–384.

Furini F, Traversi E (2019) Theoretical and computational study of several linearisation techniques for binary quadratic problems. *Annals of Operations Research* 279(1):387–411.

Gamrath G, Anderson D, Bestuzheva K, Chen WK, Eifler L, Gasse M, Gemander P, Gleixner A, Gottwald L, Halbig K, Hendel G, Hojny C, Koch T, Le Bodic P, Maher SJ, Matter F, Miltenberger M, Mühmer E, Müller B, Pfetsch ME, Schlösser F, Serrano F, Shinano Y, Tawfik C, Vigerske S, Wegscheider F, Weninger D, Witzig J (2020) The SCIP Optimization Suite 7.0. Technical report, Optimization Online.

Gamrath G, Lübbecke ME (2010) Experiments with a generic dantzig-wolfe decomposition for integer programs. *Proceedings of the 9th International Conference on Experimental Algorithms*, 239–252, SEA'10 (Berlin, Heidelberg: Springer-Verlag).

Geißler B, Martin A, Morsi A, Schewe L (2012) Using piecewise linear functions for solving minlps. Lee J, Leyffer S, eds., *Mixed Integer Nonlinear Programming*, 287–314 (New York, NY: Springer New York).

Ghaoui LE, Oks M, Oustry F (2003) Worst-case value-at-risk and robust portfolio optimization: A conic programming approach. *Operations Research* 51(4):543–556.

Gilmore PC, Gomory RE (1961) A linear programming approach to the cutting-stock problem. *Operations Research* 9(6):849–859.

Gleixner A, Maher SJ, Müller B, Pedroso JP (2020) Price-and-verify: a new algorithm for recursive circle packing using Dantzig–Wolfe decomposition. *Annals of Operations Research* 284(2):527–555.

Goyal V, Ravi R (2010) A PTAS for the chance-constrained knapsack problem with random item sizes. *Operations Research Letters* 38(3):161–164.

Joncour C, Michel S, Sadykov R, Sverdlov D, Vanderbeck F (2010) Column generation based primal heuristics. *Electronic Notes in Discrete Mathematics* 36(C):695–702.

Lübbecke M, Puchert C (2012) Primal heuristics for branch-and-price algorithms. *Operations Research Proceedings 2011*, 65–70 (Springer).

Luedtke J, Ahmed S (2008) A sample approximation approach for optimization with probabilistic constraints. *SIAM Journal on Optimization* 19(2):674–699.

Ni W, Shu J, Song M, Xu D, Zhang K (2021) A branch-and-price algorithm for facility location with general facility cost functions. *INFORMS Journal on Computing* 33(1):86–104.

Olivier P, Lodi A, Pesant G (2021) The quadratic multiknapsack problem with conflicts and balance constraints. *INFORMS Journal on Computing* 33(3):949–962.

Puchinger J, Raidl GR, Pferschy U (2010) The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing* 22(2):250–265.

Sadykov R, Vanderbeck F (2013) Bin packing with conflicts: A generic branch-and-price algorithm. *INFORMS Journal on Computing* 25(2):244–255.

Shylo OV, Prokopyev OA, Schaefer AJ (2013) Stochastic operating room scheduling for high-volume specialties under block booking. *INFORMS Journal on Computing* 25(4):682–692.

Tawarmalani M, Sahinidis NV (2005) A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming* 103(2):225–249.

Vance PH, Barnhart C, Johnson EL, Nemhauser GL (1994) Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications* 3(2):111–130.

Vielma JP, Ahmed S, Nemhauser G (2010) Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions. *Operations Research* 58(2):303–315.

Wei L, Lai M, Lim A, Hu Q (2020a) A branch-and-price algorithm for the two-dimensional vector packing problem. *European Journal of Operational Research* 281(1):25–35.

Wei L, Luo Z, Baldacci R, Lim A (2020b) A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. *INFORMS Journal on Computing* 32(2):428–443.

Zhang Y, Jiang R, Shen S (2018) Ambiguous chance-constrained binary programs under mean-covariance information. *SIAM Journal on Optimization* 28(4):2922–2944.

Zhang Z, Denton BT, Xie X (2020) Branch and price for chance-constrained bin packing. *INFORMS Journal on Computing* 32(3):547–564.