Branch and price for submodular bin packing

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

Branch and price for submodular bin packing

Liding Xu[a], Claudia D'Ambrosio[a], Sonia Haddad Vanier[a], Emiliano Traversi[b]

[a]*LIX CNRS, École Polytechnique, Institut Polytechnique de Paris, Palaiseau, 91128, France*
[b]*LIPN CNRS, Université Sorbonne Paris Nord, Villetaneuse, 93430, France*

Abstract

The Submodular Bin Packing (SMBP) problem asks for packing unsplittable items into a minimal number of bins for which the capacity utilization function is submodular. SMBP is equivalent to chance-constrained and robust bin packing problems under various conditions. SMBP is a hard binary nonlinear programming optimization problem. 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 a piece-wise linear relaxation to solve them. To speed up column generation, we develop a hybrid pricing strategy to replace the exact pricing algorithm with a fast pricing heuristic. We test our algorithms on instances generated as suggested in the literature. The computational results show the efficiency of our branch-and-price algorithm and the proposed pricing techniques.

Keywords: branch and price, submodular bin packing, submodular knapsack, piece-wise linear relaxation

1. Introduction

Bin packing (BP) is an important combinatorial optimization problem with applications in various fields, including call centers, healthcare, con-

Email addresses: lidingxu.ac@gmail.com (Liding Xu), dambrosio@lix.polytechnique.fr (Claudia D'Ambrosio), sonia.vanier@lix.polytechnique.fr (Sonia Haddad Vanier), traversi@lipn.univ-paris13.fr (Emiliano Traversi)

tainer shipping, and cloud computing. These applications are typically modeled as BP problems that aim to pack unsplittable items into a minimum number of bins, with a capacity constraint on each bin. Formally, a BP problem can be written as the following Binary Linear Programming (BIP) problem:

$$\min \quad \sum_{j\in\mathcal{M}} y_j, \tag{1a}$$

$$\text{s. t.} \quad \sum_{i\in\mathcal{N}} \mu_i v_{ij} \le c y_j, \qquad \forall j \in \mathcal{M}, \tag{1b}$$

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

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

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

where $\mathcal{M} := \{1, \cdots, m\}$ is the index set of potential bins (m is the number of potential bins), $\mathcal{N} := \{1, \cdots, n\}$ is the index set of items (n is the number of items), c is the capacity which is the same for every bin, and μ_i is the size of item i. Variable y_j decides whether bin j is used, and variable v_{ij} indicates whether item i is allocated to bin j. Capacity constraints (1b) stipulate that the capacities of bins are not exceeded, and set partition constraints (1c) require that each item is exactly allocated to one bin.

In many practical applications of BP, nominal item sizes μ are not revealed before the allocation decision is made, so uncertainty arises. Probabilistic modeling of capacity constraints (1b) allows item sizes μ to be random parameters, and thus the uncertainty is taken as a probability distribution on μ. We consider two commonly used probabilistic BP models. The first probabilistic model is the BP with chance constraints (BPCC) [1]. By assuming item sizes μ following a given (multivariate) probability distribution, BPCC requires that each capacity constraint in (1b) should be respected with a probability at least α, written as the following chance constraints [2]:

$$\mathbb{P}(\sum_{i\in\mathcal{N}} \mu_i v_{ij} \le c y_j) \ge \alpha, \qquad \forall j \in \mathcal{M}. \tag{2}$$

The second probabilistic model is the distributionally robust BP (DRBP) [3, 4]. It models the worst case of chance constraints [5]. More specifically, given a family $\mathscr{D}$ of probability distributions of μ, DRBP requires that each

chance constraint in (2) should be respected for any probability distribution within $\mathscr{D}$. Thereby, capacity constraints of DRBP can be formulated as the following distributionally robust constraints

$$\inf_{\mu \sim \mathscr{D}} \mathbb{P}(\sum_{i \in \mathcal{N}} \mu_i v_{ij} \leq c y_i) \geq \alpha, \qquad \forall j \in \mathcal{M}. \tag{3}$$

Computational optimization of BPCC and DRBP models is challenging due to probabilistic constraints. Stochastic optimization methods can tackle mathematical optimization problems with probabilistic constraints. The sample average approximation (SAA) is a common stochastic optimization method for chance-constrained and distributionally robust optimization problems [6, 7]. It approximates these problems as two/multi-stage Mixed-Integer Linear Programming (MILP) problems and computes approximate solutions that converge to an optimal solution in a probabilistic sense. Previous works, such as [3, 8, 9], apply tailored SAA methods to solve BPCC and DRBP.

Several recent works show that, under various assumptions on probabilistic distributions, BPCC and DRBP are equivalent to or well-approximated by a deterministic optimization problem, namely, submodular BP (SMBP). It is shown in [4] that, BPCC has an SMBP formulation, if item sizes μ follow independent Gaussian distributions; SMBP also provides an upper bound for BPCC with item sizes μ under general independent distributions over bounded intervals (we note that then SMBP becomes a restriction of BPCC, and thus its solution is always feasible to BPCC.). It is shown in [10] that DRBP has an SMBP formulation, if distributions in $\mathscr{D}$ have the same mean values and the same diagonal covariance matrix.

Given the applicability of the previous assumptions, SMBP is an appealing alternative formulation to BPCC and DRBP, as it can be solved optimally in a finite time, while the convergence rate of SAA methods for BPCC and DRBP depends on the number of samples. SMBP already finds its applications in cloud computing [4], surgery planning [11], and operating room planning [12]. The environment is highly dynamic for these applications, and uncertainty plays a significant role in practical models. These applications give rise to a need for efficient algorithms to solve SMBP.

In this paper, we study the exact algorithms for solving SMBP. SMBP has the following Binary Nonlinear Programming formulation:

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

$$\text{s. t.} \quad \sum_{i\in\mathcal{N}} a_i v_{ij} + \sigma\sqrt{\sum_{i\in\mathcal{N}} b_i v_{ij}} \le c y_j, \qquad \forall j \in \mathcal{M}, \tag{4b}$$

$$\sum_{j\in\mathcal{M}} v_{ij} = 1, \qquad \forall i \in \mathcal{N}, \tag{4c}$$

$$v_{ij} \in \{0,1\}, \qquad \forall i \in \mathcal{N}, j \in \mathcal{M}, \tag{4d}$$

$$y_j \in \{0,1\}, \qquad \forall j \in \mathcal{M} \tag{4e}$$

where a_i, b_i are parameters inferred from the distribution of μ_i. This formulation is a compact nonlinear version of (1). We remark that the left-hand side of the constraint (4b) is a submodular function over x [13], so SMBP is named after this function. A constraint in the form of (4b) with y_j fixed to 1 is called a submodular knapsack constraint.

To solve SMBP, the state-of-art exact algorithm uses general-purpose integer-programming solvers to solve its Binary Second-Order Conic Programming (BSOCP) reformulation [10], which valid inequalities can further strengthen [13]. The experiment in [10] shows that small instances with item number n up to 40 and bin numbers m up to 10 can be solved to optimality by this exact algorithm.

The intuition underlying this paper is that the Dantzig-Wolfe (DW) decomposition is a promising approach to tackling large-scale classical BPs: a 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 [14]. The branch-and-price algorithm integrates column generation with the branch-and-bound algorithm. It is the state-of-the-art exact algorithm for solving DW decomposition of classical BPs [15, 16].

We propose the first DW decomposition and set cover formulation for SMBP, and design a branch-and-price algorithm with tailored methods for solving pricing problems. After our DW decomposition of SMBP, the nonlinearity moves to the pricing submodular knapsack subproblem, which has a linear objective function and a submodular capacity constraint. One can avoid the growing number of nonlinear constraints (4b) in the compact formulation (4), when solving larger instances.

The DW decomposition provides a skeleton of our main algorithm. The

techniques for solving general DW decomposition problems are vast, to name a few, we refer to [17] for stabilization techniques, [18] for lexicographic pricing, [19] for goal cuts and early termination, and [20] for non-robust cuts. In [21], a simple parameterization enables the use of several advanced techniques in the branch-cut-and-price VRPsolver [22]: automatic stabilization by smoothing [23], limited-memory rank-1 cuts [24], enumeration, hierarchical strong branching over accumulated resources [25], and limited discrepancy search diving heuristics. In this paper, we focus on algorithmic innovation that exploits the specific nonlinear structure of pricing problems and new techniques to speed up the convergence of column generation.

As the study in [10] for the compact formulation, the nonlinearity is a crucial feature for model representability, so it is unavoidable and needs a special algorithmic treatment. In our case, pricing problems have submodular knapsack constraints involving nonlinear functions. We give two different views of nonlinearity. First, we can represent the submodular knapsack constraint via second-order constraints, which many general solvers then accept. Alternatively, we propose a non-convex Mixed-Binary Quadratically Constrained Programming (MBQCP) formulation for the submodular knapsack. A Piece-Wise Linear (PWL) function is linear in each partition of its domain and can be modeled by a MILP formulation [26]. PWL functions have been used to approximate or relax non-convex Mixed-Integer Nonlinear Programming (MINLP) problems [27]. Despite its non-convexity, the critical feature of the MBQCP formulation is that its only nonlinear function is a univariate quadratic function, which is easy to approximate using a PWL function. We construct the PWL relaxation for the submodular knapsack and combine it with cutting planes to form an exact PWL relaxation-based branch-and-cut (PWL-B&C) algorithm.

The submodular knapsack is essential as it models the chance-constrained knapsack problem [28]. We thus provide an approach for solving submodular knapsack problems different from the pure valid inequality approach in [13, 29].

We propose several strategies to accelerate the convergence of the branch-and-price algorithm, i.e., improve primal and dual bounds. The Farley bound [30, 31] is an early valid dual bound before the termination of the column generation procedure [15, 32]. 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, the exact pricing algorithm is unnecessary, so we can use fast pricing heuristic. Our branch-and-price

algorithms use a hybrid pricing strategy to speed up the column generation procedure. The hybrid pricing strategy is thus an intermediate between exact pricing and heuristic pricing strategies [33].

There are few publicly available instances of SMBP problems. In [4], there is a method to generate instances from BPCC and DRBP under various distributions. We generate instances of three different scales by this method and conduct computational experiments on them. We implement our branch-and-price algorithm for the set cover formulation of SMBP and find that it outperforms existing methods, which solve the compact BSOCP formulation [10]. Our core innovation, PWL-B&C pricing algorithm, and hybrid pricing strategy, significantly improve the branch-and-price algorithm.

1.1. Literature review

As mentioned, there are several steps of transformation from BP with uncertainty to SMBP. We review these transformations and algorithms for solving associated transformed models.

The surgery planning problem is a typical application of BP with uncertainty in healthcare, where the surgery duration (item size) is assumed to be stochastic. Some pioneering works [8, 9] allow violations to capacity constraints (the left-hand side of (1b) thus can be greater than the capacity) rather than consider chance constraints. To minimize these violations, they use a penalty approach by adding the expectation of the sum of these violations into the objective function. Therefore, the transformed BP model is a standard stochastic optimization problem called stochastic BP (SBP). SBP can be further modeled and solved as a stochastic two-stage mixed-integer programming problem: the first stage variables are the bin variables y, the second stage variables are the item variables v, and the expected violation is the second stage objective. In some works [34, 35], only the expected penalty is considered in the models.

Compared to SBP, BPCC, and DRBP can control the violation of each capacity constraint with a guaranteed probability bound, and thus they are more accurate models for BP with uncertainty. To solve BPCC and DRBP, there are approximation algorithms and exact algorithms. (Sampling-based) approximation algorithms usually converge asymptotically to an optimal solution when the sampling number increases (as SAA methods), and exact algorithms usually converge in finite time. However, exact algorithms are mostly available for deterministic optimization problems.

In [36], a variant of BPCC in surgery planning is studied: items (surgeries) are allocated to a given set of bins (time blocks), the goal is to minimize the sum of expected capacity residuals (undertime), subject to chance constraints for overuse of bins' capacities (overtime). Assuming that the operation duration follows a multivariate normal distribution, the authors reformulate the problem as a deterministic optimization problem containing a convex objective and submodular capacity constraints. In [1], a special BPCC with the probability distribution over finite support is studied. It has a BIP formulation, which an exact algorithm can solve. As mentioned before, for various probabilistic distributions [4, 10], BPCC and DRBP admit a deterministic SMBP reformulation, which can be solved by exact algorithms.

Regarding solution algorithms, an SAA-based algorithm [3] can solve BPCC approximately, whose scenario subproblem is solved exactly by a DW decomposition approach. In addition, SMBP reformulation of DRBP [3] can be approximated by a MILP, which is solved by a DW decomposition approach. The only tailored exact algorithm for BPCC and DRBP [10] solves their compact SMBP reformulations. As for the deterministic variant of BPCC, the authors of [36] propose an exact outer approximation algorithm enhanced with PWL relaxation of submodular knapsack constraints, and the algorithm is a multi-search tree method, i.e., an underlying MILP solver will be called multiple times. In conclusion, no exact algorithm based on DW decomposition exists to solve SMBP. Meanwhile, DW decomposition is already used for various approximated problems.

This exception may be due to the lack of efficient exact algorithms to solve submodular knapsack problems. We note that in [3], DW decomposition is *de facto* applied to a MILP problem, and thus the pricing problems are also MILPs such as classical knapsack problems [37], which can be solved efficiently by general-purpose integer programming solvers. The efficiency of general solvers is mostly due to the lifted cover inequalities, which are strong valid inequalities for knapsack polytope and can be constructed via sequence-independent lifting [38]. However, for submodular knapsack, the computation of lifted cover inequalities is not tractable [29]. As we know from the literature, the tailored algorithm can be much better than general solvers for many variants of classical knapsack, because these algorithms can exploit more problem structures than general solvers. To name a few, we refer to the quadratic knapsack [39, 40], the multidimensional knapsack [41], and the quadratic multi-knapsack [42, 43].

Although general solvers are almost as complex as a black box for users,

we can at least understand how they solve the submodular knapsack. The submodular knapsack can be reformulated as a BSOCP problem, which is an acceptable formulation to `CPLEX` [44] and `SCIP` [45]. These solvers implement LP outer approximation-based branch-and-cut (LP-B&C) algorithm [46] to solve the BSOCP or general Mixed-Integer Second-Order Conic Programming (MISOCP) problems. The LP outer approximation is sometimes called the polyhedral outer approximation (or polyhedral relaxation). In fact, any second-order conic program (SOCP) is polynomially reducible to a linear program [47]. As for a submodular knapsack constraint, general solvers will linearize it into an intersection of a set of classical knapsack constraints, thus, inefficiency arises if too many linearizations are applied.

On the other hand, there are alternative exact approaches to solve some classes of nonconvex MINLPs using PWL relaxations [48, 27]. For example, [48] obtains 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.

Regarding the submodular knapsack, its BSOCP (a convex MINLP) formulation has a nonlinear function over all the problem variables, which is difficult to approximate when the dimension is high. On the other hand, the nonconvex MBQCP formulation, where the only nonlinear function is a univariate quadratic function on a slack variable. The resulting PWL relaxation in the experiment is stronger than pure polyhedral relaxation. 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. [36] only uses PWL relaxations to approximate the submodular knapsack. It requires refining PWL relaxations to achieve convergence, so their multi-search tree algorithm needs to restart the MILP solver from scratch in each iteration. In contrast, we prefix PWL relaxations and use cutting planes to achieve convergence. Therefore, our single-search tree does not need to restart the MILP solver.

We look at the recent development of DW decomposition and branch-and-price algorithm for solving MINLPs (see [49]), such as recursive circle packing (RCP) problems [32], binary quadratic problems [50], and facility location with general nonlinear facility cost functions [51]. There may be several ways to divide a MINLP into master and subproblems, so a MINLP may admit different DW decompositions. In [50], the authors 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. Since pricing problems are solved in thousands of iterations, [32] shows that any improvement in the pricing algorithm can speed up the convergence of column generation.

1.2. Contribution

In summary, our contribution in this paper is threefold. As far as we know, the previous work applies DW decomposition for an approximated MILP for SMBP, and thus nonlinearity is not considered in the solving process. So, we are the first to apply DW decomposition for SMBP with the nonlinearity considered. Built on the basic DW decomposition and branch-and-price algorithm, we develop a new hybrid pricing strategy technique to speed up the column generation, which can avoid computationally expensive exact pricing while not worsening the dual bound. Second, for pricing submodular knapsack problems, we propose a new MBQCP formulation and its PWL relaxation, and design a new PWL-B&C algorithm as an alternative exact algorithm to the conventional LP-B&C algorithm, which is based on valid inequalities. Finally, we perform computational experiments on many instances to evaluate the proposed algorithms. The computational results show that our tailored branch-and-price algorithms for DW reformulation outperform the conventional branch-and-cut algorithm for BSOCP formulation implemented in a state-of-art commercial solver; and the PWL-B&C algorithm can be a standalone algorithm for submodular knapsack. The source code and benchmark are released on our project website https://github.com/lidingxu/cbp.

1.3. Outline

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

2. Set cover formulation

In this section, we propose a new set cover formulation for SMBP. The formulation is derived similarly to the DW decomposition of the classical linear BP [16]. 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 (4b). The set cover formulation is based on enumerating all feasible columns, the number of which can be exponential to the number of items.

Set notation:

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

Decision variables:

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

We obtain the following set cover formulation for 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. The set cover reformulation already finds applications in vehicle routing [24] and unsplittable multi-commodity flows [52] problems.

The compact formulation (4) is a MINLP, but the set cover formulation (5) is a MILP. Moreover, the number of nonlinear constraints in the compact formulations equals the number of potential bins. The nonlinearity of the set cover formulation is *de facto* 'hidden' in the pricing subproblems, and each pricing subproblem has only one nonlinear constraint.

Remark 1. *(Modelling BSOCP constraints) We give a way to obtain a BSOCP formulation of the constraint* $\sum_{i\in\mathcal{N}} a_i v_{ij} + \sqrt{\sum_{i\in\mathcal{N}} b_i v_{ij}} \leq d$*, where* $d = cy_i$ *or* $d = c$*. Since* $v_{ij} \in \{0,1\}$*, the square root* $\sqrt{\sum_{i\in\mathcal{N}} b_i v_{ij}}$ *equals* $\sqrt{\sum_{i\in\mathcal{N}} b_i v_{ij}^2}$*. Let* $d' := d - \sum_{i\in\mathcal{N}} a_i v_{ij}$*, then the constraint is equivalent to a SOCP constraint* $\sqrt{\sum_{i\in\mathcal{N}} b_i v_{ij}^2} \leq d'$*. One can transform the SOCP constraint into several 3d SOCP constraints like* $x_1 \geq \sqrt{x_2^2 + x_3^2}$*, which is acceptable by* `CPLEX` *[53].*

Through the above discussion, we can obtain a BSOCP formulation of SMBP (4). When comparing two formulations, a formulation is said to be "stronger" if it gives a better dual bound.

Proposition 2.1. *The linear relaxation of the set cover formulation* (5) *is stronger than the continuous SOCP relaxation of the BSOCP formulation of* (4)*.*

The proof can be found in Appendix A.

3. Branch and price

Solving the set cover formulation with an exponential number of binary variables is challenging. In this section, we present an exact branch-and-price algorithm for solving the set-cover formulation of 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, the column generation, the primary heuristic, and the dual bound computation.

3.1. Branching rule

Our branch-and-price algorithm uses the Ryan/Foster branching rule [54]. 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, whose nodes are constructed and selected by `SCIP`'s internal rules [55] in our implementation. 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*.

Given a solution λ of the LP relaxation, if λ is not integral, the branching rule chooses an item pair to branch. It first creates an n-by-n matrix, and computes its entries as $M_{i_1 i_2} = \sum_{p \in \mathcal{P}_{\mathcal{S},\mathcal{D}} : d_{i_1 p} = d_{i_2 p} = 1} \lambda_p$ for all $i_1, i_2 \in \mathcal{N}$. Since, for an integral solution, $M_{i_1 i_2}$ must be either 0 or 1, the branching rule chooses the most fractional entry (i'_1, i'_2) such that $i'_1, i'_2 = \operatorname{argmin}_{i'_1, i'_2 \in \mathcal{N}} |0.5 - M_{i'_1 i'_2}|$. Then, the rule adds (i'_1, i'_2) to $\mathcal{S}, \mathcal{D}$, respectively.

3.2. Column generation

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

The column generation procedure starts with a subset of $(\mathcal{S}, \mathcal{D})$-feasible columns of the master LP problem, adds columns, and solves the restricted 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 objective value of the RMLP. Otherwise, the solution for the RMLP is also optimal for the master LP problem. The column with the most negative reduced cost is determined by solving a pricing problem.

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 to a *submodular 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 a negative reduced cost $1 - \sum_{i \in \mathcal{N}'} \pi'_i x_i$ and is added to the RMLP. Otherwise, the solution of the RMLP is optimal for the master LP, and the current node is solved. The details of the pricing algorithms can be found in Sect. 4.

Since pricing problems may not be solved optimally within a time limit, a pricing algorithm may find an existing column in $\mathcal{P}'_{\mathcal{S},\mathcal{D}}$ or a column with positive reduced cost. Therefore, adding the column does not improve the

RMLP and the column generation procedure is aborted. The following simple constraint can exclude existing solutions from the pricing problem, thus reducing the search space:

$$\sum_{i \in \mathcal{N}'} \pi'_i x_i \geq 1 + \epsilon, \tag{9}$$

where ϵ is a sufficiently small positive real number. Exact algorithms can easily add this constraint to exclude existing columns in $\mathcal{P}'_{\mathcal{S},\mathcal{D}}$. This constraint also guarantees that if solutions of negative reduced costs exist, then exact algorithms can find one of them.

3.3. Primal heuristics

We discuss primal heuristics that help find primal feasible solutions for the set covering formulation. We use two heuristics: The first heuristic uses an approximation algorithm to find a primal solution that forms a set $\mathcal{P}'$ of initial columns, and the second heuristic attempts to find a primal solution once a column has been generated and added to $\mathcal{P}'$.

[4] propose approximation algorithms to find a feasible solution with 8/3-ratio to the optimal solution to the submodular bin packing. Their algorithms are greedy and easy to implement, so we employ these algorithms as the first heuristic.

During column generation, each generated column could be combined with the previous columns in $\mathcal{P}'$ into a primal feasible solution. Our second primal heuristic is similar to the greedy column selection heuristic in [56, 57]. Once a column is generated, we force it into a potential solution. Then, we greedily select an existing column from $\mathcal{P}'$ that packs the maximum number of unpacked items until all items are packed. We note that the heuristic may find columns that do not improve the RMLP.

3.4. Dual bound computation

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

The optimum of the master LP problem is a local dual bound. However, the column generation procedure usually needs to solve many pricing problems to converge to this optimum. 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 3.1 ([30, 31])**.** *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 $v_{\mathrm{F}} \coloneqq \frac{v_{\mathrm{RMLP}}}{v_{\mathrm{price}}}$ 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 search tree node. After solving each pricing problem, the branch-and-price algorithm updates v_{ld} according to the following rule:

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

Early termination rules of [15] can compare the local dual bound and the primal bound to improve the branch-and-price algorithm. The rules exploit integrality and can terminate column generation earlier than the classical algorithm. We implement these rules in our branch-and-price solver.

4. Solving the pricing problem

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 pricing heuristic. 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 approximate nonlinear functions (or relax under some conditions) 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 pricing heuristic.

4.1. Pricing heuristic

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. find the set $\overline{\Delta} := \{i \in \mathcal{N}' \setminus \Delta : A + a'_i + \sigma\sqrt{B + b'_i} \leq c\}$ of items that can be added to the bin;
3. if $\overline{\Delta} = \emptyset$, exits and outputs Δ;
4. for each unadded item $i \in \overline{\Delta}$, computes the incremental capacity usage $\gamma_i := (A + a'_i + \sigma\sqrt{B + b'_i}) - (A + \sigma\sqrt{B})$, and the profit-over-usage ratio $r_i := \frac{\pi'_i}{\gamma_i}$;
5. adds the unadded item with the maximum r_i into Δ.

The fixing-greedy heuristic enforces, for each time, an 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 formulation of the pricing problem (8) is similar to the BSOCP formulation of SMBP (4).

Applying the same technique in Remark. 1, the BSOCP formulation of the pricing problem is:

$$\max \quad \sum_{i\in\mathcal{N}'} \pi'_i x_i, \tag{10a}$$

$$\text{s. t.} \quad \sum_{i\in\mathcal{N}'} a'_i x_i + \sigma\sqrt{\sum_{i\in\mathcal{N}'} b'_i x_i^2} \leq c, \tag{10b}$$

$$x_{i_1} + x_{i_2} \leq 1, \qquad \forall (i_1, i_2) \in \mathcal{D}', \tag{10c}$$

$$x_i \in \{0, 1\}, \qquad \forall i \in \mathcal{N}'. \tag{10d}$$

Where (10b) can be represented by 3d second-order conic constraints. The BSOCP formulation (10) is a convex MINLP formulation.

In this section, we analyze the polyhedral outer approximation of the BSOCP formulation (10) and show that a finite number of cutting planes is sufficient to define an exact MILP reformulation of the BSOCP formulation (10).

To simplify the presentation, we use the following notation:

- the left-hand side of (10b):

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

- the binary set defined by (10b):

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

- 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 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 underestimator 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 $f(x) \leq c$.

A polyhedral outer approximation $\mathcal{O}$ is said *exact*, if $\mathcal{O} \cap \{0,1\}^n = \mathcal{C}$. So, solving the optimization problem over an exact polyhedral outer approximation with binary and conflict constraints is equivalent to solving the submodular knapsack problem with conflicts. Next, we identify a family of valid inequalities that give an exact polyhedral outer approximation. Each of these valid inequalities corresponds to a binary point not in $\mathcal{C}$.

Theorem 4.1. *Given a point $\hat{x} \in \{0,1\}^{\mathcal{N}'}$, the following inequality is valid for $\mathcal{C}$ and $\overline{\mathcal{C}}$:*

$$\sum_{i \in \mathcal{N}'} a'_i x_i + \frac{\sigma}{\sqrt{\sum_{i \in \mathcal{N}'} b'_i \hat{x}_i}} \sum_{i \in \mathcal{N}'} b'_i \hat{x}_i x_i \leq c. \tag{11}$$

Let

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

Moreover,

1. *if $\hat{x} \notin \mathcal{C}$, the valid inequality is violated by $\hat{x}$;*
2. *$\mathcal{O}$ is exact, and $\mathcal{C} = \mathcal{O} \cap \{0,1\}^{\mathcal{N}'}$.*

Proof. Since function f is 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 (11) in the statement is valid for $\mathcal{C}$. The left-hand side of inequality (11) 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.

Let us consider $x^* \in \{0,1\}^{\mathcal{N}'}$. If $x^* \notin \mathcal{C}$, then x^* violates the $\mathcal{L}^f_{x^*}(x) \leq c$ 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, $\mathcal{C} = \mathcal{O} \cap \{0,1\}^{\mathcal{N}'}$. □

Looking at the above theorem, we find that each binary point not in $\mathcal{C}$ gives rise to a valid inequality separating it from $\mathcal{C}$. Moreover, binary points in $\mathcal{C}$ satisfy these valid inequalities, i.e., they are in the polyhedral

outer approximation $\mathcal{O}$. We define two sets related to the polyhedral outer approximation $\mathcal{O}$. The *generating set* is defined as

$$\mathcal{X} := \{\hat{x} \in \{0,1\}^{\mathcal{N}'} : \hat{x} \notin \mathcal{C}\}, \tag{12}$$

because it generates the following *cut coefficient set*:

$$\Theta := \left\{ \theta \in \mathbb{R}^{\mathcal{N}'} : \exists \hat{x} \in \mathcal{X}\, \forall i \in \mathcal{N}'\, \theta_i = a'_i + \frac{\sigma}{\sqrt{\sum_{i\in\mathcal{N}'} b'_i \hat{x}_i}} b'_i \hat{x}_i \right\}. \tag{13}$$

By Thm. 4.1, $\mathcal{O}$ is an exact polyhedral outer approximation, so replacing $x \in \mathcal{C}$ with $x \in \mathcal{O}$ does not change the binary feasible set. This gives rise to an exact MILP formulation equivalent to the submodular knapsack problem with conflicts:

$$\begin{aligned}
\max \quad & \sum_{i\in\mathcal{N}'} \pi'_i x_i, && && \text{(14a)}\\
\text{s. t.} \quad & \theta^\top x \le c, && \forall \theta \in \Theta && \text{(14b)}\\
& x_{i_1} + x_{i_2} \le 1, && \forall (i_1, i_2) \in \mathcal{D}', && \text{(14c)}\\
& x_i \in \{0,1\}, && \forall i \in \mathcal{N}'. && \text{(14d)}
\end{aligned}$$

However, $\mathcal{X}$ (and hence Θ) is unknown before exploring the search space, 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 $\hat{x}$ is found in $\mathcal{X}$. Off-the-shelf solvers do not use this finite family of cuts, but it is a crucial component for constructing our PWL-B&C algorithm in Sect. 4.5.

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

Lemma 4.1 ([47])**.** *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 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:

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

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

$$\sigma^2 \sum_{i\in\mathcal{N}'} b'_i x_i \le (c-w)^2, \tag{15c}$$

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

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

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

Although the program contains a concave quadratic constraint (15c), the nonlinearity is only a univariate quadratic function compared to the $|\mathcal{N}'|$-dimensional nonlinear SOC function f in (10b).

4.4. PWL relaxation

A Piece-Wise Linear (PWL) function is linear on each piece of a given partition of its domain. We derive a MILP relaxation of the MBQCP formulation (15) based on the PWL relaxation for the quadratic function, and refer to this new MILP relaxation as the PWL relaxation. The approximation error of the optimal PWL relaxation is discussed in this section. Let us denote by $q(w) := (c-w)^2$ the univariate quadratic function. We denote a value of the slack variable w in the constraint (15c) as a *breakpoint*. Given an ordered set of breakpoints $\mathcal{B} = (w_1, w_2, \ldots, w_h)$ such that $w_k \in [\underline{w}, \overline{w}]$ ($k \in [h] := \{1, \cdots, h\}$), $w_1 = \underline{w}$ 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.

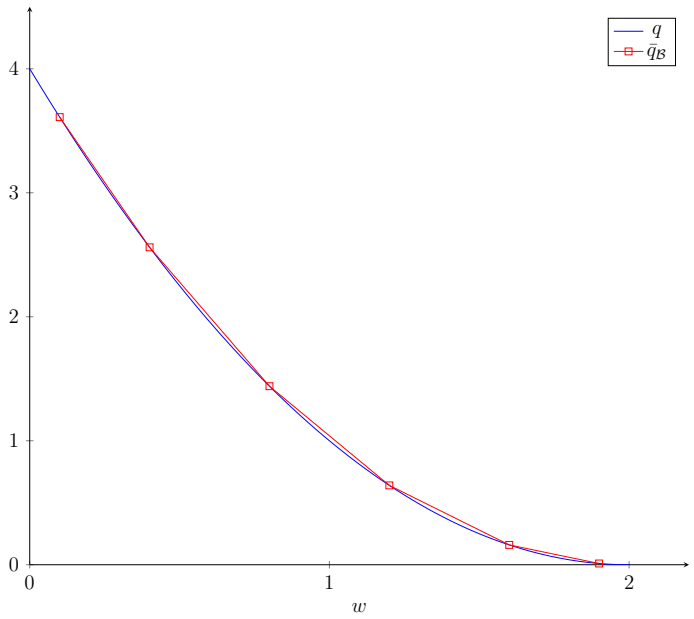

Figure 1: Graphs of the quadratic and its PWL over-estimator

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

Assume that we are given the breakpoints $\mathcal{B}$. Replacing $\sigma^2 \sum_{i \in \mathcal{N}'} b'_i x_i \leq q(w)$ with $\sigma^2 \sum_{i \in \mathcal{N}'} b'_i x_i \leq \bar{q}_{\mathcal{B}}(w)$ in the constraint (15c), we obtain the following PWL relaxation of the MBQCP formulation (15):

$$\max \quad \sum_{i \in \mathcal{N}'} \pi'_i x_i, \tag{16a}$$

$$\text{s. t.} \quad \sum_{i \in \mathcal{N}'} a'_i x_i = w, \tag{16b}$$

$$\sigma^2 \sum_{i \in \mathcal{N}'} b'_i x_i \leq \bar{q}_{\mathcal{B}}(w), \tag{16c}$$

$$x_{i_1} + x_{i_2} \leq 1, \qquad \forall (i_1, i_2) \in \mathcal{D}', \tag{16d}$$

$$x_i \in \{0, 1\}, \qquad \forall i \in \mathcal{N}', \tag{16e}$$

$$w \in [0, c]. \tag{16f}$$

Remark 2. *(Modeling PWL functions) The graphs of PWL functions have several MILP formulations, see [26]. In this paper, we consider the logarithmic model. We denote by z the auxiliary binary variables introduced in the MILP formulation of $\bar{q}_{\mathcal{B}}$. From version 20.1.0 [58],* `CPLEX` *can automatically formulate $\bar{q}_{\mathcal{B}}$ to the logarithmic model and add auxiliary variables z in the internal data structure.*

The approximation error of a PWL relaxation is expressed as ℓ_p-norm of the difference between the approximation function and the target function.

Definition 4.1. *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 $\ell_p(\bar{q}_{\mathcal{B}}, q) := (\int_{\underline{w}}^{\overline{w}} |\bar{q}_{\mathcal{B}}(w) - q(w)|^p \, dw)^{\frac{1}{p}}$.*

Since the approximation error measures the quality of a PWL approximation to the quadratic function, thus it in turn measures the error of the PWL relaxation to the MBQCP formulation. Empirically, the optimal solution to a PWL relaxation with a small approximation error should have a small gap to the optimal solution of the submodular knapsack with conflicts. On the other hand, although adding breakpoints decreases the approximation error, it increases the computation resource to solve the PWL relaxation. So a common problem is understanding the best possible approximation error given a fixed number of breakpoints (limited computational resource).

This problem can be formalized as follows. Given an integer h (number of breakpoints), 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:

$$\min_{\mathcal{B} \in \mathbb{B}^h} \ell_p(\bar{q}_{\mathcal{B}}, q). \tag{17}$$

A convex program [27] can compute the ℓ_∞-approximation error for general nonconvex functions. An error analysis [59] gives asymptotically tight bounds to quantify the ℓ_2-approximation error.

The following theorem gives the best ℓ_∞-approximation error that we can achieve: an optimal solution to the breakpoint selection problem under the ℓ_∞-approximation error is an equidistant partition of $[\underline{w}, \overline{w}]$.

Theorem 4.2. *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* $1 \leq k \leq h$*, which yields the minimum* ℓ_∞*-approximation error* $\frac{(\overline{w}-\underline{w})^2}{4(h-1)^2}$ *for the break point selection problem* (17)*.*

The proof can be found in Appendix B. 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 4.1. *Let* $\epsilon > 0$*, then there exists a 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 [26], the result follows. □

Next, we summarize the approximation errors of two relaxations to their corresponding formulations. Note that we do not consider the integrality of the binary variable x'. Comparing Lemma 4.1 and Cor. 4.1, the approximation error of the PWL relaxation (16) to the MBQCP formulation (15) is independent of the number of variables, while the approximation error of the polyhedral outer approximation to the BSOCP formulation (10) depends on this number.

We remark that our PWL relaxation differs from [36]'s PWL relaxation. The constraint (15) of MBQCP formulation is equivalent to $\sigma\sqrt{\sum_{i\in\mathcal{N}'} b'_i x_i} \leq c - w$, and a PWL relaxation was used for the left-hand side concave function $\sigma\sqrt{\sum_{i\in\mathcal{N}'} b'_i x_i}$ in [36]. However, the optimal approximation error for such PWL relaxation has yet to be discovered.

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 (13) suffices to make the PWL relaxation exact. To solve it, we propose a combined formulation and a branch-and-cut algorithm based on the PWL relaxation (PWL-B&C).

$$\max \quad \sum_{i\in\mathcal{N}'} \pi'_i x_i, \tag{18a}$$

$$\text{s. t.} \quad \sum_{i\in\mathcal{N}'} a'_i x_i = w, \tag{18b}$$

$$\sigma^2 \sum_{i\in\mathcal{N}'} b'_i x_i \le \bar{q}_{\mathcal{B}}(w), \tag{18c}$$

$$\theta^\top x \le c, \qquad \forall \theta \in \Theta \tag{18d}$$

$$x_{i_1} + x_{i_2} \le 1, \qquad \forall (i_1, i_2) \in \mathcal{D}', \tag{18e}$$

$$x_i \in \{0,1\}, \qquad \forall i \in \mathcal{N}', \tag{18f}$$

$$w \in [0, c]. \tag{18g}$$

Formulation (18) combines the MILP formulation (14) with the (redundant) PWL relaxation. As already mentioned by Thm. 4.1, the MILP formulation (14) is an exact formulation for submodular knapsack problems with conflicts, so this combined formulation (18) is also exact.

The intuition underlying the combined formulation (18) is that we cannot add numerous valid inequalities (18d) *a priori*. In practice, we add them *lazily* to exclude infeasible binary solutions to the submodular knapsack with conflicts in the course of the search, and this method is typically supported or suggested by *lazy cut callbacks* of some solvers such as `CPLEX` and `SCIP`. However, in this way, we cannot control the initial relaxation quality given solely by a few valid inequalities from (18d). On the contrary, the PWL relaxation (18c) can be enforced *a priori*, and its quality is controllable (Thm. 4.2). So we can leverage it to reduce the initial search space and refine the relaxation by adding valid inequalities lazily. This intuition and formulation give rise to a tailored Algorithm 1 for submodular knapsack with conflicts, partly inspired by algorithms in [46]. We show in experiments that this formulation with redundant constraints (18b) and (18c) can be solved much faster than the standard BSOCP formulation (10). In practice, only a few cuts in (18d) must separate before the convergence.

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 a preprocessing 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 [46].

Bound tightening The bound tightening procedure is called before the branch-and-price algorithm to shrink 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 (15c) 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 non-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 (15c) 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:

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

Similarly, we can define the *upper breakpoint*. There exists some upper breakpoint $\overline{w} \in [0, c]$, such that, for every 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:

$$
\begin{aligned}
\overline{w} := \max \quad & \sum_{i \in \mathcal{N}'} a'_i x_i, \\
\text{s. t.} \quad & \sum_{i \in \mathcal{N}'} a'_i x_i + \sigma \sqrt{\sum_{i \in \mathcal{N}'} b'_i x_i^2} \leq c, \\
& x_{i_1} + x_{i_2} \leq 1, \qquad \forall (i_1, i_2) \in \mathcal{D}', \\
& x_i \in \{0, 1\}, \qquad \forall i \in \mathcal{N}'.
\end{aligned} \tag{20}
$$

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 root node set, 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 $\underline{w}_r \leq \underline{w}$ and $\overline{w} \leq \overline{w}_r$. 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 items in a bin. We take h as the solution value given by the heuristic algorithm and assign breakpoints h equidistantly in $[\underline{w}, \overline{w}]$. The equidistant partition gives the best approximation error according to Thm. 4.2 for a fixed number of breakpoints. 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 set of active nodes $\mathscr{N}$ of the search tree, a pool of cuts $\mathscr{C}$, an incumbent solution x^* ($\sum_{i \in \mathcal{N}'} \pi'_i x_i^*$ is a primal bound).

A node (l, u, U) is characterized by the finite variable boundary vectors l and u and the node's dual upper bound U. The upper bound U is inherited from its parent node and computed via the LP relaxation. Note that the PWL function is a modeling concept. We use a MILP solver, i.e., `CPLEX`, that formulates the PWL function $q_{\mathcal{B}}$ into a MILP. We denote by z the additional binary variables to model $\bar{q}_{\mathcal{B}}$ (see Sect. 4.4). The variables z are also constructed internally by `CPLEX`, 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 (16), cuts from $\mathscr{C}$, and other cuts added by the MILP solver.

Algorithm 1: PWL-B&C algorithm

Input: a submodular knapsack problem with conflicts (18), 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 (16) ; ▷ the PWL function is modeled by auxiliary binary variable z
initialize cut pool $\mathscr{C}$ to $\emptyset$, the node list $\mathscr{N}$ of $\mathscr{M}_{\mathcal{B}}(\mathscr{C}, l_0, u_0, \infty)$ with root node (l_0, u_0), incumbent solution $x^* = 0$, and the upper bound U of the root node to ∞;
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 $(\hat{x}, \hat{z})$;
    if *upper bound* $U \leq \sum_{i \in \mathcal{N}'} \pi_i \hat{x}_i$ **then**
        continue ; ▷ fathomed by bound
    else
        set U to $\sum_{i \in \mathcal{N}'} \pi_i \hat{x}_i$; ▷ update the dual upper bound
    end
    if $(\hat{x}, \hat{z})$ *is binary* **then**
        if $\hat{x}$ *satisfies capacity constraint* (8b) **then**
            set x^* to $\hat{x}$;
            continue ; ▷ fathomed by integrality
        else
            add separation cut to $\mathscr{C}$ by Thm. 4.1;
            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 6 of Algorithm 1, the main loop removes a node (l, u, U) from $\mathscr{N}$. Line 7 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 9 immediately fathoms the node by infeasibility. The upper bound U of the node means that any feasible solution to the combined formulation (18) that satisfies the bounds of the node for (x, z) has an objective value of at most U. Since LP is a relaxation of the combined formulation (18), any feasible solution to the combined formulation (18) that satisfies the bounds of the node for x has a objective value of at most U.

Line 12 fathoms the node by bound if U is not better than the incumbent value. Otherwise, the upper bound U of the node is set to the optimal value of LP on Line 14.

If $\hat{z}$ is not binary, then the PWL function is not implicitly enforced by the integrality of $\hat{z}$, so the algorithm should continue to branch. If $\hat{z}$ is binary (the PWL function is enforced) and $\hat{x}$ is binary, then the algorithm examines the solution $\hat{x}$.

If additionally, $\hat{x}$ is feasible (the capacity constraint is satisfied), then its objective value should be at least the upper bound U. Line 18 stores the new incumbent solution $\hat{x}$, and Line 19 fathoms the node since $\hat{x}$ is an optimal binary solution with respect to the bounds (l, u). Otherwise, the constraint (18d) is violated. Line 21 adds this constraint to the cut pool $\mathscr{C}$, Line 22 adds the current node for re-optimization, and Line 23 discards $\hat{x}$ by the cut in the next optimization iteration. Finally, $(\hat{x}, \hat{z})$ must be fractional on Line 26, the algorithm branches using the information from fractionality and U.

We remark that the idea in [36] for exact algorithms does not deploy cutting planes for approximating submodular knapsack, so in each iteration, new breakpoints are added to PWL, relaxations, and the underlying MILP solver needs restarts.

4.6. Hybrid pricing strategy

The pricing heuristic in Sect. 4.1 is fast, but it cannot guarantee the dual upper bound required by the Farley bound of Lemma 3.1. The exact pricing algorithm is slow but yields the dual upper bound for the pricing problem. The hybrid pricing strategy first calls the pricing heuristic to decide whether the exact pricing algorithm can improve the local dual bound of the master problem.

In fact, the exact algorithm is required only under a particular condition. The following proposition gives the condition.

Proposition 4.1. *Let v_{heur} be the solution value of the pricing heuristic, let v_{RMLP} be the optimum of RMLP (7), and let v_{ld} be the current local dual bound for the master problem. If $\frac{v_{\mathrm{RMLP}}}{v_{heur}} \leq v_{\mathrm{ld}}$, 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 $\frac{v_{\mathrm{RMLP}}}{v_{popt}} \leq \frac{v_{\mathrm{RMLP}}}{v_{heur}} \leq v_{\mathrm{ld}}$. However, v_{popt} is the smallest pricing dual bound v_{price}, so $\frac{v_{\mathrm{RMLP}}}{v_{popt}}$ is the greatest Farley bound according to Lemma 3.1. Therefore even if the pricing algorithm is solved to optimality, we cannot obtain a better bound than v_{ld}. □

If the condition $\frac{v_{\mathrm{RMLP}}}{v_{heur}} \leq v_{\mathrm{ld}}$ holds, one can get rid of the exact pricing algorithm, and use the solution from the fixing-greedy heuristic in Sect. 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 pricing heuristic with the objective coefficients π' ; ▷ run heuristic first
4 let x, v_{heur} be the heuristic solution and its value;
5 **if** $\frac{v_{\mathrm{RMLP}}}{v_{heur}} \leq v_{\mathrm{ld}}$ *and* $1 - \sum_{i \in \mathcal{N}'} \pi'_i \bar{x}_i < 0$ **then**
6 | $x^* \leftarrow x$; ▷ heuristic solution
7 **else**
8 | call the exact pricing Algorithm 1 ; ▷ exact pricing
9 | let $\tilde{x}, v_{\mathrm{price}}$ be the primal solution and the dual bound;
10 | $x^* \leftarrow \tilde{x}$;
11 | $v_{\mathrm{ld}} = \max\{v_{\mathrm{ld}}, \frac{v_{\mathrm{RMLP}}}{v_{\mathrm{price}}}\}$; ▷ update the local dual bound
12 **end**

The heuristic algorithm is called first in Line 3. If $\frac{v_{\mathrm{RMLP}}}{v_{heur}} \leq v_{\mathrm{ld}}$, 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 made to test the effectiveness of our branch-and-price algorithms for 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. We also provide a bash file to reproduce the experiments on Linux systems.

5.1. Benchmarks

We produce benchmarks as described in [4]. The authors test their approximation algorithms on benchmarks from real cloud data centers of `Google`, which are not accessible due to confidentiality [1].

They also describe data generation methods by considering a variety of uncertainty models, and these methods have a probabilistic interpretation: parameters of SMBP instances are derived from parameters of uncertainty models. For different risk levels α, they propose three data generation methods (cases) to construct the data a, b, σ in SMBP (4), i.e., the Gaussian case, the Hoeffding inequality case, and the distributionally robust approximation case.

We describe the generation methods next. In summary, we first determine overall parameters such as capacity and item numbers, then we generate distributions, and finally cast parameters of distributions into parameters of items.

The overall parameters of instances are set as follows. 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 `CloudLarge`. There are three generation methods and six risk levels.

The distributions of instances are set as follows. We call the distribution of μ_i the *target distribution* for item i. We assume that every μ_i follows the

[1] We, therefore, create new instances using the same generation method.

same target distribution. This target distribution is unknown in [4] except for its quantiles in Table 1.

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

1. sample μ_i ($i \in \mathcal{N}$) according to Table 1;
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 1: 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 1. A histogram consists of intervals divided from the entire range $[0, 72]$, and each interval has endpoints of two consecutive quantiles of Table 1. The histogram gives a discrete non-parametric estimation of the target distribution. We apply a two-stage sampling to obtain a nominal item size μ_i ($i \in \mathcal{N}$) sampled from a continuous distribution. 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 $\overline{A}$, 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 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 $i \in \mathcal{N}$, let $a_i = \mu_i' \mu_i$ and $b_i = (\sigma_i' \mu_i)^2$.

For the Hoeffding's inequality case:

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

For the distributionally robust approximation case:

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

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.

We generate six instances with different random seeds for each combination of generation methods and risk levels. As a result, we have $108 = 6 \times 6 \times 3$ instances in a benchmark.

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 server with Intel Xeon W-2245 CPU @ 3.90GHz, 126GB main memory, and Ubuntu 18.04 system. We use `SCIP` 8.0.1 [60] as a branch-and-price (B&P) framework to solve the set cover formulation (5). We use ILOG `CPLEX` 22.1 as:

- an LP solver to solve the RMLP (7);
- a BSOCP solver to solve the BSOCP formulations of SMBP (4) and the submodular knapsack problem with conflicts (10);
- a MILP solver used by the PWL-B&C Algorithm 1;

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

Solver implementation We implement four solvers for SMBP according to the proposed techniques in this paper. Four of them are branch-and-price solvers. These solvers are as follows:

1. BSOCP-BC: a solver using CPLEX's B&C algorithm to solve the compact BSOCP formulation of SMBP.
2. DW-BC: a B&P solver for solving the set cover formulation (5), which uses CPLEX's B&C algorithm to solve the BSOCP formulation (10) of the pricing problem.
3. DW-PWL: a B&P solver for solving the set cover formulation (5), which uses the PWL-B&C algorithm to solve the combined formulation (16) of the pricing problem.
4. DW-Hybrid: DW-PWL enhanced with the hybrid pricing strategy in Algorithm 2.

We use the approximation algorithm from [4] to find an initial feasible solution that serves as a warm start for all solvers. All B&P solvers deploy the column selection heuristic in Sect. 3.3, and all exact pricing solvers add the solution exclusion constraint (9) to pricing problems. 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 primal 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.

Performance metrics and statistical tests In order to evaluate the solver performance in different instances, we compute shifted geometric means (SGMs) (see [55]) of performance metrics as aggregated statistics. Compared to arithmetic means, SMGs avoid the over-representation of biased outlier points. 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 min-utilization algorithm, which is communicated to all solvers as a warm start. The closed primal bound is defined as:

$$\delta_p := \frac{v^a - \overline{v}}{\max(\overline{v} - \underline{v}^*, 1e^{-6})} \times 100,$$

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

We report 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 the 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 BSOCP-BC, while metric (6) is 100% for the DW-PWL and DW-BC.

5.3. Comparative analysis of results

The main computational results are summarized in Table 2, for detailed results, we refer to Appendix D. For each benchmark, we report the SGM

Benchmarks	Solvers	Problem statistics						Pricing statistics			
		t	δ_d%	δ_p%	#N	#S	#I	#C	E%	τ%	t_p%
`CloudSmall` ($\|\mathcal{N}\| = 100$)	BSOCP-BC	1452	15.8	0.0	26601	18	0	-	-	-	-
	DW-BC	2129	11.4	0.9	21	20	17	1373	100	3.56	99
	DW-PWL	633	2.4	2.7	66	61	32	1869	100	0.01	99
	DW-Hybrid	330	2.0	3.4	127	65	36	3485	18	0.01	96
`CloudMedium` ($\|\mathcal{N}\| = 400$)	BSOCP-BC	3600	100.0	0.0	0	0	0	-	-	-	-
	DW-BC	3600	39.0	0.1	2	0	4	861	100	0.39	98
	DW-PWL	3600	17.2	0.4	1	0	10	3372	100	0.01	91
	DW-Hybrid	3600	11.8	0.6	12	0	15	6879	9	0.04	73
`CloudLarge` ($\|\mathcal{N}\| = 1000$)	BSOCP-BC	3600	100.0	0.0	0	0	0	-	-	-	-
	DW-BC	3600	59.6	0.0	2	0	0	741	100	0.04	89
	DW-PWL	3600	43.1	0.2	1	0	6	2105	100	0.01	63
	DW-Hybrid	3600	34.2	0.4	1	0	11	4257	4	0.01	8

Table 2: Aggregated statistics of the main computational results

statistics of the performance metrics, the number of instances solved (denoted by #S), and the number of instances with improved primal bounds (denoted by #I). We also report a computational test of adaptive selection of break points in Appendix C. Next, we analyze the main computational results by comparing the solvers.

We first compare the compact BSOCP formulation of SMBP (4) with the set cover formulation (5). So, we evaluate the performance of BSOCP-BC and DW-BC. For all the benchmarks, DW-BC achieves smaller dual gaps than BSOCP-BC. For small instances, DW-BC also explores a smaller number of nodes of the search tree. These observations agree with Prop. 2.1 that the continuous relaxation of the set covering formulation is stronger than the continuous relaxation of the compact formulation. The number of nonlinear integer constraints in the compact formulation increases with the number of bins. For medium instances, BSOCP-BC cannot even finish the root node computation of the compact formulation. However, DW-BC can prove a dual gap or improve primal solutions by solving the set cover formulation. Although the two formulations are insufficient to tackle medium or large instances, the compact formulation is better overall than the set cover formulation. Then, we will solely examine algorithms that tackle the set cover formulation in the following.

We next evaluate our core innovation to solve the pricing subproblems: the PWL relaxation and its associated combined formulation (18). So, we compare DW-BC with DW-PWL. DW-BC just calls `CPLEX` to solve the

BSCOP formulation (10) of pricing subproblems, while DW-PWL uses a tailored branch-and-cut algorithm to solve the combined formulation (18). Looking at the problem statistics for all the benchmarks, we find that DW-PWL significantly reduces the master problem' dual gap than DW-BC. Especially for small instances, DW-PWL achieves nearly five times improvement to DW-BC. More details can be found in pricing statistics. DW-PWL can solve pricing subproblems to optimality (pricing gap on average is 0.01%) in a short time and thus produce much more columns than DW-BC. Especially for large instances, we find that combined formulation (18) is still solvable. The overall quality of the combined formulation for submodular knapsack outperforms that of the BSOCP formulation (10).

We examine the hybrid pricing strategy, which replaces the computationally expensive exact pricing with computationally cheap heuristic pricing when the exact pricing is not in need. So, we compare DW-PWL with DW-Hybrid. Looking at the problem statistics, the hybrid pricing strategy achieves smaller dual gaps, especially for large instances; it also saves computational time for small instances. Looking at the pricing statistics, the hybrid pricing strategy can generate twice the number of columns than the exact pricing, for all the instances. As a byproduct, with more columns, `SCIP` can find more improved primal solutions. We find that the hybrid pricing strategy gives rise to consistent improvement.

We look at the column selection heuristic. So, we compare DW-Hybrid with DW-Hybrid*. The column selection heuristic can find more improved solutions.

Finally, we summarize our computational results. The set cover formulation is better than the compact formulation regarding scalability, although both formulations are unsolvable for medium and large instances. Our techniques can improve the column generation procedure for the set cover formulation. Regarding pricing subproblems, a dense BSOCP constraint might be reformulated as a submodular knapsack constraint, so the good performance of PWL relaxations suggests that PWL relaxations can provide strong MILP relaxations for dense BSOCP constraints. This finding can also help solve other BSOCP problems. The hybrid pricing strategy uses a hint from the Farley bound, so it reduces computational time and is applicable for other column generation problems. As for benchmarks, `CloudSmall` is a suitable testbed for comparing solvers, `CloudMedium` is suitable for testing the pricing algorithms, and `CloudLarge` is still too big to handle.

6. Conclusion

We develop a PWL-B&C algorithm for solving pricing submodular knapsack problems. The PWL-B&C algorithm is more efficient than the conventional LP-B&C algorithm implemented in `CPLEX` for the pricing submodular knapsack problems. The PWL-B&C algorithm can also be extended to solve the multiple submodular knapsack problems. In general MINLP problems, if a nonlinear constraint can be reformulated into a linear part and a univariate concave part, then the univariate concave part can be convexified by PWL relaxation.

Our hybrid pricing strategy applies to the column generation procedure, where the master problems are in set cover formulations, as long as there are fast pricing heuristics. This pricing strategy is helpful for large instances. As a future study, we can apply this strategy to solve the DW decomposition of the capacitated vehicle routing problem, for which the pricing problem is complex.

The primary efforts of this paper are solving pricing submodular knapsack subproblems with conflicts via PWL relaxations and speeding up column generation via a hybrid pricing strategy. There is still much room for improvement in future studies. Since the submodular knapsack with conflicts is solved multiple times with different parameters, the information of previous column generation iterations can be leveraged statistically to reduce the search space of pricing subproblems.

On the other hand, commonly known techniques for branch-and-price algorithms are generally helpful. Combining our techniques with other advanced elements from general-purpose framework [21] could be also useful. For example, we can use stabilization techniques to speed up the convergence of the column generation or use cutting planes to tighten the relaxation of the master problem.

ACKNOWLEDGMENT

The authors would like to thank Leo Liberti and Sandra Ulrich Ngueveu for discussing this paper with the authors.

Appendix A. Proof of Prop. 2.1

Proof. Let

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

be the feasible set of the $j-$th constraint in the BSOCP formulation of (4). 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

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

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 vectors and $(p, 1)$ $(p \in \mathcal{P})$. Therefore, its convex hull is

$$\begin{aligned}&\mathrm{conv}(F_j) = \\ &\{(v_{1j}, \cdots, v_{nj}, y_j) \in [0,1]^{n+1} : \exists \lambda_p \in [0,1]^{\mathcal{P}} \wedge \sum_{p \in \mathcal{P}} \lambda_p = y_j \wedge v = \sum_{p \in \mathcal{P}} d_p \lambda_p\}.\end{aligned}$$

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

The optimum of the continuous relaxation of the BSOCP formulation is $\min_{(v,y) \in \overline{F}, v \text{ satisfies (4c)}} \sum_{j \in \mathcal{M}} y_j$.

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 $\min_{(v,y) \in \prod_{j \in \mathcal{M}} \mathrm{conv}(F_j), v \text{ satisfies (4c)}} \sum_{j \in \mathcal{M}} y_j$. Since $\prod_{j \in \mathcal{M}} \mathrm{conv}(F_j) \subset \overline{F}$, the result follows. □

Appendix B. Proof of Thm. 4.2

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:

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

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 (17) 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. Non-equidistant breakpoints

According to Thm. 4.2 in Sect. 4.4, the optimal breakpoints under the ℓ_∞ error form an equidistant partition of $[\underline{w}, \overline{w}]$. In this section, we investigate whether adaptive non-equidistant breakpoints can improve the DW-PWL. There are many possibilities for non-equidistant breakpoints, and we propose a regression approach using the previous pricing information.

We recall that the DW-PWL solver adds a lazy cut at each infeasible solution $\hat{x}$. Let $\hat{w} := \sum_{i \in \mathcal{N}'} a_i' \hat{x}_i$ be the corresponding value of variable w, and we call it an infeasible w-value. For an objective coefficient vector c', let $[w_\ell(c'), w_u(c')]$ be the range of the set of infeasible w-values, which are recorded during the PWL-B&C algorithm for every pricing problem. Our intuition is that for a new pricing problem with an objective coefficient vector c, one may reduce the search space by concentrating breakpoints to the range $[w_\ell(c), w_u(c)]$, because this refines the PWL relaxation in that region.

Usually $[w_\ell(c), w_u(c)]$ is unknown, so one can only use a predication range $[w'_\ell(c), w'_u(c)]$. Given the fixed number of breakpoints of $\mathcal{B}$, with this limited resource, we use the $k$nn regression approach to learn $[w'_\ell(c), w'_u(c)]$ and concentrate a subset of of $\mathcal{B}$ to $[w'_\ell(c), w'_u(c)]$.

The $k$nn regression is as follows. Let T be the number of pricing iterations, and we use a list $\{[w_\ell(c^t), w_u(c^t)]\}_{1\leq t\leq T}$ to record the set of intervals, where $w_\ell(c^t), w_u(c^t)$ are the lower and upper bounds of the set of infeasible w-values in the t-th pricing problem. For the new objective coefficient vector c, we sort the list in an increasing order w.r.t. the ℓ_2-norm distances between $\{c^t\}_{1\leq t\leq T}$ to c. The predicted range $[w'_\ell(c), w'_u(c)]$ is as follows:

$$w'_\ell(c) = \sum_{1\leq t\leq k} w_\ell(c^t)/k, w'_u(c) = \sum_{1\leq t\leq k} w_u(c^t)/k.$$

Recall that there are in total h breakpoints in the range $[\underline{w}, \overline{w}]$. Let $r := (w'_u(c) - w'_\ell(c))/(\overline{w} - \underline{w})$ be the range ratio. Then, given a concentration scale $s > 1$, we put $h * r * s$ number of breakpoints equidistantly in $[w'_\ell(c), w'_u(c)]$, and $h * (1 - r * s)$ number of breakpoints equidistantly in the remaining breakpoint region. This concentration results in a PWL relaxation with a better approximation in $[w'_\ell(c), w'_u(c)]$. Therefore, we hope that the adaptive PWL relaxation could use the previous pricing information.

To understand the performance of the $k$nn regression approach, we have several configurations with combinations of $k \in \{1, 3, 5\}$ and $s \in \{1.5, 2, 2.5\}$. We note that with $k = 1, s = 1$, the configuration is exactly the DW-PWL solver. To test these configurations, we generate two new benchmarks with $|\mathcal{N}| \in \{500, 900\}$, each containing 36 instances. The aggregated computational results are presented in Table C.3, which displays SGMs of the relative dual gap of master problems, and the number of generated columns.

The non-equidistant breakpoints do not lead to an improvement of the algorithm. In most cases, $k$nn regression approach is even worse than the equidistant breakpoint approach. Therefore, finding good breakpoints is a complex task.

Appendix D. Detailed results

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

	$k=1$				$k=3$			$k=5$		
	$s=1$	$s=1.5$	$s=1.5$	$s=2.5$	$s=1.5$	$s=2$	$s=2.5$	$s=1.5$	$s=2$	$s=2.5$
$\delta_d\%$	35.29	35.58	35.66	35.36	35.28	36.28	35.29	35.39	34.95	35.29
$\#C$	2134.12	2120.03	2097.46	2123.38	2118.57	2091.97	2148.48	2111.01	2155.95	2134.12

Table C.3: Master and pricing problem statistics of different configurations

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	BSOCP-BC						DW-BC						DW-PWL						DW-Hybrid						DW-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 D.4: Master problem statistics of `CloudSmall` with 108 instances ($|\mathcal{N}| = 100$)

Case	α	BSOCP-BC						DW-BC						DW-PWL						DW-Hybrid						DW-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 D.5: Master problem statistics of `CloudMedium` with 108 instances ($|\mathcal{N}| = 400$)

Case	α	BSOCP-BC						DW-BC						DW-PWL						DW-Hybrid						DW-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 D.6: Master problem statistics of `CloudLarge` with 108 instances ($|\mathcal{N}| = 1000$)

References

[1] Y. Song, J. R. Luedtke, S. Küçükyavuz, Chance-constrained binary packing problems, INFORMS Journal on Computing 26 (4) (2014) 735–747.

[2] A. Charnes, W. W. Cooper, Deterministic equivalents for optimizing and satisficing under chance constraints, Operations Research 11 (1) (1963) 18–39.

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

[4] M. C. Cohen, P. W. Keller, V. Mirrokni, M. Zadimoghaddam, Overcommitment in cloud services: Bin packing with chance constraints, Management Science 65 (7) (2019) 3255–3271.

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

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

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

[8] B. T. Denton, A. J. Miller, H. J. Balasubramanian, T. R. Huschka, Optimal allocation of surgery blocks to operating rooms under uncertainty, Operations Research 58 (4 PART 1) (2010) 802–816.

[9] S. Batun, B. T. Denton, T. R. Huschka, A. J. Schaefer, Operating room pooling and parallel surgery processing under uncertainty, INFORMS Journal on Computing 23 (2) (2011) 220–237.

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

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

Case	α	DW-BC				DW-PWL				DW-Hybrid				DW-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 D.7: Pricing problem statistics of `CloudSmall` with 108 instances ($|\mathcal{N}| = 100$)

Case	α	DW-BC				DW-PWL				DW-Hybrid				DW-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 D.8: Pricing problem statistics of `CloudMedium` with 108 instances ($|\mathcal{N}| = 400$)

Case	α	DW-BC				DW-PWL				DW-Hybrid				DW-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 D.9: Pricing problem statistics of `CloudLarge` with 108 instances ($|\mathcal{N}| = 1000$)

[12] S. Wang, J. Li, S. Mehrotra, Chance-constrained multiple bin packing problem with an application to operating room planning, INFORMS Journal on Computing 33 (4) (2021) 1661–1677.

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

[14] P. C. Gilmore, R. E. Gomory, A linear programming approach to the cutting-stock problem, Operations Research 9 (6) (1961) 849–859.

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

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

[17] A. Frangioni, B. Gendron, A stabilized structured dantzig–wolfe decomposition method, Mathematical Programming 140 (1) (2013) 45–76.

[18] S. Coniglio, F. D'andreagiovanni, F. Furini, A lexicographic pricer for the fractional bin packing problem, Operations Research Letters 47 (6) (2019) 622–628.

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

[20] M. Jepsen, B. Petersen, S. Spoorendonk, D. Pisinger, Subset-row inequalities applied to the vehicle-routing problem with time windows, Operations Research 56 (2) (2008) 497–511.

[21] A. Pessoa, R. Sadykov, E. Uchoa, Solving bin packing problems using vrpsolver models, in: Operations Research Forum, Vol. 2, Springer, 2021, p. 20.

[22] A. Pessoa, R. Sadykov, E. Uchoa, F. Vanderbeck, A generic exact solver for vehicle routing and related problems, Mathematical Programming 183 (2020) 483–523.

[23] A. Pessoa, R. Sadykov, E. Uchoa, F. Vanderbeck, Automation and combination of linear-programming based stabilization techniques in column generation, INFORMS Journal on Computing 30 (2) (2018) 339–360.

[24] D. Pecin, A. Pessoa, M. Poggi, E. Uchoa, Improved branch-cut-and-price for capacitated vehicle routing, Mathematical Programming Computation 9 (2017) 61–100.

[25] S. Gélinas, M. Desrochers, J. Desrosiers, M. M. Solomon, A new branching strategy for time constrained routing problems with application to backhauling, Annals of Operations Research 61 (1995) 91–109.

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

[27] B. Geißler, A. Martin, A. Morsi, L. Schewe, Using piecewise linear functions for solving MINLPs, in: J. Lee, S. Leyffer (Eds.), Mixed Integer Nonlinear Programming, Springer New York, New York, NY, 2012, pp. 287–314.

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

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

[30] A. A. Farley, Note on bounding a class of linear programming problems, including cutting stock problems, Operations Research 38 (5) (1990) 922–923.

[31] P. H. Vance, C. Barnhart, E. L. Johnson, G. L. Nemhauser, Solving binary cutting stock problems by column generation and branch-and-bound, Computational Optimization and Applications 3 (2) (1994) 111–130.

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

[33] V. Blanco, R. Gázquez, D. Ponce, J. Puerto, A branch-and-price approach for the continuous multifacility monotone ordered median problem, European Journal of Operational Research 306 (1) (2023) 105–126.

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

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

[36] O. V. Shylo, O. A. Prokopyev, A. J. Schaefer, Stochastic operating room scheduling for high-volume specialties under block booking, INFORMS Journal on Computing 25 (4) (2013) 682–692.

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

[38] Z. Gu, G. L. Nemhauser, M. W. Savelsbergh, Lifted flow cover inequalities for mixed 0-1 integer programs, Mathematical Programming 85 (3) (1999) 439–467.

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

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

[41] J. Puchinger, G. R. Raidl, U. Pferschy, The multidimensional knapsack problem: Structure and algorithms, INFORMS Journal on Computing 22 (2) (2010) 250–265.

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

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

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

[45] T. Berthold, S. Heinz, S. Vigerske, Extending a CIP framework to solve MIQCPs, in: J. Lee, S. Leyffer (Eds.), Mixed Integer Nonlinear Programming, Springer New York, New York, NY, 2012, pp. 427–444.

[46] C. Coey, M. Lubin, J. P. Vielma, Outer approximation with conic certificates for mixed-integer convex problems, Mathematical Programming Computation 12 (2) (2020) 249–293.

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

[48] C. D'Ambrosio, J. Lee, A. Wächter, An algorithmic framework for MINLP with separable non-convexity, in: J. Lee, S. Leyffer (Eds.), Mixed Integer Nonlinear Programming, Springer New York, New York, NY, 2012, pp. 315–347.

[49] A. Allman, Q. Zhang, Branch-and-price for a class of nonconvex mixed-integer nonlinear programs, Journal of Global Optimization 81 (4) (2021) 861–880.

[50] A. Ceselli, L. Létocart, E. Traversi, Dantzig–Wolfe reformulations for binary quadratic problems, Mathematical Programming Computation (2022) 1–36.

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

[52] L. Xu, S. Haddad Vanier, Branch-and-price for energy optimization in multi-hop wireless sensor networks, Networks 80 (1) (2022) 123–148.

[53] P. Bonami, A. Tramontani, Recent improvement to MISOCP in CPLEX, Tech. rep. (2015).

[54] D. M. Ryan, B. A. Foster, An integer programming approach to scheduling, Computer scheduling of public transport urban passenger vehicle and crew scheduling (1981) 269–280.

[55] T. Achterberg, T. Berthold, T. Koch, K. Wolter, Constraint integer programming: A new approach to integrate CP and MIP, in: L. Perron, M. A. Trick (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 6–20.

[56] M. Lübbecke, C. Puchert, Primal heuristics for branch-and-price algorithms, in: Operations Research Proceedings 2011, Springer, 2012, pp. 65–70.

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

[58] IBM ILOG, Automatic reformulation of special ordered sets, https://www.ibm.com/docs/en/icos/20.1.0?topic=sos-automatic-reformulation-spec [Online; accessed 29-March-2022] (11 2021).

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

[60] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, et al., Enabling research through the scip optimization suite 8.0, ACM Transactions on Mathematical Software (2023).