Expert Systems With Applications 271 (2025) 126534

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Deep reinforcement learning-based column generation for the two-dimensional vector variable-sized packing problem

Zongcheng Zhang [a], Yangsheng Jiang [a], Mujin Gao [b], Yanru Chen [c,*], M.I.M. Wahab [d]

[a] School of Transportation and Logistics, Southwest Jiaotong University, 111, 1st Section, Northern 2nd Ring Road, Chengdu, Sichuan 610031, China
[b] Department of Economics and Management, Taiyuan Institute of Technology, 31, Xinlan Road, Taiyuan, Shanxi 030008, China
[c] School of Economics and Management, Southwest Jiaotong University, 111, 1st Section, Northern 2nd Ring Road, Chengdu, Sichuan 610031, China
[d] Department of Mechanical, Industrial and Mechatronics Engineering, Toronto Metropolitan University, 350 Victoria Street, Toronto M5B2K3, Canada

ARTICLE INFO

Keywords:
Variable-sized packing problem
Time window
Conflict graph
Column generation
Deep reinforcement learning

ABSTRACT

This study introduces a new variant of classical packing problems, the two-dimensional vector variable-sized packing problem with conflicts and time windows (2DVVSPPCTW). This problem involves packing items with varying weights, volumes, and time windows into bins of different sizes and costs. Some items are in conflict and cannot be packed together. The objective is to minimize the overall cost of packing the items into the bins. To enhance the tractability of the 2DVVSPPCTW, a deep reinforcement learning-based column generation (DRLCG) is developed. This algorithm provides a unique way to perform column generation, enabling the quick solution of various NP-hard packing problem variants. Unlike traditional column generation algorithms that typically use a single method for all instances and pricing problems—often resulting in time inefficiencies—this algorithm introduces two specialized methods tailored for different instances and pricing problems. The first method is a community-detection-based actor-critic algorithm, a new deep reinforcement learning technique that efficiently generates a significant number of promising columns through batch processing. The second method is the decomposed label-setting algorithm, which decomposes the label expansion process into multiple subprocesses to quickly identify the search space containing columns with a negative reduced cost. An adaptive selection strategy is incorporated to determine the most appropriate method for each specific instance and pricing problem. Extensive experiments are conducted to evaluate the performance of the DRLCG algorithm. The results show that the DRLCG algorithm significantly outperforms the GUROBI solver and four existing approaches, achieving 4%-10% lower costs in a shorter computation time.

1. Introduction

In the logistics industry, there is a common need to efficiently pack items that vary in weight, volume, and have specific time windows into bins of different sizes and costs. Additionally, some items may conflict and cannot be packed together. This challenge is characterized as the two-dimensional vector variable-sized packing problem with conflicts and time windows (2DVVSPPCTW). The 2DVVSPPCTW is a novel variant of bin packing problems (BPPs), which are well-known for their practical applications in various real-world scenarios. These include task allocation in cloud computing, load planning for diverse vehicle fleets, and file storage management (Ekici, 2022), as well as engineering tasks (Wang et al., 2022), and database storage in the context of the colored bin packing problem (Borges et al., 2024b).

The 2DVVSPPCTW is motivated by a global shipping company, which faces the complex challenge of arranging a diverse range of products such as electronics, fresh produce, and consumer goods into different-sized shipping containers. Each item has a specific time window for delivery, adding another layer of complexity to the packing process. Electronics like smartphones and laptops, for instance, must be separated from magnetic materials to prevent data corruption. Meanwhile, perishable items like fruits and vegetables require expedited handling and delivery due to their limited shelf life, so they must be packed in the bins with other items that have similar urgent delivery schedules. This requirement that items packed in the same bin must adhere to a common delivery time frame further complicates the

* Corresponding author at: School of Economics and Management, Southwest Jiaotong University, Chengdu, Sichuan 610031, China.
E-mail addresses: zongcheng.zhang@my.swjtu.edu.cn (Z. Zhang), jiangyangsheng@swjtu.cn (Y. Jiang), gaomujin@tit.edu.cn (M. Gao), chenyanru@swjtu.cn (Y. Chen), wahab@torontomu.ca (M.I.M. Wahab).

https://doi.org/10.1016/j.eswa.2025.126534
Received 11 May 2024; Received in revised form 5 December 2024; Accepted 10 January 2025
Available online 20 January 2025
0957-4174/© 2025 Elsevier Ltd. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

packing process. The main goal for this global shipping company is to minimize the total cost of the container and ensure adherence to the delivery schedules of these incompatible and time-sensitive items. Based on the real logistics scenario, we address the 2DVVSPPCTW. To the best of our knowledge, no research addresses the 2DVVSPPCTW.

The proposed 2DVVSPPCTW is an extension of several existing problems: the two-dimensional vector packing problem (2DVPP) (e.g., Wang et al., 2022), the variable-sized bin packing problem (VSBPP) (e.g., Ekici, 2023; Fleszar, 2023), and the bin packing problem with conflicts (BPPC) (e.g., Ekici, 2021; Ekici, 2022). Since these problems are well-known for being classified as NP-hard problems, the 2DVVSPPCTW can also be classified as NP-hard. Column generation (CG) is the prominent approach for these problems (Liu et al., 2021; Haouari & Mhiri, 2024). In the CG, the pricing problem is the most time-consuming, thus, we emphasize the algorithm for the pricing problem.

When solving the pricing problem in the CG for the 2DVPP, the dynamic programming (DP) is usually used (Wei, Lai, et al., 2020). In this paper, the time window constraint is considered based on the 2DVPP, which needs to be solved separately according to different time partitions when DP is employed (Liu et al., 2021), and the search space under different time partitions overlaps, which affects the efficiency of DP. Besides, the BPPC contains conflict constraints, unlike constraints of weight and volume that can be quantified, conflict constraints are usually qualitative, denoted as “can” or “cannot” be packed together, therefore, extra effort is required to deal with conflicts in the DP recursion functions when solving the pricing problem, it significantly increases the computational burden. Thus, a label-setting algorithm or heuristic label-setting algorithm is usually employed when dealing with complex constraints (Baldacci et al., 2024). However, the VSBPP contains multiple pricing problems corresponding to different bin types. The label setting requires a deeper search depth when solving pricing problems with large-sized bins, which increases the computational complexity, and even results in difficulty to solve the problem in a reasonable time.

In response to the challenges outlined above, this study has the following objectives: (1) To examine the intricate characteristics of the proposed 2DVVSPPCTW and to formulate it using integer programming modeling techniques; (2) To develop a specialized column generation algorithm that can efficiently solve such combinatorial optimization problem as the 2DVVSPPCTW; and (3) To explore the potential of deep reinforcement learning techniques to enhance the efficiency of solutions for NP-hard problems.

This study provides the following contributions:

a) Motivated by observations from real-world applications, we introduce a novel variant of the packing problem, the 2DVVSPPCTW. To our knowledge, this is the first study to simultaneously address a range of real-world constraints in the context of two-dimensional vector packing, including item weight and volume, variable-sized bins, time windows, and conflict constraints. We formulate the 2DVVSPPCTW as an integer programming model.
b) To address the 2DVVSPPCTW, we develop the deep reinforcement learning-based column generation (DRLCG) algorithm. This algorithm offers a unique perspective for designing the column generation process, enhanced by deep reinforcement learning, to quickly solve the 2DVVSPPCTW. It is also the first to be applied to Bin Packing Problems. Specifically, traditional column generation algorithms typically use a single method for all instances and pricing problems, which can be time-consuming due to the complexity of the 2DVVSPPCTW. In contrast, the DRLCG algorithm identifies the unique characteristics of instances and pricing problems, introducing two specialized methods tailored to different instances and pricing problems. The first approach is the community-detection-based actor-critic (CDAC) algorithm, which divides all items into multiple communities with low conflict levels, facilitating the generation of a substantial number of promising columns via batch processing. The second approach is the decomposed label-setting algorithm, which breaks down the label expansion into multiple subtasks to rapidly identify the search space containing promising columns. It is worth specifying that the proposed algorithm can also be applied to facilitate the column generation process in exact algorithms like the branch-and-price algorithm. From a methodological point of view, this study develops a specialized column generation algorithm which extends the work on solving computationally intensive pricing problems. The proposed CDAC and decomposed label-setting algorithm are rooted in the concept of decomposition, making them more effective in managing large-scale instances that have a high computational burden. Additionally, this study investigates the potential of deep reinforcement learning techniques to significantly accelerate solutions to NP-hard combinatorial optimization problems, such as packing problems and vehicle routing problems.
c) Extensive comparative experiments are conducted to examine the performance of the DRLCG algorithm, CDAC algorithm, decomposed label-setting algorithm, and different accelerating strategies. The results reveal that these algorithms demonstrate significant advantages, regarding solution quality and computation time.

The structure of this paper is organized as follows. Section 2 comprehensively reviews the existing literature on the VSBPP, 2DVPP, BPPC, and their related variations. Section 3 introduces a mathematical formulation of the 2DVVSPPCTW, which is further transformed into a master problem and pricing problems. The DRLCG algorithm is proposed in Section 4. The performance of the DRLCG algorithm is then investigated through comparative experiments, discussed in Section 5. Finally, Section 6 concludes the paper and suggests potential directions for future research.

2. Literature review

The 2DVVSPPCTW is a generalization of the bin packing problem (BPP), which has emerged as a rapidly evolving area of research within combinatorial optimization (Mezghani et al., 2023). In recent years, numerous studies have explored various variants of BPP, developed mathematical models, and proposed different methodologies to address these problems. The applications of BPP are diverse, extending to fields such as scheduling, engineering, and computer science (e.g., Borges et al., 2024a).

Solving Bin Packing Problems (BPPs) and their variants presents significant challenges, as they are classified as NP-hard problems. To address these challenges, various solution techniques have been developed, including approximation methods, constructive heuristics, metaheuristics, and exact algorithms such as branch-and-price and branch-and-bound. Additionally, machine learning methods have emerged as effective tools in this domain.

In recent years, expert system-based solution techniques have gained attention for their ability to capture and retain the knowledge of human experts, thereby preserving valuable expertise in decision-making processes. Constructive algorithms, in particular, leverage expert experience to formulate a series of heuristic rules aimed at optimizing packing order and space utilization. For instance, Dhahbi et al. (2021) introduced a worst-fit-based load balancing algorithm for the bin stretching problem, which selects the bin with the smallest available space for packing items while employing heuristic rules to minimize empty space in advance. Similarly, Schepler et al. (2022) utilized expert knowledge to sort items by non-increasing weights, implementing the first-fit decreasing heuristic as an initial solution for robust bin-packing problems. Aydın et al. (2020) developed a constructive look-ahead heuristic for the temporal bin packing problem, creating expert-system-based packing rules that evaluate the current system state and predict future request arrivals. Moreover, machine learning methods harness extensive historical data and scenarios, akin to an expert knowledge pool, to learn packing rules and generate optimal packing patterns. Cheng et al. (2024)

proposed a deep reinforcement learning approach for the class-constrained bin packing problem, utilizing Graph Convolutional Networks (GCN) to extract node features of items and their interrelationships, ultimately outputting the probability of two items being packed together.

As the 2DVVSPPCTW can be seen as an extension of packing problems such as the VSBPP, 2DVPP, and BPPC, the primary emphasis of the literature review in this study is on these three specific problems.

2.1. The VSBPP

In recent years, VSBPP, as a variant of the BPP, has attracted significant attention. Liu et al. (2021) develop three heuristic algorithms to tackle the variable-sized bin packing problem with time windows (VSBPPTW): an improved best-fit heuristic, an iterative local search algorithm, and a primal heuristic. Fleszar (2023) introduces a novel mixed-integer linear programming model and devised four heuristics for VSBPPTW, namely a best-fit heuristic, an improved variant of the best-fit heuristic, a two-way best-fit heuristic, and a least-compatible-first heuristic. In Ekici (2023), the variable-sized bin packing problem with conflicts (VSBPPC) is considered, and a large neighborhood search (LNS) algorithm and two new lower bounds are developed for this problem.

2.2. The 2DVPP

The 2DVPP, which also extends the BPP, focuses on efficiently packing a set of items into a minimal number of bins while satisfying weight and volume constraints. In the context of the apparel industry, Wang et al. (2022) introduce a specific instance of the 2DVPP. In this instance, the weight related to volume of items is described as volumetric weight. Most major logistics companies calculate the charge of products using volumetric weight. Besides, Pessoa et al. (2020) and Pessoa et al. (2021) develop a generic VRPSolver based on the branch-and-price-and-cut algorithm for various versions of the BPP. Similarly, Wei, Lai, et al. (2020) introduce a branch-and-price algorithm with rounded capacity inequalities for the 2DVPP. Additionally, some studies introduce cost functions into the 2DVPP.

2.3. The BPPC

The BPPC is another variation of the BPP that considers incompatibilities between certain items. In terms of algorithms, various heuristics and exact methods have been employed to solve the BPPC within a reasonable computational time (Wei, Luo, et al., 2020). Wei, Luo, et al. (2020) focus on the BPP and BPPC developing the branch-and-price-and-cut algorithm and introducing the sub-set row inequality to enhance the lower bound. Ekici (2021) explores the BPPC with item fragmentation, dividing items into two or more parts during the packing process. For this problem, a sequential maximum degree packing heuristic is developed based on the observation of oversized items.

2.4. Machine learning for optimization methods

In recent years, researchers have a growing interest in utilizing machine learning to enhance optimization methods. Jia and Shen (2021) propose a machine-learning-enhanced benders decomposition algorithm to select optimality cuts for solving two-stage stochastic programs. Morabit et al. (2021) introduce a graph neural network technique to accelerate the CG algorithm by identifying preferred columns based on degeneracy. To the best of our knowledge, no existing literature investigates the 2DVVSPPCTW problem, and current methods involving machine learning for accelerating column generation are limited. Quesnel et al. (2022) develop a deep neural network in the branch-and-price algorithm for the personalized crew rostering problem. The deep learning technique is used to select pairings that are likely to be contained in the optimal solution. Huang et al. (2023) introduce a machine learning-based column-and-row generation approach for the aircraft and cargo recovery problem. The machine learning reduces the number of flight delay decisions, and saves the process time of the algorithm. Santos et al. (2023) develop a software to monitor environmental conditions using sensors and algorithms aiming at animal production, which proved to be efficient and sustainable.

3. Problem formulation

3.1. Problem description

The 2DVVSPPCTW can be characterized as follows. Given a set of items I with a total of n items, each item i has a weight, w_i, volume, v_i, and time window, $[e_i, l_i]$, where $i \in I$. The conflict between items can be represented using a conflict graph $G = (I, E)$, where an edge $(i, q) \in E$ if items i and q are in conflict. In addition, there is a set of bin types J with a total of m different types. Each bin type $j \in J$ has a weight capacity, W_j, volume capacity, V_j, and cost, c_j. The goal is to pack all the items into multiple bins, minimizing the total cost of the bin types. There is no limitation on the number of bins of any particular type. The items are packed into bins without exceeding the weight and volume capacities of the bins, and items in conflict are placed in different bins. Moreover, the time windows of items packed into the same bin must overlap for a shared delivery time.

To illustrate the 2DVVSPPCTW, we present an instance comprising two distinct types of bins with $W_1 = 100$, V_1=100, c_1=100, W_2=150, V_2=150, and $c_2 = 150$. This scenario involves seven items, and the characteristics of these items are presented in Table 1.

The time windows for the items are visually represented in Fig. 1(a). The optimal solution to the 2DVVSPPCTW involves packing the items into three bins, as demonstrated in Fig. 1(b). Specifically, two bins of type 1 and one bin of type 2 are utilized. It is important to note that the time windows of items allocated to the same bin must overlap, e.g., $[e_1, l_1] \cap [e_4, l_4] \neq \varnothing$ for bin 1. Additionally, items that are in conflict must be packed in different bins.

3.2. Mathematical model

To model the 2DVVSPPCTW, the sets, parameters, and decision variables are defined in Table 2 as follows.

An integer programming model for the 2DVVSPPCTW is developed as follows.

$$\min z = \sum_{k \in K} \sum_{j \in J} c_j y_{kj}, \tag{1}$$

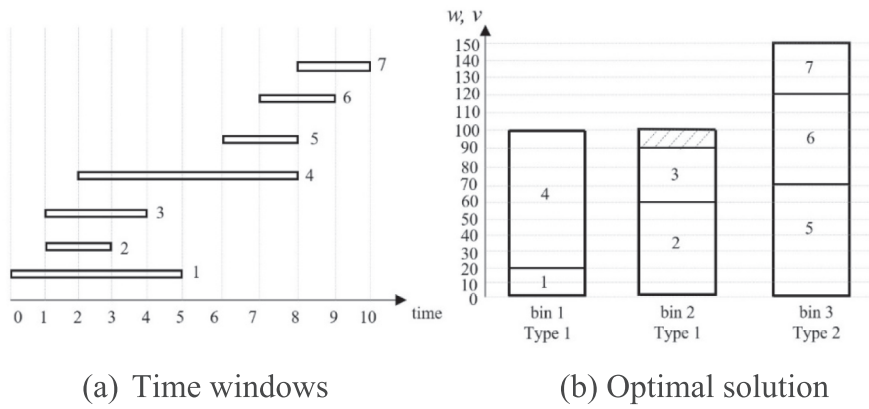
$$s.t. \sum_{k \in K} x_{ik} = 1, \forall i \in I, \tag{2}$$

$$\sum_{j \in J} y_{kj} \leq 1, \forall k \in K, \tag{3}$$

$$\sum_{i \in I} w_i x_{ik} \leq \sum_{j \in J} W_j y_{kj}, \forall k \in K, \tag{4}$$

Table 1
An instance of the 2DVVSPPCTW.

Item	w_i	v_i	$[e_i, l_i]$	Conflicting items
1	20	20	[0, 5]	
2	60	60	[1, 3]	
3	30	30	[1, 4]	
4	80	80	[2, 8]	5, 6, 7
5	70	70	[6, 8]	4
6	50	50	[7, 9]	4
7	30	30	[8, 10]	4



(a) Time windows (b) Optimal solution

Fig. 1. Example of the 2DVVSPPCTW.

Table 2
Mathematical notation.

Notice	Definition
Sets	
I	Set of items
J	Set of bin types
K	Set of bins
Parameters	
W_j	Weight capacity of bin type j
V_j	Volume capacity of bin type j
c_j	Cost of bin type j
$[e_i, l_i]$	Time windows of item i
w_i	Weight of item i
v_i	Volume of item i
M	A large positive constant
Auxiliary Variables	
t_k	Delivery time of bin k
Decision Variables	
x_{ik}	Binary variable taking 1 if item i is packed to bin k, and 0 otherwise
y_{kj}	Binary variable taking 1 if bin k is from bin type j, and 0 otherwise

$$\sum_{i\in I} v_i x_{ik} \le \sum_{j\in J} V_j y_{kj}, \forall k \in K, \tag{5}$$

$$x_{pk} + x_{qk} \le \sum_{j\in J} y_{kj}, \forall (p,q) \in E, \forall k \in K, \tag{6}$$

$$e_i x_{ik} \le t_k \le l_i x_{ik} + M(1 - x_{ik}), \forall i \in I, k \in K, \tag{7}$$

$$t_k \le t_{k+1}, \forall k \in K, \tag{8}$$

$$\sum_{j\in J} y_{kj} \ge \sum_{j\in J} y_{k+1j} \forall k \in K, \tag{9}$$

$$t_k \ge 0, \forall k \in K, \tag{10}$$

$$x_{ik} \in \{0,1\}, \forall i \in I, k \in K, \tag{11}$$

$$y_{kj} \in \{0,1\}, \forall k \in K, j \in J. \tag{12}$$

The objective function (1) is to minimize the total cost of the bin types. Constraint (2) ensures that each item is packed into exactly one bin. Constraint (3) guarantees that the bin used is from a particular type. Constraints (4) and (5) are the weight and volume limits of bins. Constraints (6) guarantees that items in conflict should not be packed into the same bin. Constraint (7) represents the time window of items. Constraints (8) and (9) are used to reduce symmetry. Constraint (10) denotes that the delivery time of the bin is positive. Constraints (11) and (12) define the binary decision variables.

3.3. Reformulation of the problem

3.3.1. The master problem

For the CG algorithm to solve the problem, the integer programming model is usually reformulated as a master problem and pricing problem with the Danzig-Wolfe decomposition (Dantzig & Wolfe, 1960). The restricted master problem (RMP) is formulated by initializing a few columns since the master problem contains too many columns. A packing pattern characterizes how a set of items can be packed into a bin without violating any constraints (Zhang et al., 2023). Let Ω be the set of all packing patterns. Let p be a subset of items that fit into a selected bin type. Let c_p be the cost of the selected bin type in packing pattern p. Let λ_P be the binary decision variable that is equal to 1 if the packing pattern p is the optimal solution to the problem. Let a_{ip} be the binary coefficient that is equal to 1 if item i is in packing pattern p. The set partitioning formulation of the 2DVVSPPCTW is as follows.

$$\min z = \sum_{p\in\Omega} c_p \lambda_p, \tag{13}$$

$$s.t. \sum_{p\in P} a_{ip}\lambda_p \ge 1, \forall i \in I \tag{14}$$

$$\lambda_p \in \{0,1\}, \forall p \in P. \tag{15}$$

The objective function (13) is to minimize the total cost of selected packing patterns. Constraint (14) makes sure that each item is packed into at least one bin. Constraint (15) defines the binary decision variable.

3.3.2. The pricing problem

The pricing problem determines whether it can find promising packing patterns and add to the RMP to improve its current solution. Let π_i denote the dual variable associated with Eq. (14). The binary decision variable, β_i, is set to 1 if item i is in the optimal solution. The variable t represents the delivery time of the items in the packing pattern. The pricing problem can be formulated as follows.

$$\min \overline{c_p} = c_j - \sum_{i\in I} \pi_i \beta_i, \tag{16}$$

$$\sum_{i\in I} w_i \beta_i \le W_j, \forall j \in J, \tag{17}$$

$$\sum_{i\in I} v_i \beta_i \le V_j, \forall j \in J, \tag{18}$$

$$\beta_p + \beta_q \leq 1, \forall (p,q) \in E, \tag{19}$$

$$e_i\beta_i \leq t \leq l_i\beta_i + M(1-\beta_i), \forall i \in I, \tag{20}$$

$$t \geq 0, \tag{21}$$

$$\beta_i \in \{0,1\}, \forall i \in I. \tag{22}$$

It is worth noting that a pricing problem is formulated to correspond with each bin type, meaning the number of pricing problems is the same as the bin types. One pricing problem includes a certain number of packing patterns.

4. Solution technique

4.1. Framework of DRLCG algorithm

Given the effectiveness exhibited by the CG technique in solving bin packing problems (Alves & De Carvalho, 2007; Puchinger & Raidl, 2007; Cintra et al., 2008; Li et al., 2022), it is reasonable to apply it to address the proposed 2DVVSPPCTW. However, efficiently solving the pricing problem within the CG algorithm poses significant challenges (Wang

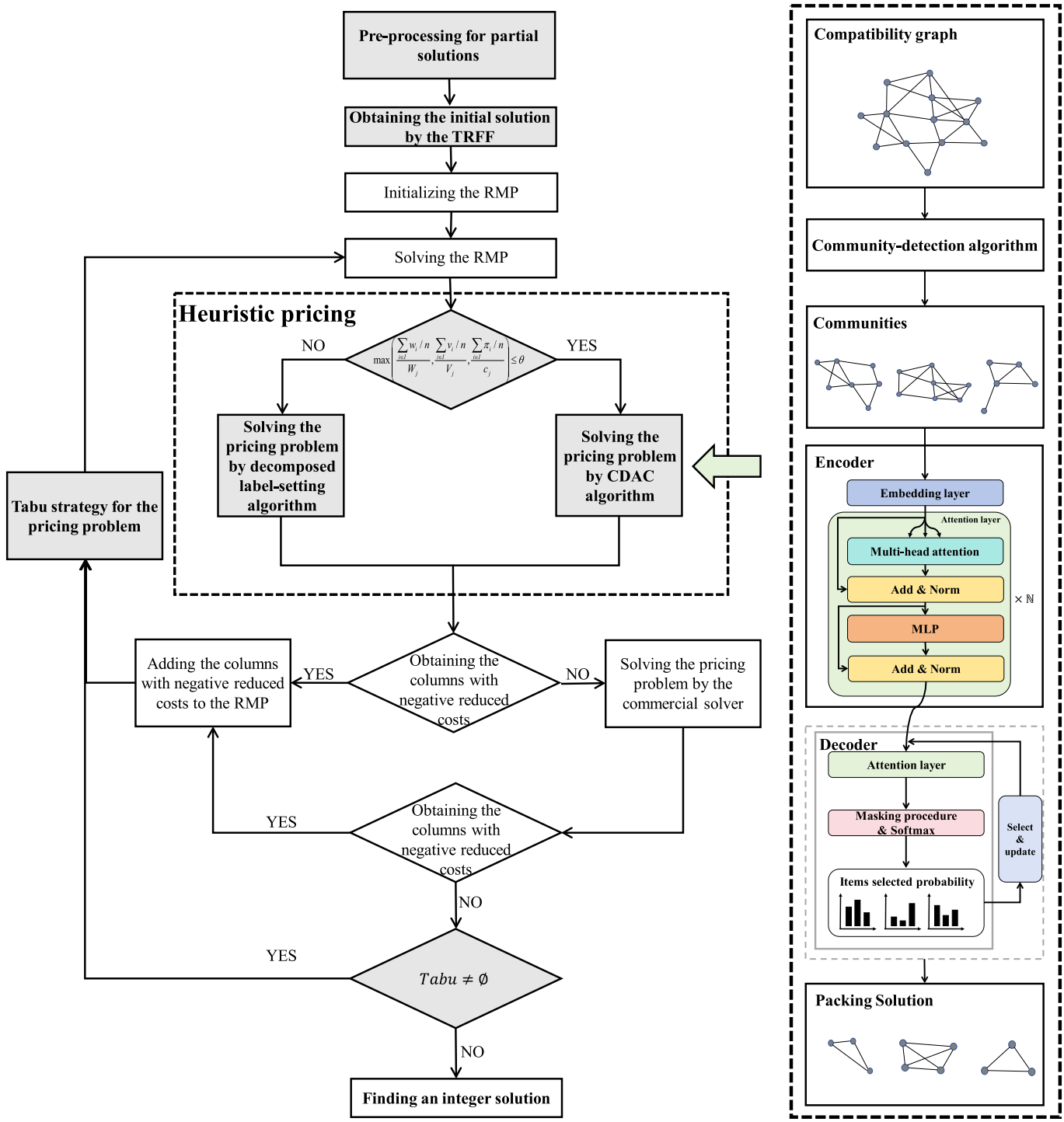


Fig. 2. Outline of the DRLCG.

et al., 2021; Seman et al., 2023; Zhang et al., 2024), particularly for the more computationally demanding 2DVVSPPCTW compared to general bin packing problems. In the context of the 2DVVSPPCTW, which contains multiple pricing problems corresponding to different bin types, these pricing problems differ significantly in structure and complexity. Thus, we propose the DRLCG algorithm, which introduces two specialized methods tailored to the characteristics of the columns for efficient pricing problem solutions. One method is the CDAC algorithm, which divides all items into multiple communities with low conflict levels, and then employs reinforced learning techniques to solve the multiple pricing problems simultaneously; the other is an enhanced version of the basic label-setting algorithm known as the decomposed label-setting algorithm. Within the DRLCG algorithm, the choice between the CDAC algorithm and the decomposed label-setting algorithm is dynamically determined according to the packing patterns associated with columns. When both methods fail to find columns with negative reduced costs, the commercial solver is invoked to solve the pricing problem optimally. Finally, an integer solution method is employed to obtain a near-optimal solution for the proposed 2DVVSPPCTW within a short time.

Additionally, the DRLCG algorithm includes two acceleration strategies: the pre-processing partial solution and the tabu strategy for the pricing problem. The preprocessing strategy can obtain partially optimal packing patterns that significantly reduces the computational burden. Besides, the tabu strategy for the pricing problem improves the efficiency of column generation by reducing the number of the pricing problems to be solved. The framework of DRLCG is shown in Fig. 2 as follows.

4.2. The initial solution

The RMP is generally initialized by initial packing patterns. In this research, a simple heuristic is devised, namely, a first-fit algorithm based on a time range algorithm (TRFF). It is improved by the first-fit algorithm (Beyaz et al., 2015; Meliani et al., 2022). In the TRFF, the items are ordered according to their start time. Then, the items are packed into the bin individually if they can satisfy the constraints of weight, volume, time windows, and conflict. A new bin is used when the items cannot be packed into the bin with available capacity. Regarding the time complexity, let n represent the number of items, and m indicates the number of bins. The time complexity of the TRFF algorithm is O (n*m).

The packing patterns provided by TRFF initialize the RMP. The linear programming (LP) relaxation of the RMP is solved by the GUROBI solver. The dual values are obtained and used for the pricing problem.

4.3. Solving the pricing problem

It is crucial to efficiently solve the pricing problem in the CG (He et al., 2022; Gao et al., 2023). The 2DVVSPPCTW is characterized by varying numbers of items, types of bins, and different pricing problems with varying dual values of items, resulting in columns of varying lengths during iterations of column generation. Thus, it is reasonable to adaptively select an appropriate algorithm for solving the pricing problem, considering the specific characteristics of the instances and the pricing problem itself. Two approaches are proposed to tackle the pricing problem efficiently, the CDAC algorithm is presented in Subsection 4.3.1, and decomposed label-setting is provided in Subsection 4.3.2. How to select the algorithm for solving the pricing problem quickly is introduced in Subsection 4.3.3. The exact pricing method is provided in Subsection 4.3.4.

4.3.1. Heuristic pricing with CDAC

To demonstrate the CDAC algorithm, the following definition is provided.

Definition 1. *Item i is compatible with item h, if*

$$w_i + w_h \le W, \forall i, h \in I, (i,h) \notin E \tag{23}$$

$$v_i + v_h \le V, \forall i, h \in I, (i,h) \notin E \tag{24}$$

$$(l_i - e_h)(l_h - e_i) \ge 0, \forall i, h \in I, (i,h) \notin E \tag{25}$$

According to Definition 1, the compatibility among items can be depicted through a compatibility graph $G = (I, \Xi)$, where an edge $(i,h) \in \Xi$ if item i is compatible with item h. Furthermore, we present the following definition based on Newman (2004) and Fortunato (2010).

Definition 2. *A community is characterized as a set of items where most are compatible with each other.*

With the help of the compatibility graph, we can observe that communities are formed by groups of vertices exhibiting a dense network of edges within the same community. In contrast, the connections between different communities are comparatively sparse.

Conflict constraints in the 2DVVSPPCTW present a challenge for the performance of deep reinforcement learning (DRL). This is because the feature extraction process in DRL requires handling adjacency relationships and adjacency features within a graph. However, conflicts between items differ from simple adjacency relationships. For instance, if items i and j cannot be adjacent, it indicates that item j cannot be visited directly after item i, but item j can still be included in subsequent extensions. This situation only affects the order of adjacencies between the two items. In contrast, if items i and j are in conflict, not only is item j prohibited from being visited immediately after item i, but it also cannot appear in any subsequent expansions. The impact of the conflict constraint extends to the label's future expansions, and this effect accumulates throughout the expansion process, rather than being limited to the next extension. Therefore, treating conflicts merely as a type of adjacency in the network for feature extraction is insufficient to fully capture this complex dependency. Therefore, regarding conflicts as a class of adjacencies in the network for feature extraction is not sufficient to fully capture this complex dependency. Hence, a CDAC algorithm is proposed in this paper. The CDAC algorithm employs a two-step approach, as shown in Fig. 3. Firstly, it divides all items into groups by identifying communities in the compatibility graph using the community-detection algorithm. This grouping strategy ensures that most items within a community are compatible, minimizing the influence of conflicts on the performance of the actor-critic (AC) algorithm. Additionally, it facilitates the identification of feasible packing patterns. Moreover, this grouping approach reduces the number of items that the AC algorithm must handle. Subsequently, the AC algorithm is employed to simultaneously identify columns with negative reduced costs within a large number of communities through batch processing, which is particularly important for accelerating CG. Meanwhile, the AC algorithm does not need to explicitly traverse the travel over the entire state space, thus significantly enhancing the efficiency of pricing problems with large-sized bins.

(1) Community-detection algorithm

The community-detection (CD) algorithm aims to identify communities within a compatibility graph. It is noted that not all items within a community are necessarily compatible with each other. This is because checking compatibility amongst all items can be time-consuming. Furthermore, to prevent the reduction in the feasible solution space that may result from strict division, we allow overlapping communities, where an item can belong to multiple communities simultaneously, as depicted in Fig. 4.

However, the overlapping of communities introduces the possibility of generating duplicate columns, thereby slowing down the optimization process for the problem. Moreover, too many communities increase the computational burden. However, the overlapping of communities

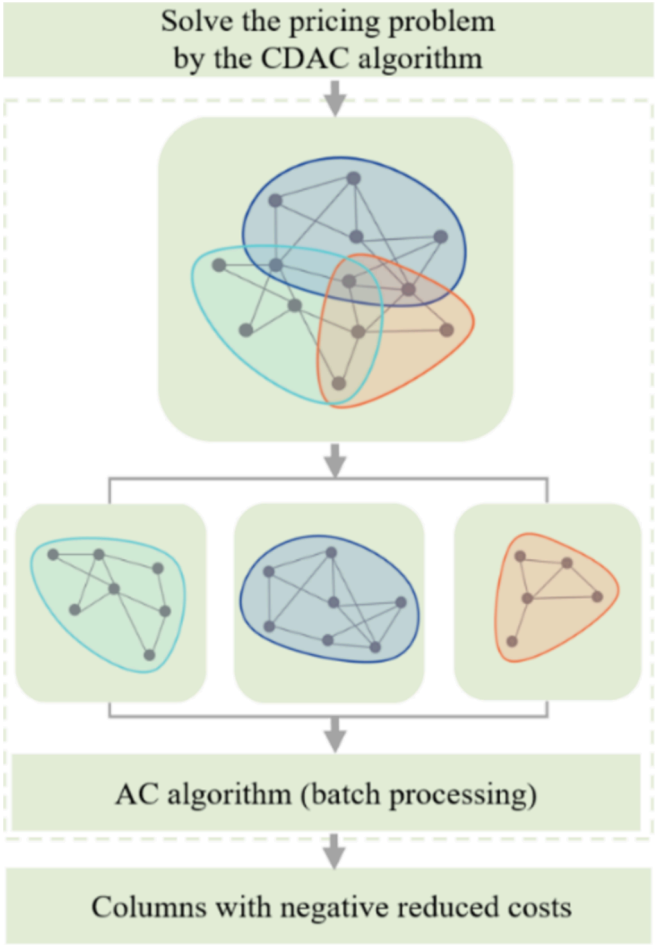


Fig. 3. Outline of CDAC.

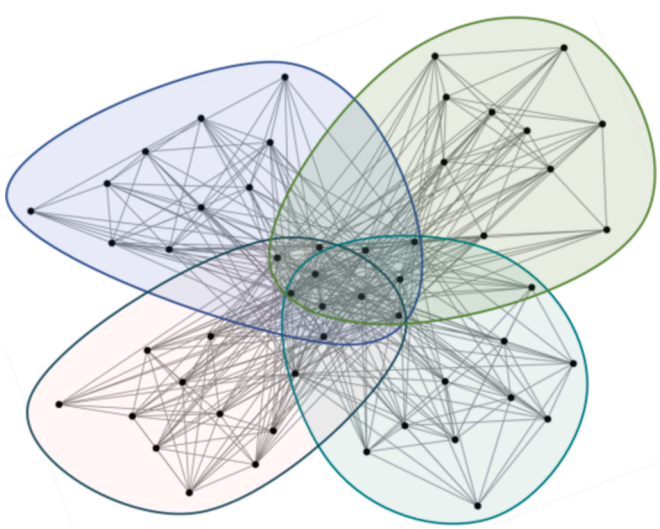


Fig. 4. Example of overlapping community.

introduces the possibility of generating duplicate columns, thereby slowing down the optimization process for the problem. Moreover, too many communities increase the computational burden. The proposed CD algorithm seeks to balance the trade-off between conflict density, potential overlaps among communities, and the computational burden associated with managing conflicts (The corresponding experiments can be found in Appendix B).

In terms of time complexity, let n denote the number of items. To speed up community-detection algorithm, inter-compatible Υ nodes are extended each time. The time complexity for each iteration is O (n*Υ). In the worst-case scenario, such as with a fully connected graph, the search depth becomes n/Υ, leading to an overall time complexity of O (n^2). However, in practice, the computation time for detecting communities can be considerably reduced due to the application of specific associated rules, including Rule A.1, Rule A.2, and Rule A.3. The details of the community-detection algorithm are shown in Appendix A.

(2) Actor-critic algorithm

After identifying the communities, we apply a deep reinforcement learning algorithm to each community to generate the columns for the pricing problem. We chose deep reinforcement learning over supervised learning for the following reasons: (1) In reinforcement learning, an agent learns to make decisions by interacting with its environment, making it well-suited for solving optimization problems like the pricing problem addressed in this study. In contrast, supervised learning involves training a model on a labeled dataset, where the goal is to minimize the discrepancy between predicted outputs and actual outputs. This approach is commonly used for tasks such as image classification, spam detection, and regression. However, supervised learning is often not applicable to most combinatorial optimization problems due to the challenges associated with obtaining optimal labels (Bello et al., 2016). (2) Deep reinforcement learning is capable of optimizing long-term objectives through a series of successive decision-making processes. In this study, it can generate more effective columns by learning how partial path expansions influence the reduced cost of these columns. Additionally, deep reinforcement learning does not require labeled data in the traditional sense, whereas supervised learning necessitates a large volume of labeled data for training, which is difficult to acquire in this context.

Typically, the deep reinforcement learning technique, specifically the actor-critic (AC) algorithm, has been commonly employed to tackle problems such as the traveling salesman problem or vehicle routing problem through the implementation of the DRL technique, enabling the efficient generation of a significant number of routes. In this study, the AC algorithm simultaneously applies to all communities, resulting in the generation of multiple permutations of items, specifically packing patterns.

A pricing problem instance, denoted as Θ, is defined as a graph consisting of n items. Each item i is described by its features α_i. A solution to the pricing problem, denoted as $\aleph = (\aleph_1, \aleph_2, \cdots, \aleph_n)$, represents a permutation of items. The probability, p_i, of item i being packed at a specific time step t is defined as follows.

$$p_i = p(\aleph_t = i | \Theta, \aleph_{1:t-1}) \tag{26}$$

The attention-based encoder-decoder model establishes a stochastic policy, $p(\aleph|\Theta)$, responsible for selecting a solution, $\aleph$, for a given problem instance, denoted as Θ. This policy is factorized and parameterized by $\hbar$ in Equation (27).

$$p_\hbar(\aleph|\Theta) = \prod_{t=1}^{n} p_\hbar(\aleph_t | \Theta, \aleph_{1:t-1}) \tag{27}$$

The architecture of the AC algorithm is shown as follows.

In Fig. 5, before solving the pricing problem, a strategy for filtering item is introduced, where items with non-positive reduced costs are removed. This is because only items with positive reduced costs can lower the objective value of the pricing subproblem when placed in the bin.

4.3.1.1. Encoder. The AC algorithm employs an encoder to analyze all the items within each community of the input graph. In this research, the input features are defined as Eqs. (28)-(32).

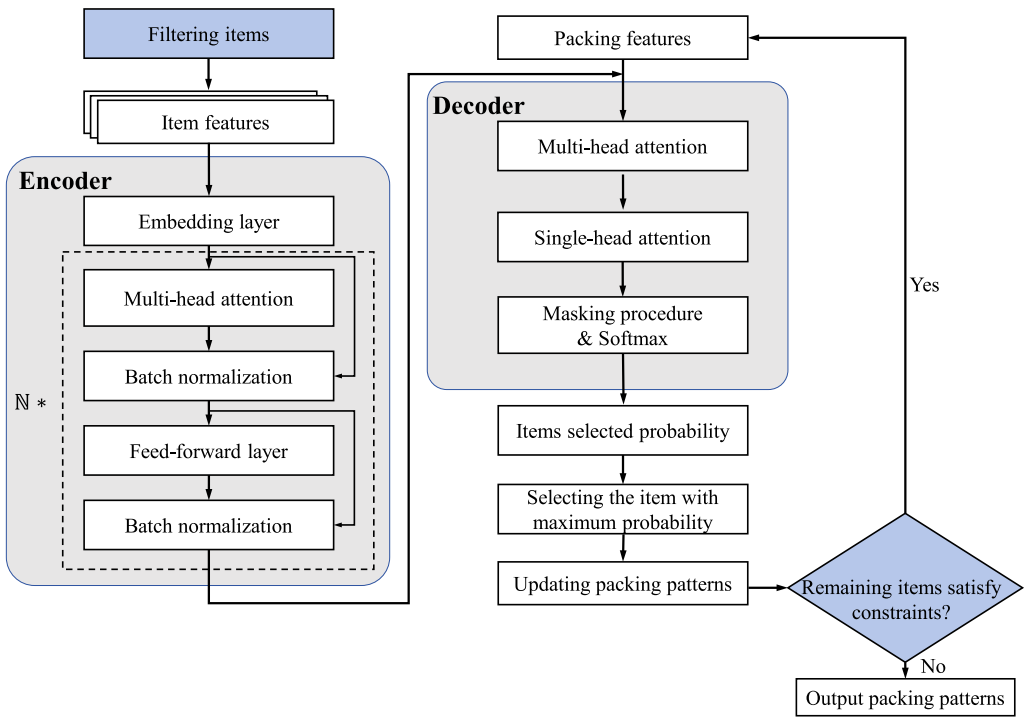


Fig. 5. The architecture of the AC method.

$$\alpha_i^1 = w_i/W, \tag{28}$$

$$\alpha_i^2 = v_i/V, \tag{29}$$

$$\alpha_i^3 = \pi_i/c_j, \tag{30}$$

$$\alpha_i^4 = d_i/(n-1), \tag{31}$$

$$\alpha_i^5 = \eta_i/(d_i \cdot (d_i - 1)/2), \tag{32}$$

where α_i^1 represents the normalized item weight obtained by dividing the weight of item i by the weight of the bin. α_i^2 denotes the normalized item volume, which is determined by dividing the volume of item i by the volume of the bin. α_i^3 refers to the normalized dual value of the item i. α_i^4 signifies the item degree of centrality. α_i^5 denotes the clustering coefficient of item i. d_i represents the degree of item i, η_i represents the number of edges between two neighboring nodes of item i.

The encoder receives a graph containing item features as input and generates embeddings, which uses the multiple attention layers, each of which contains two sub-layers: the multi-head attention layer and the fully connected feed-forward layer. Each layer employs skip connection and batch normalization (Kool et al., 2018).

4.3.1.2. Decoder. The packing features, such as remaining weight capacity and remaining volume of the bin, and the encoder's output are fed into the decoder. The decoder calculates an attention layer and obtains the probabilities of items being packed at the current step. A packing pattern is generated using a greedy strategy, which involves sequentially choosing the item with the highest probability. To prevent item repetition in the same column, the decoder detects a mask to identify which item has been packed. Additionally, items that do not meet the constraints are also masked. Specifically, item q is masked under any of the following conditions.

$$\pi_q < 0, \tag{33}$$

$$W - \sum_{i \in R} w_i - w_q < 0, \tag{34}$$

$$V - \sum_{i \in R} v_i - v_q < 0, \tag{35}$$

$$(i, q) \in E, \forall i \in R, \tag{36}$$

where R is the set of items that have been packed.

It is important to note that in certain specific scenarios where the density of the conflict graph is relatively low and the time window range is quite large, the CD algorithm can produce only a limited number of communities when grouping all items. Consequently, this leads to few columns with negative costs when using the greedy strategy for item selection, which diminishes the advantages of subsequent batch processing in the AC algorithm and reduces the overall efficiency of the algorithm. To tackle this issue, when the number of communities obtained falls below a threshold $\Re$, we increase the number of communities by replicating the existing ones until the total equals the threshold. For these newly generated communities, we employ a random sampling strategy instead of the greedy strategy to select items for generating columns.

4.3.1.3. Reinforce with greedy rollout baseline. To facilitate model training, a policy gradient approach with a greedy rollout baseline is employed (Kool et al., 2018). This method involves two networks: (a) an actor-network that utilizes Monte-Carlo sampling and continually updates its parameters to predict the probability distribution, and (b) a critic network that utilizes a greedy rollout policy to estimate the reward for a given state. The critic network, serving as a baseline, is only updated when the current policy demonstrates significant improvement compared to the baseline policy. Having a reliable baseline policy enhances learning speed typically.

4.3.2. Heuristic pricing with decomposed label-setting algorithm

Label-setting algorithm, as an exact DP approach, has been commonly used to solve the pricing problem of packing problem (Feillet

et al., 2004; Wei, Luo, et al., 2020). However, the label extension process in the label-setting algorithm can be time-consuming. To address this issue, we propose a decomposed label-setting algorithm as shown in Fig. 6. The decomposed label-setting algorithm is a heuristic label-setting algorithm, which prioritizes search spaces that have more potential to generate columns with a negative reduced cost, thus, for the pricing problems with small-sized bin, where the search space is smaller, decomposed label-setting algorithm can obtain a high-quality and stable solution in a reasonable computational time.

To illustrate the decomposed label-setting algorithm, we present the definitions of labels, the process of label extension, and the dominance rules as follows.

(1) Label definition

Label $L_i = \{w, v, c, \kappa, \psi\}$represents a partial packing pattern with multidimensional resources from the first item to item i. Each label has the following identities.

$w(L_i)$: The cumulated weight of items after packing item i.

$v(L_i)$: The cumulated volume of items after packing item i.

$c(L_i)$: The cumulated reduced cost of packing pattern after packing item i.

$\kappa(L_i)$: The set of items packed after packing item i.

$\psi(L_i)$: The set that is compatible with items in $\kappa(L_i)$.

(2) Label extension

In the label-setting algorithm, when considering label L_i, it is permissible to extend it to item h, $h \in I$, only if all constraints are satisfied. By extending label L_i, a new label L_h is generated as follows.

$$w(L_h) = w(L_i) + w_h, \tag{37}$$

$$v(L_h) = v(L_i) + v_h, \tag{38}$$

$$c(L_h) = c(L_i) - \pi_h, \tag{39}$$

$$\kappa(L_h) = \kappa(L_i) \cup \{h\}, \tag{40}$$

$$\psi(L_h) = \psi(L_i) \backslash \mathbb{R}_h \backslash \{h\}, \tag{41}$$

where $\mathbb{R}_h$ is the set of items that are incompatible with set $\kappa(L_h)$.

(3) Dominance rules

To reduce the number of labels, it is necessary to proactively delete certain labels that cannot be extended to the optimal solution (Duman et al., 2022). Consider two labels, L_1 and L_2, with the same index representing the last item in the packing pattern. If the following constraints are satisfied, then label L_1 dominates label L_2.

$$w(L_1) \le w(L_2) \tag{42}$$

$$v(L_1) \le v(L_2), \tag{43}$$

$$c(L_1) \le c(L_2), \tag{44}$$

$$\psi(L_1) \supseteq \psi(L_2). \tag{45}$$

Constraint (42) shows that the weight of label L_1 is less than or equal to that of L_2. Constraint (43) denotes that the volume of label L_1 is less than or equal to that of L_2. Constraint (44) indicates that the reduced cost of label L_1 is less than or equal to that of L_2. Constraint (45) represents that the compatible set of label L_2 is a subset of the compatible set of L_1.

(4) The framework of the decomposed label-setting algorithm

Based on the experiments in Appendix C, it has been found that more than 95 % of the columns with negative reduced costs contain items with large dual values. Therefore, in order to speed up the label extension process, a predetermined number of items with large dual values are initially chosen in the proposed decomposed label-setting algorithm using the following rule:

Rule 1. Item h is selected, if

$$\pi_h \ge \sum_{i \in I} \pi_i \Big/ n \tag{46}$$

where π_h is the dual value of item h.

Subsequently, the label extension begins with the selected items. The framework of the decomposed label-setting algorithm is shown in Appendix D.

It is noted that items with positive dual values are expanded in Algorithm B, since only these items effectively contribute to reducing the objective function value of the pricing problem. The termination condition of Algorithm B is reached either when no more labels are available for expansion or when the number of columns exceeds twice the number of items.

To reduce the creation of duplicate columns, we propose two acceleration policies. Under the first strategy, when a node (e.g., node r) is chosen to initiate label extension, it will not be extended to any subsequent expansion process initiated by other nodes. This is because extending a label to node r would result in the same packing pattern that is generated when node r initiates the label extension. This redundancy consumes unnecessary computation.

The second policy ensures that the selected item with a higher index

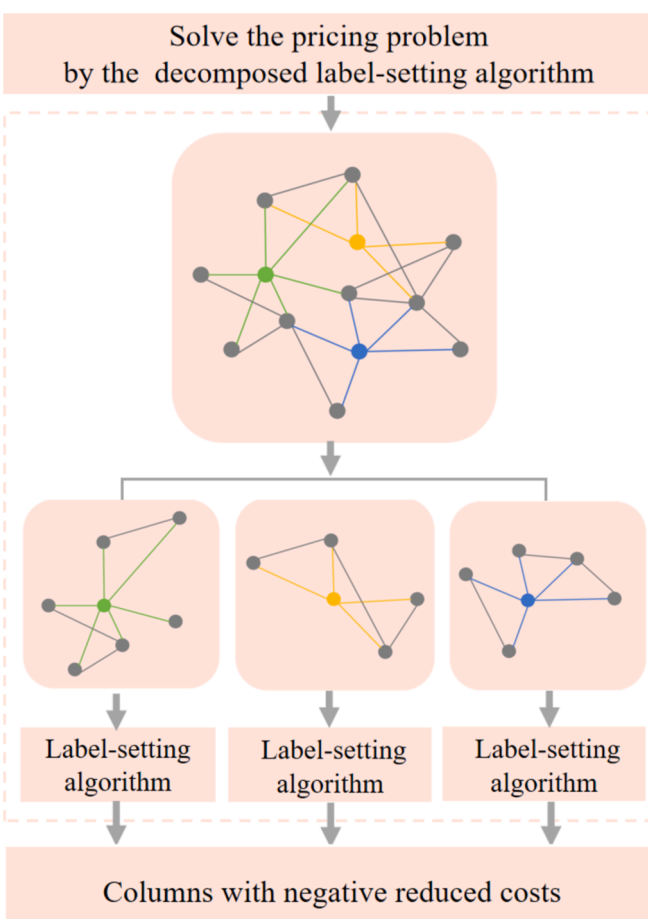


Fig. 6. Outline of decomposed label-setting algorithm.

does not extend to the selected item with a lower index, as it would result in duplicate columns. For instance, we have five items denoted as 1, 2, 3, 4, and 5. If items 1 and 3 are selected, and Column $\{1, 2, 3\}$ is extended by item 1 while Column $\{3, 1, 2\}$ is extended by item 3, it becomes evident that these columns represent the same packing patterns. Therefore, it is prohibited for item 3 to extend to item 1. Notably, due to the nature of label extension enumeration, the acceleration strategy does not cause any loss of columns.

4.3.3. Heuristic pricing algorithm selection

This study introduces two approaches, namely the CDAC algorithm and the decomposed label-setting algorithm, to tackle the pricing problem. Through experimental observations, when the average item size is smaller, the bin size is larger, or when the average dual value of items is smaller, the resulting columns become longer. Hence, the resulting columns include more items, making finding columns with negative reduced costs more challenging. Thus, it is reasonable to adaptively select an appropriate algorithm for solving the pricing problem, considering the specific characteristics of the instances and the pricing problem itself. To address this, we propose Rule 2 as follows.

Rule 2. To tackle pricing problem j, the CDAC algorithm is utilized when the specified condition $\max\left(\frac{\sum_{i\in I} w_i/n}{W_j}, \frac{\sum_{i\in I} v_i/n}{V_j}, \frac{\sum_{i\in I} \pi_i/n}{c_j}\right) \leq \theta$ is met; otherwise, the decomposed label-setting algorithm is employed.

According to Rule 2, the CDAC algorithm is employed to address the pricing problem when dealing with longer columns consisting of more items because the CDAC algorithm can efficiently generate packing patterns that accommodate a greater quantity of items through methods such as community detection and batch processing. The value of θ, a predefined parameter, can be determined through experimental evaluation.

4.3.4. Exact pricing

Two heuristic algorithms, the CDAC and the decomposed label-setting algorithm, are employed to efficiently address the pricing problem in a timely manner. When both methods fail to identify columns with negative reduced costs, we turn to a commercial solver for an optimal solution to the pricing problem. In the past year, advanced commercial solvers, such as the GUROBI, have demonstrated significant improvements in performance, offering not only reduced computational times but also enhanced stability and reliability when tackling optimization problems with complex constraints. Consequently, we utilize the GUROBI solver to optimally solve the pricing problem associated with the 2DVVSPPCTW.

4.4. Finding an integer solution

The CG technique is commonly used to solve LP models, but the solutions obtained using this method are typically fractional. To obtain an integer solution, a branching policy is often preferred. However, the proposed 2DVVSPPCTW incorporates several complexities, such as item weight and volume, variable-sized bins, time windows, and conflict constraints. Due to these challenges, finding an efficient branching policy that can avoid an extensive search tree is difficult (Yu et al., 2022). Additionally, through experimental observations (refer to Appendix E), we have found that a high ratio of integer solutions can be obtained at the root node of the RMP, indicating a high-quality LP bound. Therefore, we adopt the policy proposed by Firat et al. (2020) to obtain an integer solution for the 2DVVSPPCTW. Specifically, we convert the master LP model with the columns enumerated so far into an integer linear programming (ILP) model. We then employ a commercial solver to solve the ILP model, resulting in an integer solution. Our experiments demonstrate that a standard solver can quickly find the optimal solution for these ILP models. Similar to Firat et al. (2020), our experiments show that the gap between the lower and upper bounds obtained from the proposed CG algorithm is small, indicating that the obtained integer solution is close to optimal.

4.5. Acceleration strategies

4.5.1. Pre-processing for partial solutions

To improve the efficiency of the proposed algorithm, particularly for large-scale instances, we develop a preprocessing strategy for partial solutions. Initially, the compatible matrix is generated based on Definition 1. During this process, it is observed that certain items have a limited number of compatible items or no compatible items at all in the matrix. Consequently, partial solutions can be generated by applying the following rules.

Rule 3. If item i does not have any compatibility with other items, it is packed into an available bin with the smallest cost. This results in the creation of a packing pattern.

Rule 4. If item i is only compatible with one or two other items, which are also mutually compatible, they are packed together into an available bin with the smallest cost, leading to the formation of a packing pattern.

By applying Rules 3 and 4, a partial solution is generated. The items that are packed according to these rules do not require further processing by the DRLCG algorithm, thereby reducing the computational burden. Once an integer solution is obtained through the DRLCG algorithm, a complete solution is formed by combining it with the previously obtained partial solution.

4.5.2. Tabu strategy for the pricing problem

In the 2DVVSPPCTW, each bin type corresponds to a pricing problem. To accelerate the solving process of pricing problems, a tabu strategy for the pricing problem is proposed. Initially, all pricing problems are sorted based on ascending bin cost. These problems are then sequentially solved. If a pricing problem fails to find a column with a negative reduced cost, it is marked as tabu and excluded from the next iteration to minimize the search space, as it is likely to find no column with a negative reduced cost for the same pricing problem in subsequent iterations. If no columns with a negative reduced cost can be found for the pricing problems outside the tabu list, the pricing problems within the tabu list are removed and solved individually. The process continues until all pricing problems cannot find a column with a negative reduced cost.

5. Computational results

The proposed algorithm was implemented using the Python programming language, and the AC algorithm was implemented in Pytorch. All experiments were conducted on a PC running Windows 10 Professional 64-bit operating system with an Intel Xeon W-2245 CPU and 32 GB RAM. Gurobi Optimizer 9.1 was employed as the linear relaxation solver for the RMP and the exact solver for the pricing problem. A time limit of 3600 s is applied to all experiments. To train the AC, we randomly generate instances of the 2DVVSPPCTW described in Section 5.1, and then obtain instances of pricing problem by iterative solving them by the CG. Thereby, 30,000 instances of pricing problem are generated for each size of problem. We train for 100 epochs using training instances. To provide the best performance for the DRLCG algorithm, we use the IRACE package (López-Ibáñez et al., 2016) to obtain the best configuration settings for parameters related to the CD algorithm and the heuristic pricing algorithm selection rules within the DRLCG. The hyperparameters of AC algorithms are set based on Kool et al. (2018). The parameters of the DRLCG algorithm are listed in Table 3. The detailed configuration settings are shown in Appendix F.

5.1. Instance generation

Since the 2DVVSPPCTW is a new variant of the variable-sized bin-

Table 3
Parameters in the DRLCG algorithm.

Parameter	Value	Parameter	Value
$\Re$(see Appendix A)	1024	Batch size	256
γ (see Appendix A)	5	Embedding size	128
ς_1(see Appendix A)	2	Hidden layer size	512
ς_2(see Appendix A)	0.9	Learning rate	0.0001
ς_3(see Appendix A)	20	Number of attention layers	3
θ	0.25	Number of attention heads	8
		Optimizer	Adam

packing problem, no existing benchmark instances is available. Therefore, we generated instances for the 2DVVSPPCTW based on the well-known benchmark instances designed for the variable-sized bin-packing problem by Haouari and Serairi (2009), incorporating additional considerations for item volume, time window, and conflict constraints. In these instances, there are a total of seven types of bins available. The bin weights and volumes are $W_1 = V_1 = 70$, $W_2 = V_2 = 100$, $W_3 = V_3 = 130$, $W_4 = V_{41} = 160$, $W_5 = V_5 = 190$, $W_6 = V_6 = 220$, and $W_7 = V_7 = 250$. Additionally, the cost functions are classified into three types, each accompanied by specific functions and relevant real-world scenarios as detailed below.

Type 1. Linear cost function, $C_j = W_j (j = 1, 2, \cdots, 7)$.

In scenarios characterized by a linear cost function, a bin cost increases directly in proportion to the weight capacity of the bin. This type of cost function is particularly relevant in the context of bulk commodities or large-scale deliveries. For instance, logistics companies often adopt straightforward pricing strategies where the packing and delivery cost for bulk goods is uniformly calculated based on weight or volume.

Type 2. Concave cost function, $C_j = \lfloor 10 \cdot \sqrt{W_j} \rfloor (j = 1, 2, \cdots, 7)$.

A concave cost function indicates that the cost associated with a bin rises at a decreasing rate as its weight capacity increases. This phenomenon often illustrates economies of scale, where larger volumes result in lower per-unit costs. A typical example of this can be seen in bulk packaging for retail products, where packing items in larger quantities lowers the average packing cost per item due to reduced labor and material expenses.

Type 3. Convex cost function, $C_j = \lfloor 0.1 \cdot W_j^{3/2} \rfloor (j = 1, 2, \cdots, 7)$.

A convex cost function suggests that the cost of bin increases at an accelerating rate as its weight capacity increases, often due to the complexities and constraints encountered during the packing process. This is particularly relevant in industries that require custom packaging solutions, such as electronics or fragile goods. In these cases, the initial costs for packing a small number of items may be relatively low, but as the quantity increases, the need for specialized materials, protective packaging, and additional handling can significantly escalate costs.

The time windows for the items are generated using the benchmark established by Liu et al. (2021). The time interval length is varied across 1, 2, 4, 8, 16, 32, 64, and 128. Additionally, the densities of the conflict graph are varied across a range from 0.1 to 0.9, with increments of 0.1 (Ekici, 2023). In the first set of instances, similar to Haouari and Serairi (2009), the number of items is set to 100, 200, 500, 1000, and 2000. The weights and volumes of the items are uniformly distributed within the range of [1, 250].

To include instances with a higher number of items packed into each bin, the second set is generated based on the benchmark established by Caprara and Toth (2001). In this set, the weights and volumes of the items are uniformly distributed within the range of [13, 50], which matches the distribution used in the instance of class 4 of Caprara and Toth (2001). The number of items is set to 100, 200, and 300.

As a result, a total of 1728 instances are generated by combining the two sets. This leads to a total of 21168, which is $72 \times 6 + 1728 \times 12$, experiments being conducted.

5.2. Performance of the DRLCG

In this section, we evaluate the performance of the DRLCG method by conducting a comparative analysis with five algorithms: GUROBI solver, a primal heuristic based on the CG algorithm (PH) (Liu et al., 2021), a variable neighborhood search metaheuristic (VNS) proposed by Hemmelmayr et al. (2012, a local improvement subset-sum heuristic algorithm (SSP) (Haouari & Mhiri, 2024), and a LNS proposed by Ekici (2023).

Liu et al. (2021) proposed a primal heuristic based on column generation, called the PH algorithm, for solving the VSBPPTW problem. To address the 2DVVSPPCTW, we modified the set-partitioning formulation by incorporating volume and conflict constraints for items. The pricing problem is solved using the GUROBI solver instead of the DP algorithm due to the DP algorithm's limited effectiveness in handling pricing problems with conflicting items.

Hemmelmayr et al. (2012) proposed the VNS algorithm for the VSBPP problem. We extend the VNS algorithm to handle the 2DVVSPPCTW by incorporating volume, time windows, and conflict constraints. We utilize the first fit algorithm instead of the lower bound heuristic to generate the initial solution. This decision is driven by the incorporation of multiple constraints, which make the computation of the lower bound extremely time-consuming. To ensure a fair performance comparison between the two algorithms, we set the time limits of the VNS algorithm equal to the computation time of the DRLCG algorithm for each instance.

Haouari and Mhiri (2024) derived a bin packing problem with concave loading cost, which is solved by CG and heuristics. The result of CG is considered as a lower bound. Besides, a local improvement subset-sum heuristic, called the SSP algorithm, is proposed. To address the 2DVVSPPCTW, we solved the subset-sum problem by the GUROBI solver instead of the heuristic due to the conflict constraint. The SSP algorithm is limited to the same computation time as the DRLCG algorithm.

Ekici (2023) proposed the LNS algorithm for the VSBPPC. We extend the LNS algorithm to handle the 2DVVSPPCTW by incorporating volume and time windows constraints. To ensure a fair performance comparison between the two algorithms, we set the time limits of the LNS algorithm equal to the computation time of the DRLCG algorithm for each instance.

Figs. 7 and 8 present a comparison between the DRLCG algorithm and the GUROBI solver. Due to the computational challenges the GUROBI solver faces in solving instances with 100 items within the time limit of 3600 s, it is not utilized for solving large-scale instances. Figs. 9 and 10 comprehensively compare the DRLCG, PH, VNS, SSP, and LNS algorithms. The detailed comparison results are shown in Appendix G.

For the first set of instances, the DRLCG algorithm can obtain optimal solutions for 191 instances out of 216 instances, including the optimal solution obtained because of the integer solution at the root node, and the same optimal solution as that obtained by the GUROBI solver. In contrast, the GUROBI solver only manages to find the optimal solutions for 96 instances. Moreover, the results presented in Fig. 7 demonstrate that the average computational time for the GUROBI solver is 2338.66 s, while the DRLCG only requires 0.93 s. For the second set of instances, the results presented in Fig. 8 demonstrate that the DRLCG algorithm can obtain optimal solutions for 77 instances out of 216 instances. In contrast, the GUROBI solver only manages to find the optimal solutions for 53 instances. Moreover, the average computational time for the GUROBI solver is 2964.62 s, while the DRLCG only requires 15.61 s. Notably, the DRLCG algorithm achieves optimal solutions for 62.04 % of instances with a size of 100 items in a short amount of time, while the remaining instances cannot be verified to be optimal because the GUROBI solver is unable to provide an optimal solution.

In Figs. 9 and 10, a comparison is made among the DRLCG, PH, VNS, SSP, and LNS algorithms for all instances. For the first set of instances, the PH algorithm exhibits an average cost gap of 4.66 % and an average computational time of 1112.94 s. The VNS algorithm shows an average

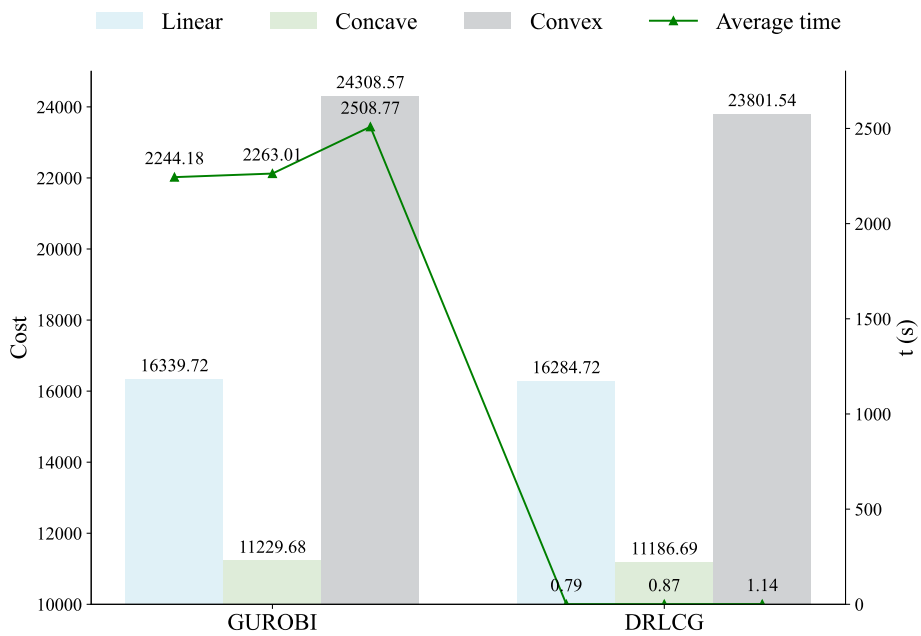


Fig. 7. Comparison of results with GUROBI for small-scale instances of the first set.

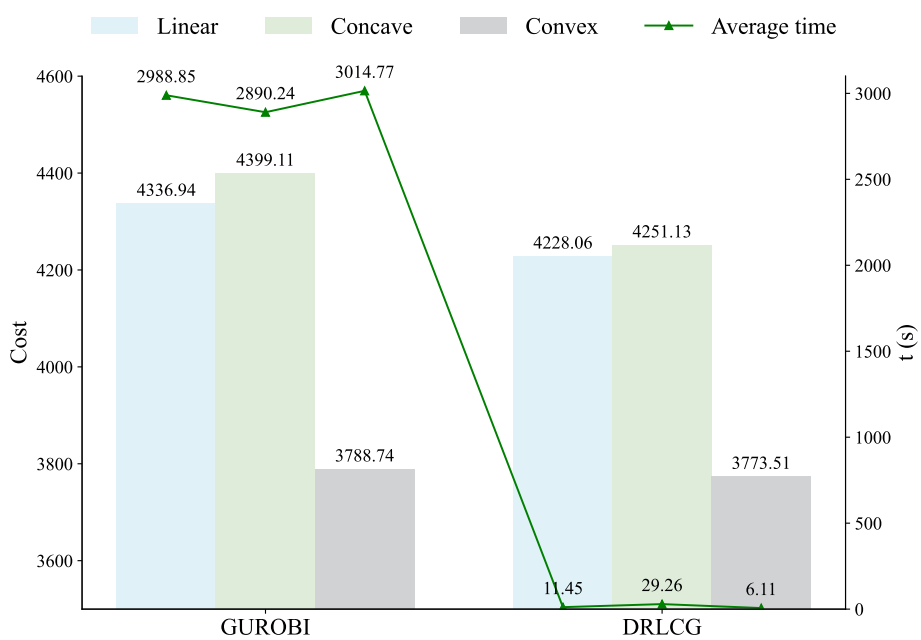


Fig. 8. Comparison of results with GUROBI for small-scale instances of the second set.

cost gap of 14.16 % and an average computational time of 361.81 s. The SSP algorithm exhibits an average cost gap of 4.68 % and an average computational time of 361.70 s. The LNS algorithm shows an average cost gap of 1.34 % and an average computational time of 1190.67 s. Moving on to the second set of instances, which is more challenging due to a larger number of items packed into each bin, the PH algorithm demonstrates an average cost gap of 19.65 % and an average computational time of 1331.62 s. Meanwhile, the VNS algorithm exhibits a significantly larger average cost gap of 67.94 % and an average computational time of 102.15 s. Besides, the SSP algorithm demonstrates an average cost gap of 12.95 % and an average computational time of 102.68 s. The LNS algorithm exhibits a larger average cost gap of 7.89 % and an average computational time of 129.64 s. Consequently, for both sets of instances, the DRLCG algorithm outperforms the PH, VNS, SSP and LNS algorithms regarding solution quality and computation time.

To further evaluate the performance of the DRLCG method in additional applications, we apply it to the VSBPP and compare its results with those obtained using a genetic algorithm (GA) proposed by Haouari and Serairi (2009) specifically for the VSBPP. In addressing the VSBPP, we adapted the set-partitioning formulation by omitting volume, time windows, and conflict constraints for items in the DRLCG approach. The results, displayed in Table G.8 (in Appendix G), demonstrate that the DRLCG algorithm outperform the GA algorithm in terms of total cost while maintaining similar computation time.

We also utilize the DRLCG method to tackle the VSBPPTW and compare its performance with that of a primal heuristic (PH) based on the CG algorithm developed by Liu et al. (2021) specifically for the

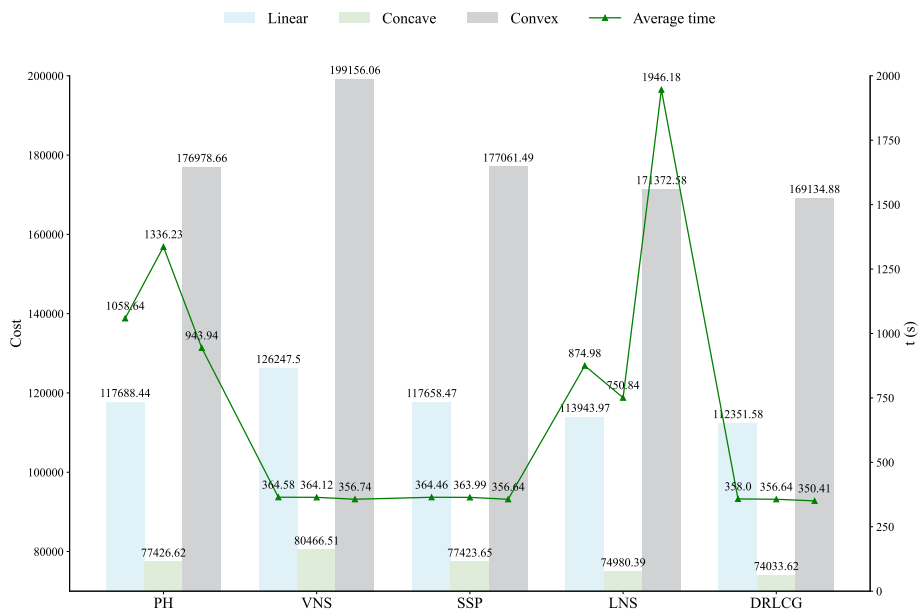


Fig. 9. Comparison of results for instances of the first set.

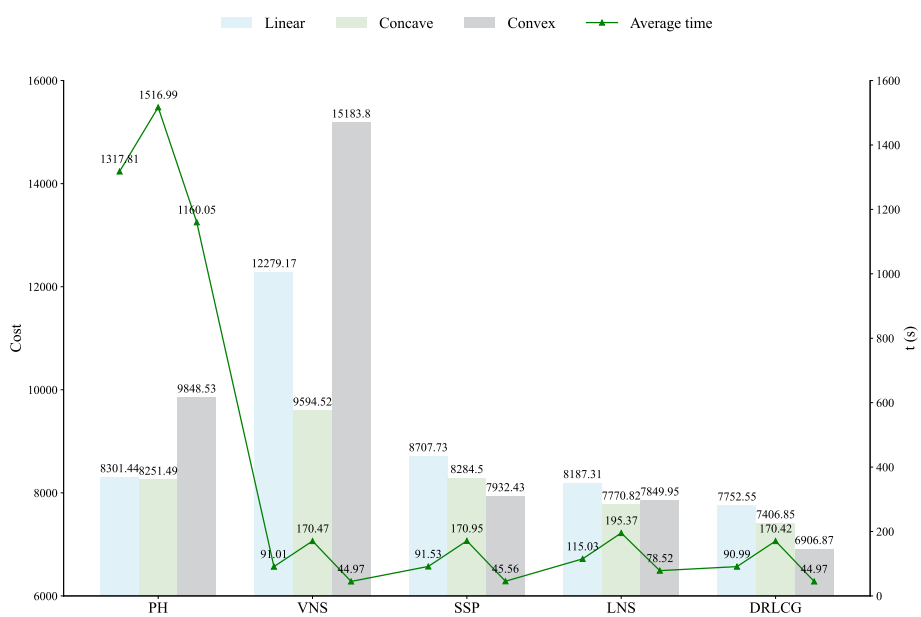


Fig. 10. Comparison of results for instances of the second set.

VSBPPTW. In adapting the DRLCG algorithm for the VSBPPTW, we adjusted the set-partitioning formulation by excluding volume and conflict constraints for the items. The comparative results, presented in Table G.9 (in Appendix G), indicate that the DRLCG algorithm not only achieves lower costs than the PH algorithm but also produces 32.30 % more optimal solutions compared to the PH algorithm.

5.3. Ablation analysis

5.3.1. Effectiveness of the heuristic pricing

To evaluate the performance of heuristic pricing, we compare the DRLCG approach and the DRLCG algorithm without heuristic pricing. The DRLCG algorithm without heuristic pricing is a modified version of the DRLCG where the decomposed DP and CDAC algorithms are removed, and the pricing problem is solely solved using the GUROBI solver. The results are presented in Figs. 11 and 12. The detailed comparison results are shown in Appendix H.

According to Fig. 11, the computational time for instances of the first set using the DRLCG with and without heuristic pricing is recorded as 355.02 and 608.90 s, respectively. It can be observed that the DRLCG without heuristic pricing takes approximately twice as long as the DRLCG approach. Regarding the results for instances of the second set, as shown in Fig. 12, the computational time for the DRLCG without heuristic pricing is 19.17 % longer than that of the DRLCG method. Thus, the heuristic pricing approach significantly reduces the computational time required for column generation.

5.3.2. Effectiveness of tabu strategy for the pricing problem

To investigate the performance of the tabu strategy for the pricing problem, we analyze by removing the tabu strategy for the pricing problem from the proposed DRLCG algorithm. The comparison results between the DRLCG with and without the tabu strategy for the pricing

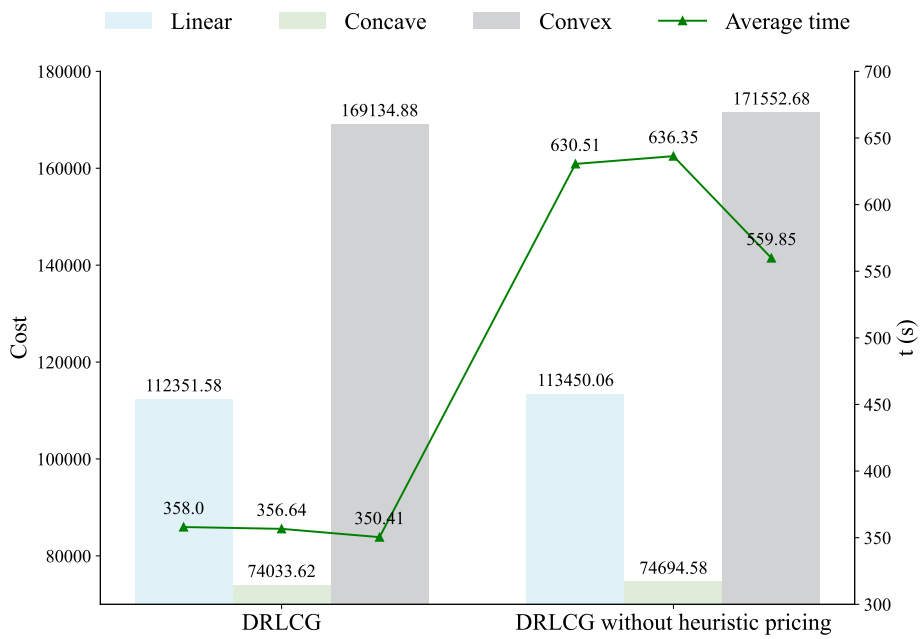


Fig. 11. Comparison of results between DRLCG with and without heuristic pricing for instances of the first set.

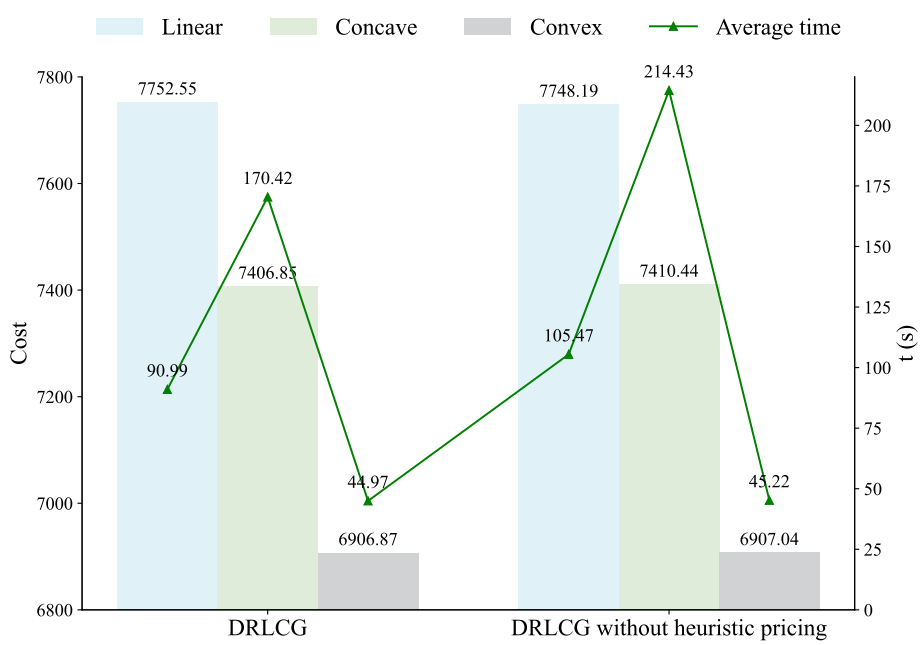


Fig. 12. Comparison of results between DRLCG with and without heuristic pricing for instances of the second set.

problem can be found in Figs. 13 and 14. The detailed comparison results are shown in Appendix H.

The results presented in Figs. 13 and 14 demonstrate the impact of the tabu strategy for the pricing problem on the computational time of the DRLCG algorithm. For instances of the first set, the DRLCG with tabu strategy for the pricing problem takes 355.02 s, while the DRLCG without tabu strategy for the pricing problem takes 432.50 s. It indicates that the computational time for the DRLCG without tabu strategy for the pricing problem is 21.82 % longer than that for the DRLCG with tabu strategy for the pricing problem. Similarly, for instances of the second set, the DRLCG with tabu strategy for the pricing problem takes a computational time of 102.13 s, while the DRLCG without tabu strategy for the pricing problem takes 110.75 s. The computational time for the DRLCG without tabu strategy for the pricing problem is 8.44 % longer than that for the DRLCG with tabu strategy for the pricing problem. Although the cost of the DRLCG algorithm remains similar with or without tabu strategy for the pricing problem, it is evident that utilizing the tabu strategy for the pricing problem reduces the computational burden of the column generation procedure.

5.3.3. *Effectiveness of the pre-processing strategy*

We evaluate the effectiveness of the pre-processing strategy by removing it from the proposed DRLCG algorithm. The results are presented in Figs. 15 and 16. The detailed comparison results are shown in Appendix H.

Figs. 15 and 16 summarize the impact of the pre-processing on the computational time of the DRLCG algorithm. For instances of the first set, the DRLCG with pre-processing takes 355.02 s, while the DRLCG

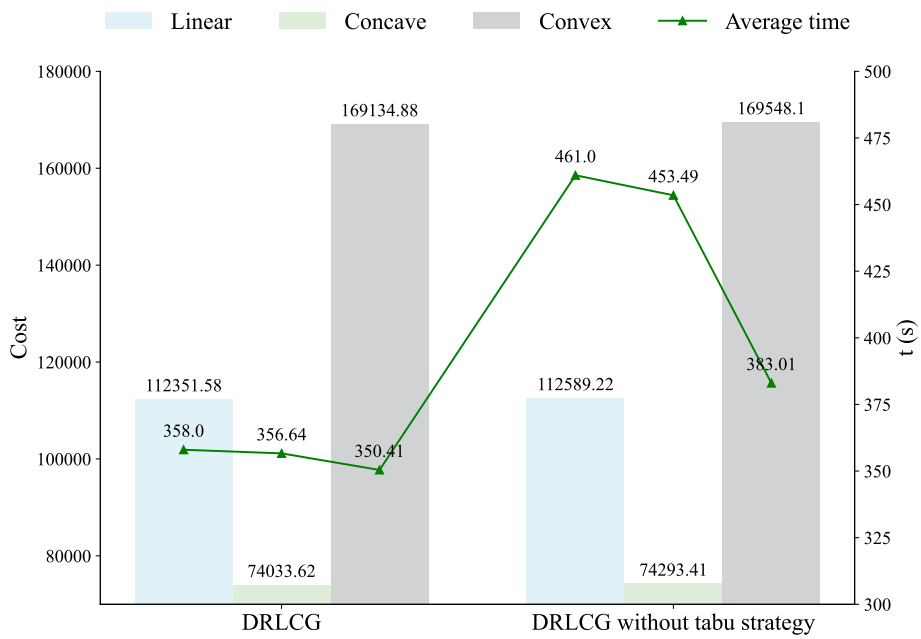


Fig. 13. Comparison of results between DRLCG with and without tabu strategy for instances of the first set.

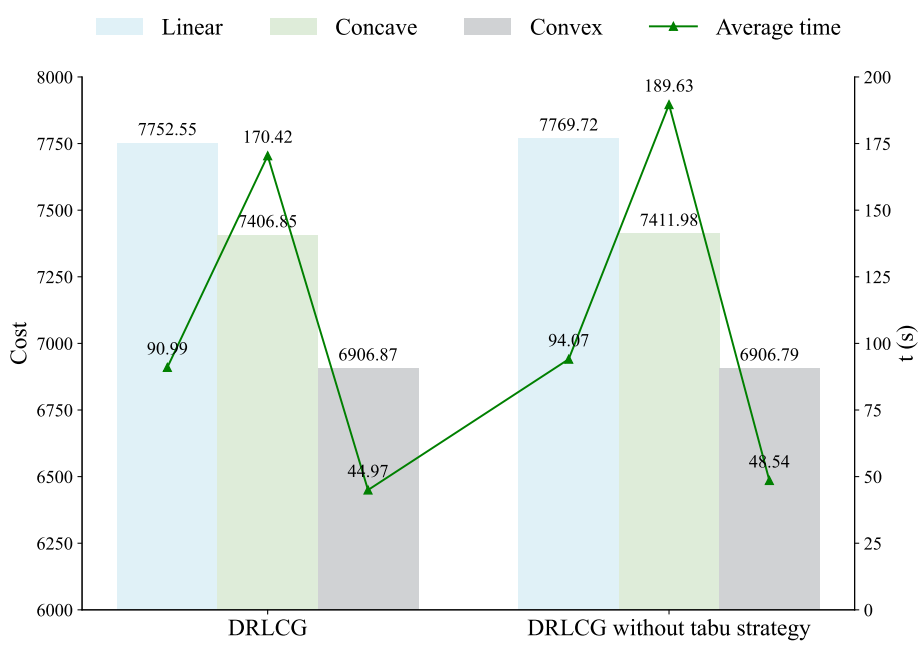


Fig. 14. Comparison of results between DRLCG with and without tabu strategy for instances of the second set.

without pre-processing takes 681.70 s. It indicates that the computational time for the DRLCG without pre-processing is 92.02 % longer than that for the DRLCG with pre-processing. Similarly, for instances of the second set, the DRLCG with pre-processing takes a computational time of 102.13 s, while the DRLCG without pre-processing takes 134.02 s. The computational time for the DRLCG without pre-processing is 31.22 % longer than that for the DRLCG with pre-processing. Hence, it is obvious that employing preprocessing can reduce the computational burden of the column generation process. Besides, the cost of the DRLCG algorithm with pre-processing is slightly less than that of the DRLCG algorithm without pre-processing.

5.3.4. Effectiveness of the TRFF, CDAC, and DRL algorithm

(1) Effectiveness of the TRFF

We developed the TRFF as an initial solution technique for the 2DVVSPPCTW. In existing studies, the subset sum algorithm (SSP) (Haouari & Serairi, 2009) and the best fit (BF) algorithm (Liu et al., 2021) are commonly used as initial solution techniques for the variable-sized bin packing problem and the variable-sized bin packing problem with time windows, which are closely related to the 2DVVSPPCTW. Consequently, these two algorithms can serve as initial solution techniques for the 2DVVSPPCTW.

The SSP algorithm is a step-by-step constructive heuristic designed to iteratively solve a subset-sum problem. When applied to the

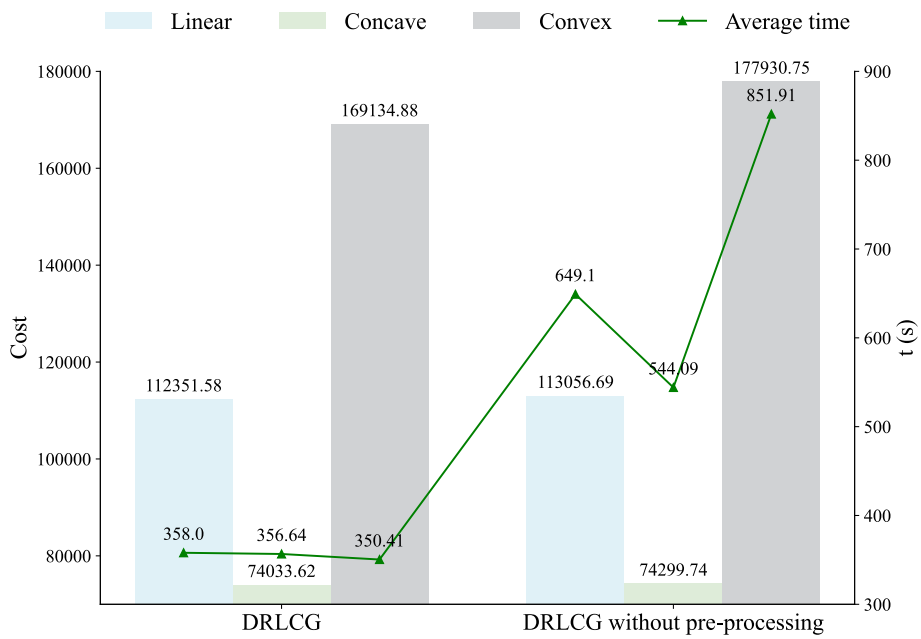


Fig. 15. Comparison of results between DRLCG with and without pre-processing for instances of the first set.

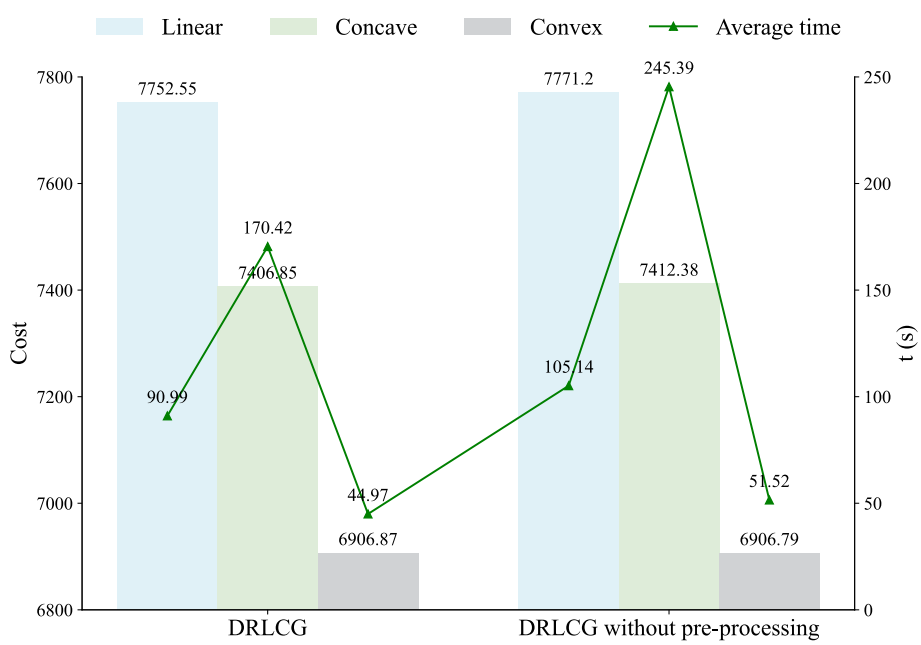


Fig. 16. Comparison of results between DRLCG with and without pre-processing for instances of the second set.

2DVVSPPCTW, the SSP algorithm must account for additional factors such as time windows, weight, volume, and conflict constraints. This complexity significantly increases the computation time.

The BF algorithm inserts items using a greedy approach, which makes the time windows of the bin narrower as more items are added, resulting in difficulty in packing subsequent items. In contrast, the TRFF algorithm tends to prioritize items with earlier time windows by inserting them in ascending order based on their left time window. This strategy groups items with similar time windows into the same bin, preventing the creation of packing patterns with overly narrow time windows. As a result, fewer bins need to be used when more items are inserted.

To further evaluate the performance of the TRFF algorithm, we conducted a comparative experiment involving the DRLCG with the TRFF algorithm, the DRLCG with the BF algorithm, and the DRLCG with the SSP algorithm. The results, presented in Tables I.1 and I.2 (in Appendix I), indicate that the DRLCG with the TRFF algorithm outperforms both the DRLCG with the BF algorithm and the DRLCG with the SSP algorithm in terms of solution quality and computational time.

(2) Effectiveness of the community-detection-based actor-critic algorithm

We develop CDAC algorithm to efficiently address the pricing problem. The maximal clique approach, as noted by Tomita et al. (2006), is another well-established method for handling graphs, similar to the community-detection technique. By employing a maximal clique-based actor-critic algorithm, the pricing problem can similarly be

tackled. Utilizing the maximal clique technology ensures that no items within the same clique are in conflict, allowing the proposed problem to be broken down into several independent pricing problems. However, this approach also has its limitations, particularly when dealing with large-scale instances. For example, generating a maximal clique requires examining a vast number of nodes and edges, with a worst-case time complexity of O ($3^{n/3}$) as noted by Tomita et al. (2006). As the graph structure becomes more intricate and the number of nodes increases, the time required for computation also rises, and the number of maximal cliques can grow exponentially. This results in increased complexity when applying the maximal clique-based actor-critic algorithm. Furthermore, there is a significant degree of item overlap among cliques, meaning that some items may appear in multiple cliques simultaneously. This overlap not only increases computation time but also results in resource inefficiency when using the actor-critic approach. In contrast, the proposed community-detection algorithm effectively overcomes these limitations by implementing a more adaptable community division strategy.

To further evaluate the performance of the CDAC algorithm, we conducted a comparison experiment between the DRLCG with the CDAC algorithm and the DRLCG with the maximal clique-based actor-critic algorithm. To ensure that the result is obtained within a given time, the same maximum number of communities limit is applied to the maximal clique-based actor-critic algorithm. In experiments of Tables J.1 and J.2 (in Appendix J), the DRLCG with the CDAC algorithm outperforms the DRLCG with the maximal clique-based actor-critic algorithm regarding computational time.

(3) Effectiveness of the DRL algorithm

To evaluate the performance of the DRL algorithm, we substitute the proposed policy gradient approach with the deep Q-network (DQN) in the framework, resulting in the DQNCG algorithm. We performed experiments to compare the DQNCG with the DRLCG. The results are shown in Tables K.1 and K.2 (in Appendix K). It is observed that the proposed DRLCG, utilizing the policy gradient approach, outperforms DQNCG with DQN in terms of computation time.

Besides, we substitute the proposed AC approach with the heat-map (HM)method within the CDAD framework, resulting in the CDHM method. The heat-map method is another well-established end-to-end deep learning technique, which can also be utilized to predict the probability of each item being a component of the optimal solution, and generate the packing patterns based on the beam search (Joshi et al., 2019). The CDHM method divides all items into multiple communities, and generates of a substantial number of promising columns via the HM method.

Additionally, we substitute the proposed CDAC approach with the heat-map method within the DRLCG framework, resulting in the DRLCG with HM method, which generates of a substantial number of promising columns directly by HM method without the decomposition of the CD method.

We performed experiments to compare the DRLCG with CDAC, the DRLCG with CDHM, and the DRLCG with HM in Tables K.3 (in Appendix K). The results indicate that the proposed DRLCG, utilizing the CDAC approach, outperforms the DRLCG with the CDHM and the DRLCG with HM. Besides, the CD method can reduce the number of items that the learning method must handle, thus, the DRLCG with HM outperforms the DRLCG with CDHM.

6. Conclusions

In this study, we introduce the 2DVVSPPCTW driven by practical applications within the logistics industry. To address this problem, we propose the DRLCG algorithm that utilizes DRL and column generation techniques. We develop two tailored methods to solve the pricing problem efficiently. The first method is the CDAC algorithm, which is specifically designed for finding longer columns with a larger number of items. The second method is a decomposed label-setting algorithm more suitable for pricing problems with fewer items in a column. Additionally, we propose two acceleration strategies for the DRLCG algorithm.

We conducted comprehensive computational experiments to evaluate the performance of the DRLCG algorithm. The results are compared with the GUROBI solver and two existing approaches known as the PH and VNS algorithms. The findings reveal that the DRLCG algorithm achieved remarkable results for the first set of instances, solving them with an average solution time of 355.02 s. In contrast, the PH and VNS algorithms exhibited average computational times of 1112.94 s and 361.81 s, respectively. The average cost gap for the PH and VNS algorithms was 4.66 % and 14.16 %, respectively. Similarly, for the second set of instances, the DRLCG algorithm outperformed the PH and VNS algorithms, solving them with an average solution time of 102.13 s. On the other hand, the PH and VNS algorithms exhibited average computational times of 1331.62 s and 102.15 s, respectively. The average cost gap for the PH and VNS algorithms was 19.65 % and 67.94 %, respectively. In conclusion, the DRLCG algorithm performs best regarding solution quality and computational time for both sets of instances.

While the proposed DRLCG algorithm effectively tackles the complex combinatorial optimization problem, the 2DVVSPPCTW, it does have some limitations. One such limitation is that each bin type corresponds to a separate pricing problem that must be solved independently, resulting in a significant computational burden. Besides, the actor-critic algorithm used in this paper has limitations, which have low data utilization since each interaction experience is usually used only once and then discarded. Furthermore, the policy gradient approach's sampling process is random, resulting in high variance, so a baseline is used to reduce the variance. Moreover, the proposed CDAC algorithm and the pre-processing strategy within the DRLCG algorithm may face challenges in specific large-scale scenarios characterized by low conflict graph density and wide time window ranges. Also, the computational burden increases when utilizing commercial solvers to address the RMP and to find an integer solution of the proposed problem in large-scale instances.

In the future, developing strategies to integrate all pricing problems into a single problem that can be addressed simultaneously would be a valuable research direction. Additionally, designing parallel heuristic algorithms to achieve more accurate baselines in the policy gradient approach represents a promising research direction, as it facilitates faster convergence and enhances stability. Moreover, exploring a strategy for selecting key samples for replay in the actor-critic algorithm is a valuable area for future research, as it can enhance data utilization while reducing computational overhead. Furthermore, developing improved strategies for large-scale scenario would be an important research focus.

CRediT authorship contribution statement

Zongcheng Zhang: Investigation, Methodology, Software. **Yangsheng Jiang:** Investigation, Supervision, Writing – review & editing. **Mujin Gao:** Investigation, Methodology, Formal analysis, Writing – original draft. **Yanru Chen:** Conceptualization, Methodology, Writing – review & editing, Funding acquisition. **M.I.M. Wahab:** Investigation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Nature Science Foundation of China [grant number 72371206].

Appendix A. . The framework of the community-detection algorithm

The framework of community-detection algorithm is shown in Algorithm A.

Algorithm A. Community-detection algorithm

Input: Graph $G = (I, \Xi)$.
Output: Communities.
1. Let S be the growing community, Υ be the set of expandable prospective nodes compatible with S, A be the set of nodes to be allocated, Φ be the set of resulting communities, $\Re$ be the maximum number of resulting communities, and U_i be the set of compatible items with item i.
2. $A = \Upsilon \leftarrow I, S \leftarrow \varnothing$.
3. **While** $|\Phi| \leq \Re$ **do**
4. $\mathbb{Z} \leftarrow$ the set of inter-compatible γ nodes with the largest degree in Set A.
5. $A \leftarrow \Upsilon \backslash \bigcup_{i \in \mathbb{Z}} (\{i\} \cup U_i)$.
6. **If** $A \neq \varnothing$ **then**
7. $Stack \leftarrow (S, A, \Upsilon \cap (\bigcup_{i \in A} (\{i\} \cup U_i)))$.
8. **End if**
9. $S \leftarrow S \cup \mathbb{Z}$.
10. $\Upsilon \leftarrow \Upsilon \backslash A \backslash \mathbb{Z}$.
11. **If Rule A.3** does not hold **then**
12. $A \leftarrow \Upsilon$.
13. **Else**
14. **If Rule A.1** and **Rule A.2** holds **then**
15. Community S is merged.
16. **Else**
17. $\Phi \leftarrow \Phi \cup S$.
18. **End if**
19. **If** $Stack = \varnothing$ **then**
20. Return Φ.
21. **Else**
22. $(S, A, \Upsilon) \leftarrow$ Popped from the end of $Stack$.
23. **End if**
24. **End if**
25. **End While**
26. Return Φ.

When given a compatibility graph $G = (I, \Xi)$, the number of γ nodes with the highest degree is chosen in the community-detection algorithm. This selection is based on the understanding that nodes with higher degrees have more compatible nodes, accelerating the community generation process. These selected nodes are then added to the growing community S. Subsequently, from the nodes compatible with S, we further select nodes with the highest degrees and incorporate them into S. This process continues until there are no more compatible nodes remaining. At this point, a new community S is formed.

At each iteration, set A, set Υ, and the growing community S are jointly added to a stack. After a community is generated, the tuple (S, A,Υ) is popped from the end of the stack, and the new community is generated. A stack is last-in-first-out; the last nodes added to the stack are more likely to be compatible with the current nodes. Thus, they are more likely to merge to reduce the size of the community.

Algorithm A introduces the following three rules to minimize community overlap and reduce the overall number of communities.

Rule A.1. Community ζ merges with community ζ', if.

$$|\zeta| \leq \varsigma_1 \text{ , } and \, |\zeta'| \leq \varsigma_1 \tag{A-1}$$

where ζ indicates the current growing community. ζ' denotes the newly added community to set Φ. Rule A.1 shows that once the number of nodes in both communities falls below a specific threshold, ς_1, they will ultimately merge into a single community.

Rule A.2. Community ζ merges with community ζ', if

$$\max\{|\zeta \cap \zeta'|/|\zeta|, |\zeta \cap \zeta'|/|\zeta'|\} \geq \varsigma_2 \tag{A-2}$$

Rule A.2 indicates that when the degree of overlap between two communities, ζ and ζ', exceeds a defined threshold, ς_2, they are required to merge.

Rule A.3. Set Ξ and community ζ merge together, if

$$|\Xi| \leq \varsigma_3 \tag{A-3}$$

where set Ξ is formed by the compatible items associated with each item in community ζ. Rule A.3 states that when the number of items in set Ξ falls below a predefined threshold, ς_3, set Ξ is merged with the current growing community, ζ.

For a better understanding of the community-detection algorithm, an example is shown in Fig. A.1, Fig. A.2, and Fig.A.3. In Fig. A.1, a graph and its resulting communities are displayed.

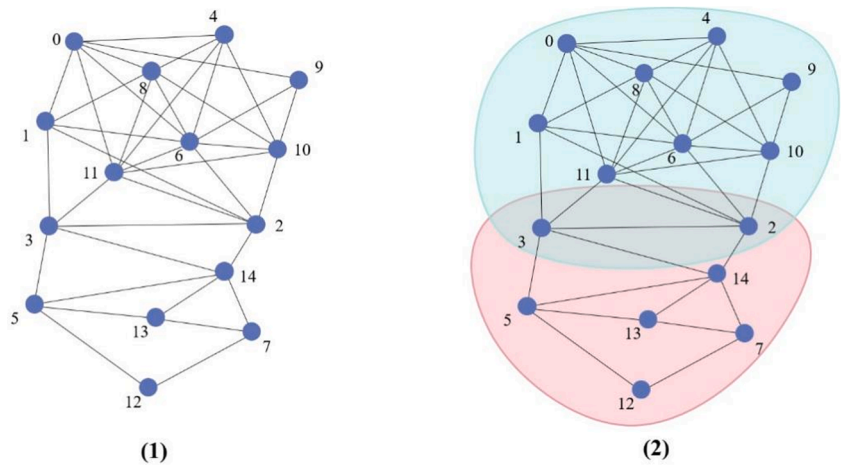


Fig. A.1. . An example of a graph and the resulting communities

The graph $G = (I, \Xi)$ of the example consists of 15 items in Fig. A.1 (1). A line between two vertices represents that two items are compatible with each other. After the community-detection algorithm, two communities are obtained in Fig. A.1 (2). The community generation process is demonstrated in Fig. A.2.

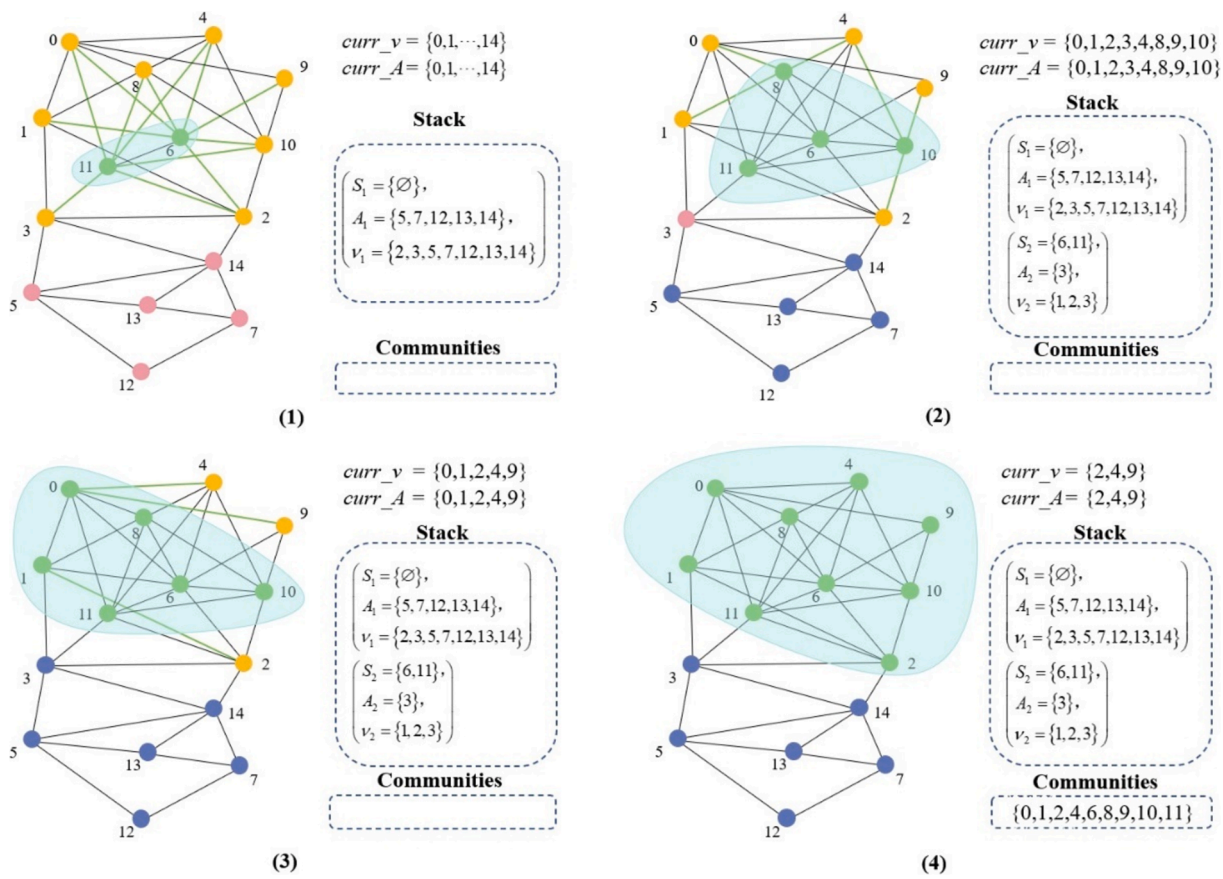


Fig. A.2. . An example of a community detection process.

In Fig. A.2 (1), nodes 6 and 11, with the largest degree in all nodes, are selected. Thus, $\mathbb{Z}_1 = \{6, 11\}$. The growing community $S = \{6, 11\}$ is shown as a green block. The tuple (S_1, A_1, ν_1) is added to the stack. The tuple is updated as shown in Eq. (A-4) to Eq. (A-6).

$$\begin{aligned} A_1 &= curr_\nu \setminus \bigcup_{i \in Z_1} (i \cup u(i)) \\ &= \{0, 1 \cdots 14\} \setminus \bigcup_{i \in \{6,11\}} (i \cup u(i)) \\ &= \{0, 1 \cdots 14\} \setminus \{0, 1, 2, 3, 4, 6, 8, 9, 10, 11\} \\ &= \{5, 7, 12, 13, 14\} \end{aligned} \tag{A-4}$$

$$\begin{aligned}\nu_1 &= curr_\nu \cap \bigcup_{i\in A_1}(i \cup u(i))\\ &= \{0,1\cdots 14\} \cap \bigcup_{i\in\{5,7,12,13,14\}}(i \cup u(i))\\ &= \{0,1\cdots 14\} \cap \{2,3,5,7,12,13,14\}\\ &= \{2,3,5,7,12,13,14\}\end{aligned} \tag{A-5}$$

$$\begin{aligned}curr_\nu' &= curr_\nu \backslash A_1 \backslash Z_1\\ &= \{0,1\cdots 14\}\backslash\{5,7,12,13,14\}\backslash\{6,11\}\\ &= \{0,1,2,3,4,8,9,10\}\end{aligned} \tag{A-6}$$

In Fig. A.2 (2), nodes 8 and 10, with the largest degree, are selected in the current set A. Thus, $\mathbb{Z}_2 = \{8,10\}$. The growing community S is updated to $\{6,8,10,11\}$. The sets are updated as follows.

$$curr_\nu \leftarrow curr_\nu' \tag{A-7}$$

$$curr_A = curr_\nu = \{0,1,2,3,4,8,9,10\} \tag{A-8}$$

$$\begin{aligned}A_2 &= curr_\nu \backslash \bigcup_{i\in Z_2}(i \cup u(i))\\ &= \{0,1,2,3,4,8,9,10\}\backslash \bigcup_{i\in\{8,10\}}(i \cup u(i))\\ &= \{0,1,2,3,4,8,9,10\}\backslash\{0,1,2,4,8,9,10\}\\ &= \{3\}\end{aligned} \tag{A-9}$$

$$\begin{aligned}\nu_2 &= curr_\nu \cap \bigcup_{i\in A_2}(i \cup u(i))\\ &= \{0,1,2,3,4,8,9,10\} \cap \bigcup_{i\in\{3\}}(i \cup u(i))\\ &= \{0,1,2,3,4,8,9,10\} \cap \{1,2,3,5,11,14\}\\ &= \{1,2,3\}\end{aligned} \tag{A-10}$$

$$S_2 = \{6,11\} \tag{A-11}$$

In Fig. A.2 (3), nodes 0 and 1, with the largest degree, are selected in current set A. The growing community S is updated to $\{0,1,6,8,10,11\}$. Nodes 2, 4, and 9 are merged with S according to Rule A.3. Then, a resulting community $S = \{0,1,2,4,6,8,9,10,11\}$is generated in Fig. A.2 (4).

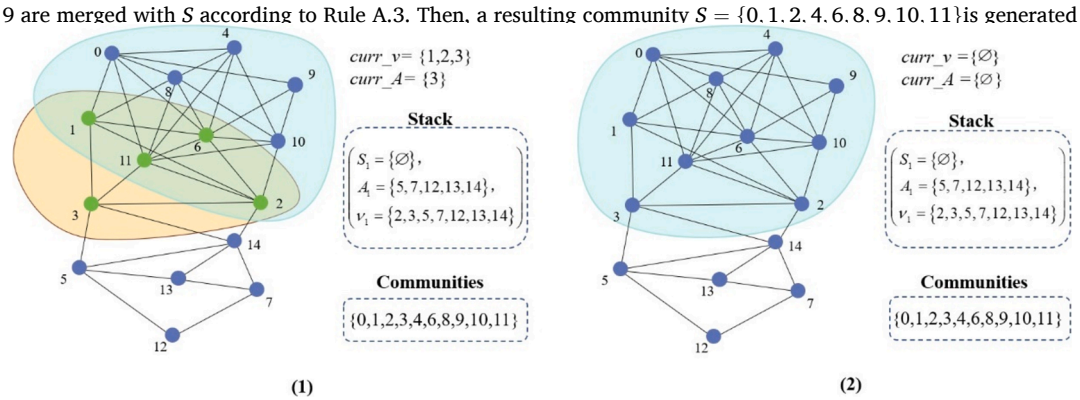


Fig. A.3. . An example of community reduction.

Fig. A.3 illustrates the process of community reduction. In Fig. A.3 (1), the tuple (S_1, A_1, ν_1) is popped from the end of the stack. A new community $S' = \{1,2,3,6,11\}$ is generated. According to Rule A.2, the community S' is merged with S. The resulting merged community $S \cup S' = \{0,1,2,3,4,6,8,9,10,11\}$ is shown in Fig. A.3 (2).

Appendix B. . Metrics of community-detection algorithm and maximal clique

.

Table B1
Comparative result of CD algorithm and maximal clique.

n	Obtained Number of communities(cliques)		Conflict density		Overlaps between communities(cliques)		Computation time	
	CD	maximal clique	CD	maximal clique	CD	maximal clique	CD	maximal clique
100	31.32	362573.19	0.03	0	3.90 %	9.73 %	0.01	3.14
200	73.28	848347.85	0.05	0	4.60 %	11.50 %	0.09	8.10
300	108.19	1050084.64	0.07	0	5.02 %	13.79 %	0.25	10.49
Average	70.93	753668.56	0.05	0	4.51 %	11.67 %	0.12	7.24

Table B demonstrates that the maximal clique technique yields significantly more cliques than the communities identified by the CD algorithm. This results in a computational burden during the subsequent label extension phase. Additionally, the maximal clique technique leads to greater overlaps compared to the CD algorithm, causing an increase in duplicate columns and thereby slowing down the solution process. While the conflict

density of the CD algorithm is only slightly higher than that of the maximal clique technique, the computation time for the maximal clique approach is substantially longer. These results indicate that the proposed CD algorithm effectively balances the trade-off between conflict density, potential overlaps among communities, and the computational burden associated with managing conflict issues.

Appendix C. . Percentage of columns with negative reduced costs contain items with large dual values

.

Table C1
Percentage of the columns with negative reduced costs contain items with large dual values of the first set.

type	n	Percentage
Linear	100	98.42 %
	200	99.28 %
	500	99.70 %
	1000	99.83 %
	2000	99.89 %
Concave	100	98.47 %
	200	99.39 %
	500	99.79 %
	1000	99.93 %
	2000	99.97 %
Convex	100	97.47 %
	200	98.12 %
	500	98.76 %
	1000	99.16 %
	2000	99.29 %
Total		99.16 %

Appendix D. . The framework of the decomposed label-setting algorithm

Algorithm D. Decomposed label-setting algorithm

Let U be the set of columns with a negative reduced cost.
Let Γ be the set of labels.
Let A be the set of nodes that meet Rule 1.
$U \leftarrow \varnothing$.
While $|A| \neq 0$ and $|U| \leq 2 \bullet |I|$ **do**
  $r = A[0]$.
  $A \leftarrow A \backslash r$.
  $\Gamma \leftarrow \{(w_r, v_r, c - \pi_r, \{r\}, \psi_r)\}$.
  While $|\Gamma| > 0$ and $|U| \leq 2 \bullet |I|$ **do**
    $L_{min} = \underset{L_i}{argmin}\{L_i^c, L_i \in \Gamma\}$.
    $\Gamma \leftarrow \Gamma \backslash L_{min}$.
    If L_{min} dominate L_i, where $L_i \in \Gamma$ **then**
    $\Gamma \leftarrow \Gamma \backslash L_i$.
  Else
    For each $\partial \in \psi_r$ **then**
      If ∂ equals the dummy item **then**
        If $L_{min}^c < 0$ **then**
          Add the corresponding column of L_{min} to U.
        End if
      Else
        If L_{min} extends to ∂ yielding a feasible packing pattern **then**
          Extend L_{min} to generate a new label L_∂.
          $\Gamma \leftarrow \Gamma \cup L_\partial$.
      End if
    End if
  End for
End if
End While
End While
Return U.

Appendix E. . Integrality of LP solutions

We randomly select a set of instances to examine the number of integer solutions in the LP relaxation of the RMP upon termination of the CG process. The results are presented in Table E.

Table E1
Ratio of integer solutions for instances of the first set.

type	n	Ratio of integer solutions	Percentage
Linear	100	55/72	76.39 %
	200	40/72	55.56 %
	500	33/72	45.83 %
	1000	32/72	44.44 %
	2000	21/72	29.17 %
Concave	100	49/72	68.06 %
	200	38/72	52.78 %
	500	30/72	41.67 %
	1000	29/72	40.28 %
	2000	9/72	12.50 %
Convex	100	72/72	100.00 %
	200	65/72	90.28 %
	500	59/72	81.94 %
	1000	57/72	79.17 %
	2000	37/72	51.39 %
Total		626/1080	57.96 %

Table E displays the ratio of integer solutions, which is determined by dividing the number of integer solutions by the total number of instances. The findings indicate that the DRL-CG algorithm achieves 57.96 % of integer solutions in the LP relaxation of the RMP for instances in the first set. Notably, for instances with the convex cost function, the CG process results in 80.56 % of integer solutions. Consequently, only a few instances necessitate the solver to compute integer solutions.

Appendix F. . Parameters in the DRLCG algorithm

The detailed configuration settings by the IRACE are shown in Table F.

Table F1
Elite configurations by the IRACE.

Parameter	Type	Range	Best configuration
$\Re$	Categorical	[512, 1024, 2048]	1024
γ	Integer	[1–10]	5
ς_1	Integer	[1–10]	2
ς_2	Categorical	[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]	0.9
ς_3	Categorical	[10, 20, 30, 40, 50]	20
θ	Categorical	[0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]	0.25

Appendix G. . Performance of the DRLCG

From Tables G.1 to G.7, the column labeled "class" indicates the specific set of instances being considered. The column labeled "type" denotes the type of cost function being used. The column labeled "n" represents the number of items in each instance. The column labeled "cost" displays the average cost determined by each algorithm for the given set of instances. The column labeled "$t(s)$" shows the average computational time required for the set of instances. The column labeled "#opt" reports the number of optimal solutions for the set of instances. It contains the proved optimal solutions, including the optimal solutions due to the integer solution on the root node, and optimal solutions that are the same as those obtained by the GUROBI solver, but does not cover some unverified optimal solutions. Lastly, the column labeled "Gap (%)" under "PH", "VNS","SSP", and "LNS" indicates the difference in cost between the current approach and the DRLCG algorithm, expressed as a percentage.

Each row of data in Tables G.1 to G.3 corresponds to the average performance across 72 instances. These instances include eight types of time windows and nine types of conflict graph density. Tables G.4 and G.5 provide the comparative average results based on different time windows for instances of the first and second sets. Tables G.6 and G.7 provide the comparative average results based on conflict graph density for instances of the first and second sets.

Table G1
Comparison of results with GUROBI for small-scale instances of two sets.

class	type	n	GUROBI			DRLCG		
			Cost	t (s)	#opt	Cost	t (s)	#opt
Set 1	Linear	100	16339.72	2244.18	36	16284.72	0.79	62
	Concave	100	11229.68	2263.01	33	11186.69	0.87	57
	Convex	100	24308.57	2508.77	27	23801.54	1.14	72
Set 2	Linear	100	4336.94	2988.85	16	4228.06	11.45	24
	Concave	100	4399.11	2890.24	21	4251.13	29.26	27
	Convex	100	3788.74	3014.77	16	3773.51	6.11	26
Average			10733.79	2651.64	–-	10587.61	8.27	–-
Sum			–-	–-	149	–-	–-	268

Table G2
Comparison of results for instances of the first set.

type	n	PH			VNS			SSP			LNS			DRL-CG	
		Cost	Gap (%)	t (s)	Cost	Gap (%)	t (s)	Cost	Gap(%)	t(s)	Cost	Gap(%)	t(s)	Cost	t (s)
Linear	100	16415.00	0.80 %	3.65	16865.14	3.56 %	1.10	17313.06	6.31 %	0.79	16396.11	0.68 %	1.35	16284.72	0.79
	200	31674.44	0.54 %	21.27	36752.08	16.66 %	3.04	33460.69	6.21 %	3.02	31898.19	1.25 %	7.54	31504.44	3.02
	500	77554.31	2.11 %	238.62	87845.00	15.66 %	26.75	80333.75	5.77 %	26.70	77500.14	2.04 %	82.88	75948.89	26.69
	1000	152267.64	3.78 %	1717.95	166869.03	13.74 %	281.51	155212.08	5.79 %	281.39	150393.06	2.51 %	638.81	146715.28	281.39
	2000	310530.83	6.60 %	3311.70	322906.25	10.85 %	1510.50	301972.78	3.66 %	1510.40	293532.36	0.76 %	3644.31	291304.58	1478.13
Concave	100	11187.58	0.01 %	12.40	11450.74	2.36 %	1.13	11706.19	4.64 %	0.87	11254.72	0.61 %	1.28	11186.69	0.87
	200	21342.15	0.01 %	81.20	23460.81	9.94 %	3.20	22306.14	4.53 %	3.19	21587.38	1.16 %	6.66	21340.07	3.19
	500	50859.29	0.46 %	731.50	55953.07	10.52 %	31.45	53193.90	5.07 %	31.40	51581.61	1.88 %	73.16	50627.39	31.40
	1000	100812.18	3.92 %	2387.64	106225.92	9.50 %	369.92	101939.76	5.09 %	369.83	99060.38	2.12 %	636.30	97005.63	361.32
	2000	202931.88	6.80 %	3468.43	205242.01	8.02 %	1414.87	197972.24	4.19 %	1414.68	191417.88	0.74 %	3036.82	190008.31	1386.43
Convex	100	23801.54	0.00 %	2.64	24926.83	4.73 %	1.39	25945.11	9.01 %	1.14	23951.50	0.63 %	2.10	23801.54	1.14
	200	46459.61	0.00 %	13.78	56994.96	22.68 %	3.34	49452.13	6.44 %	3.33	47039.68	1.25 %	12.43	46459.35	3.32
	500	113332.17	0.00 %	167.49	137408.24	21.24 %	32.10	119919.68	5.81 %	32.06	115350.50	1.78 %	162.95	113331.35	32.06
	1000	225585.88	2.19 %	1404.65	264966.76	20.03 %	203.81	233290.50	5.68 %	203.70	225515.89	2.16 %	1164.54	220751.75	203.70
	2000	475714.10	7.79 %	3131.15	511483.50	15.90 %	1543.08	456700.06	3.48 %	1542.97	445005.35	0.83 %	8388.87	441330.40	1511.84
Average		124031.24	4.66 %	1112.94	135290.02	14.16 %	361.81	124047.87	4.68 %	361.70	120098.98	1.34 %	1190.67	118506.69	355.02

Table G3
Comparison of results for instances of the second set.

type	n	PH			VNS			SSP			LNS			DRL-CG	
		Cost	Gap (%)	t (s)	Cost	Gap (%)	t (s)	Cost	Gap(%)	t(s)	Cost	Gap(%)	t(s)	Cost	t (s)
Linear	100	4231.81	0.09 %	99.83	4682.64	10.75 %	11.50	4547.08	7.55 %	12.00	4405.56	4.20 %	14.31	4228.06	11.45
	200	7825.56	0.10 %	1424.37	13448.33	72.02 %	66.58	8913.19	14.01 %	67.14	8256.53	5.61 %	84.53	7817.92	66.58
	300	12846.94	14.59 %	2429.22	18706.53	66.85 %	194.95	12662.92	12.94 %	195.45	11899.86	6.14 %	246.24	11211.67	194.95
Concave	100	4241.69	−0.22 %	120.34	4584.68	7.85 %	29.40	4813.19	13.22 %	29.81	4427.90	4.16 %	32.46	4251.13	29.26
	200	8285.93	10.61 %	1873.83	10365.26	38.37 %	115.44	8474.54	13.13 %	115.91	7855.51	4.86 %	134.19	7491.18	115.43
	300	12226.85	16.69 %	2556.80	13833.63	32.02 %	366.58	11565.76	10.38 %	367.15	11029.06	5.26 %	419.47	10478.24	366.57
Convex	100	3777.76	0.11 %	81.38	3994.68	5.86 %	6.11	4122.67	9.25 %	6.72	4122.22	9.24 %	9.13	3773.51	6.11
	200	8302.54	19.10 %	1199.39	16063.97	130.44 %	28.05	7968.43	14.31 %	28.58	7911.94	13.50 %	52.26	6971.11	28.05
	300	17465.29	75.07 %	2199.39	25492.75	155.54 %	100.75	11706.19	17.34 %	101.38	11515.68	15.43 %	174.16	9975.99	100.75
Average		8800.49	19.65 %	1331.62	12352.50	67.94 %	102.15	8308.22	12.95 %	102.68	7936.03	7.89 %	129.64	7355.42	102.13

Table G4
Comparative average results based on different time windows in the first set.

type	n	len = 1		len = 2		len = 4		len = 8		len = 16		len = 32		len = 64		len = 128	
		Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)
Linear	100	17775.56	0.50	17263.33	0.54	16704.44	0.57	16484.44	0.65	16377.78	0.75	15415.56	0.94	15294.44	1.11	14962.22	1.26
	200	34590.00	1.22	33871.11	1.47	32573.33	2.00	31533.33	2.51	30897.78	3.37	29643.33	3.59	29538.89	4.44	29387.78	5.56
	500	82800.00	9.13	81475.56	11.72	80264.44	15.83	76032.22	21.95	74518.89	30.58	72417.78	33.34	69777.78	40.81	70304.44	50.19
	1000	159828.89	63.74	156107.78	93.08	152675.56	95.80	147834.44	115.43	142915.56	129.96	139480.00	201.52	138651.11	520.48	136228.89	1031.09
	2000	312212.22	606.68	304725.56	773.63	296266.67	978.37	287644.44	1175.63	281377.78	1694.63	285297.78	2024.99	279471.11	2142.29	283441.11	2428.82
Concave	100	12647.11	0.47	12160.78	0.54	11759.89	0.63	11365.00	1.12	11141.78	0.74	10385.67	0.96	10128.33	1.05	9905.00	1.46
	200	24350.56	1.27	23597.44	1.45	22364.78	1.94	21406.11	2.42	20697.67	3.26	19666.78	3.96	19432.22	5.33	19205.00	5.87
	500	57144.56	9.27	55588.22	11.59	54170.22	15.16	50639.33	26.96	49103.89	25.80	47281.56	37.35	45323.22	45.64	45768.11	79.39
	1000	108596.89	63.11	104991.89	90.56	101658.00	84.03	97430.78	100.27	93293.00	150.85	90476.33	226.28	89808.44	559.91	89789.67	1615.55
	2000	209574.33	358.78	202297.56	633.62	194949.89	650.20	185931.11	925.93	185030.78	1817.14	181508.89	1921.26	179272.00	2196.03	181501.89	2588.49
Convex	100	25386.44	0.46	24772.78	0.54	23990.89	0.65	23976.44	0.77	24116.11	0.85	22772.22	1.49	22934.67	2.74	22462.78	1.63
	200	49722.11	1.23	48996.33	1.51	47386.67	2.29	46425.67	2.57	45824.44	3.81	44224.78	4.41	44612.89	5.17	44481.89	5.61
	500	120264.78	10.55	119464.44	11.60	118817.44	15.30	113404.22	26.11	112086.22	29.98	109556.22	47.91	106206.89	48.89	106850.56	66.11
	1000	234748.33	61.69	231285.00	77.69	227973.33	110.49	222555.33	155.77	216286.11	168.95	212424.78	244.23	211877.89	286.95	208863.22	523.79
	2000	463486.22	489.35	456036.11	691.88	446126.89	844.50	432441.33	1068.87	425172.89	1676.43	425390.00	2072.53	439523.33	2624.26	442466.44	2626.87
Average		127541.87	111.83	124842.26	160.09	121845.50	187.85	117673.61	241.80	115256.04	382.47	113729.44	454.98	113456.88	565.67	113707.93	735.45

Table G5
Comparative average results based on different time windows in the second set.

type	n	len = 1		len = 2		len = 4		len = 8		len = 16		len = 32		len = 64		len = 128	
		Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)
Linear	100	5544.44	1.31	5090.00	2.03	4590.00	2.79	4223.33	5.38	3854.44	9.44	3685.56	12.69	3436.67	18.61	3400.00	39.38
	200	9815.56	7.15	9033.33	10.57	8345.56	22.77	7772.22	50.67	7312.22	75.42	6915.56	97.09	6747.78	119.15	6601.11	149.82
	300	13683.33	26.36	12734.44	53.02	11814.44	110.62	11184.44	144.06	10517.78	159.84	10115.56	247.56	9910.00	266.34	9733.33	551.82
Concave	100	6310.89	0.71	5661.56	1.14	4910.00	2.81	4245.11	6.22	3653.00	22.86	3323.44	53.31	3004.11	72.14	2900.89	74.87
	200	10669.56	4.99	9547.56	14.14	8358.56	41.61	7389.78	82.34	6610.89	121.67	6056.22	180.83	5786.67	220.30	5510.22	257.56
	300	14528.56	29.03	12904.33	75.66	11339.78	162.71	10359.00	232.02	9417.67	283.18	8769.67	584.54	8373.89	723.66	8133.00	841.73
Convex	100	4879.56	1.63	4521.11	2.13	4133.89	3.08	3835.78	4.15	3484.89	5.48	3298.89	7.14	3030.78	12.00	3003.22	13.26
	200	8790.56	6.21	8100.89	8.36	7574.56	13.33	6960.22	14.20	6534.33	17.95	6120.00	37.52	5932.78	59.50	5755.56	67.30
	300	12318.78	18.09	11508.22	26.77	10686.11	32.35	10047.00	47.15	9308.22	61.51	8869.33	139.79	8633.11	182.05	8437.11	298.26
Average		9615.69	10.61	8789.05	21.54	7972.54	43.56	7335.21	65.13	6743.72	84.15	6350.47	151.16	6095.09	185.97	5941.60	254.89

Table G6
Comparative average results based on different conflict graph densities in the first set.

type	n	d = 0.1		d = 0.2		d = 0.3		d = 0.4		d = 0.5		d = 0.6		d = 0.7		d = 0.8		d = 0.9	
		Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)
Linear	100	15582.50	1.10	15488.75	1.04	15842.50	0.85	15672.50	0.90	16132.50	0.79	16686.25	0.72	16261.25	0.68	17270.00	0.55	17626.25	0.48
	200	29482.50	4.33	29717.50	4.49	30503.75	3.57	29912.50	4.23	30896.25	2.89	31555.00	2.62	32608.75	2.10	33387.50	1.77	35476.25	1.18
	500	70993.75	58.01	72096.25	35.95	72278.75	31.37	72671.25	35.87	74275.00	23.18	76251.25	22.14	78062.50	16.25	81120.00	10.51	85791.25	6.97
	1000	138388.75	603.24	139033.75	596.05	139665.00	354.26	141366.25	315.17	143903.75	331.26	146446.25	118.57	149823.75	103.98	155367.50	71.74	166442.5	38.21
	2000	281378.75	2406.58	285905.00	2670.87	289275.00	2413.41	283470.00	1816.64	280973.75	1337.08	281861.25	1363.63	292757.50	574.45	301306.25	454.53	324813.75	265.96
Concave	100	10369.00	1.23	10442.50	1.04	10622.13	0.97	10525.50	1.05	10939.00	0.84	11480.63	0.74	11341.75	0.85	12120.00	0.62	12839.75	0.48
	200	19463.13	5.28	19636.00	4.16	20183.50	4.15	19877.00	3.69	20696.38	2.84	21311.13	3.06	22301.88	2.43	23245.13	1.74	25346.5	1.35
	500	46298.50	53.77	47116.88	77.13	47300.63	31.51	47674.50	36.11	48932.88	26.97	50446.63	19.60	52338.25	18.94	55360.25	11.25	60,178	7.29
	1000	90718.13	1078.50	90162.50	618.13	91852.13	617.92	91799.88	401.93	93908.38	239.86	95900.63	109.98	98989.13	85.20	104380.88	65.11	115,339	35.24
	2000	182741.00	2303.45	183482.88	2300.03	184262.00	2075.79	182551.63	1906.41	181409.75	1303.86	183109.63	1271.26	191247.75	533.23	199436.75	507.83	221833.375	276.03
Convex	100	23237.88	1.43	22905.13	1.67	23524.00	1.38	23259.25	1.43	23740.63	1.23	24337.63	1.00	23530.00	0.79	24896.38	0.73	24,783	0.59
	200	44169.13	5.24	44415.88	4.79	45740.88	3.72	44611.25	4.45	46020.63	3.49	46579.38	2.48	47767.25	2.39	48272.63	2.02	50557.125	1.31
	500	107567.00	52.77	109065.50	53.47	109245.75	41.52	109580.13	37.62	111760.00	34.49	114172.63	25.18	115946.13	23.32	119059.50	12.71	123585.5	7.44
	1000	211241.75	307.58	211825.88	393.36	212709.63	335.99	215107.25	232.45	218261.00	177.93	221152.00	148.93	224826.38	119.52	230187.50	70.52	241454.375	46.98
	2000	429246.25	2316.47	432240.75	2376.70	437150.38	2051.15	442625.38	1937.19	430815.88	1561.03	428413.13	1408.37	443324.25	971.68	451627.25	619.79	476530.375	364.15
Average		113391.87	613.27	114235.68	609.26	115343.73	531.17	115380.28	449.01	115511.05	336.52	116646.89	299.89	120075.10	163.72	123802.50	122.09	132173.13	70.24

Table G7
Comparative average results based on different conflict graph densities in the second set.

type	n	d = 0.1		d = 0.2		d = 0.3		d = 0.4		d = 0.5		d = 0.6		d = 0.7		d = 0.8		d = 0.9	
		Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)	Cost	t (s)
Linear	100	3437.50	27.05	3550.00	23.15	3596.25	17.83	3712.50	13.36	3870.00	10.81	4147.50	5.13	4562.50	3.04	5196.25	1.92	5980	0.81
	200	6663.75	111.38	6678.75	115.56	6738.75	105.21	6848.75	87.49	7121.25	88.89	7493.75	58.64	8260.00	20.02	9297.50	9.26	11258.75	2.77
	300	9712.50	210.97	9776.25	423.94	9795.00	316.01	9931.25	187.04	10130.00	280.71	10718.75	209.66	11675.00	89.92	13128.75	28.57	16037.5	7.73
Concave	100	2741.13	58.66	2932.88	68.30	3062.75	53.12	3394.13	41.21	3765.38	29.87	4358.63	7.66	5051.75	2.74	5937.50	1.25	7016	0.50
	200	4875.63	219.55	5042.50	190.87	5421.75	181.28	5778.13	141.69	6501.25	153.22	7467.75	93.60	8762.25	47.75	10464.75	8.72	13106.625	2.18
	300	6914.00	523.32	7167.62	771.40	7467.00	568.46	8120.00	395.22	9032.50	527.82	10348.75	297.85	12186.25	166.80	14497.75	40.39	18570.25	7.86
Convex	100	3159.38	11.92	3253.00	10.84	3283.88	8.71	3388.13	7.43	3462.75	5.50	3682.38	4.32	4019.63	2.95	4555.13	2.16	5157.375	1.14
	200	6061.00	49.51	6061.50	57.14	6080.87	40.31	6214.63	40.01	6405.00	28.37	6684.88	15.58	7324.25	11.71	8141.88	6.54	9766	3.24
	300	8726.88	196.44	8836.13	180.00	8826.25	169.12	8861.25	116.33	9071.00	119.49	9599.63	70.07	10344.38	30.15	11545.25	17.73	13973.125	7.39
Average		5810.19	156.53	5922.07	204.58	6030.28	162.23	6249.86	114.42	6595.46	138.30	7166.89	84.72	8020.67	41.67	9196.08	12.95	11207.29167	3.73

Table G8
Comparison of results for instances of the VSBPP.

n	DRLCG		GA	
	Cost	*t*(s)	Cost	*t*(s)
25	1238.00	0.52	1244.00	0.53
50	2503.00	3.13	2505.00	3.14
100	4978.00	13.99	4979.00	14.01
200	10215.00	55.22	10218.00	55.24
500	25419.00	391.89	25438.00	391.94
Average	8870.60	92.95	8876.80	92.97

Table G9
Comparison of results for instances of the VSBPPTW.

type	*n*	DRLCG		PH	
		Cost	#opt	Cost	#opt
Linear	60	5001.00	9	5000.00	10
	120	10000.00	10	10000.00	10
	180	15000.00	10	15005.00	6
	240	20000.00	10	20016.00	1
	300	25001.00	9	25018.00	0
Concave	60	3180.00	10	3180.00	10
	120	6360.00	10	6360.00	10
	180	9554.70	8	9540.00	10
	240	12778.20	2	12726.30	9
	300	15966.30	2	15943.40	4
Convex	60	1180.00	10	1180.00	10
	120	2360.00	10	2360.00	10
	180	3540.00	10	3563.60	6
	240	4724.10	9	4775.40	0
	300	5908.20	8	5957.20	0
Average		9370.23	–-	9374.99	–-
Sum		–-	127	–-	96

Appendix H. . Ablation analysis

.

Table H1
Comparison of results between DRLCG with and without heuristic pricing for instances of the first set.

type	*n*	DRLCG		DRLCG without heuristic pricing		
		Cost	*t (s)*	Cost	Gap (%)	*t (s)*
Linear	100	16284.72	0.79	16284.72	0.00 %	1.17
	200	31504.44	3.02	31504.44	0.00 %	5.71
	500	75948.89	26.69	75948.89	0.00 %	64.37
	1000	146715.28	281.39	147724.72	0.69 %	671.28
	2000	291304.58	1478.13	295787.50	1.54 %	2410.04
Concave	100	11186.69	0.87	11186.76	0.00 %	1.46
	200	21340.07	3.19	21340.18	0.00 %	6.45
	500	50627.39	31.40	50627.43	0.00 %	74.45
	1000	97005.63	361.32	97474.46	0.48 %	711.63
	2000	190008.31	1386.43	192844.07	1.49 %	2387.76
Convex	100	23801.54	1.14	23801.54	0.00 %	1.20
	200	46459.35	3.32	46459.35	0.00 %	5.48
	500	113331.35	32.06	113331.31	0.00 %	59.49
	1000	220751.75	203.70	221495.31	0.34 %	492.35
	2000	441330.40	1511.84	452675.90	2.57 %	2240.71
Average		118506.69	355.02	119899.11	1.17 %	608.90

Table H2
Comparison of results between DRLCG with and without heuristic pricing for instances of the second set.

type	n	DRLCG		DRLCG without heuristic pricing		
		Cost	t (s)	Cost	Gap (%)	t (s)
Linear	100	4228.06	11.45	4226.81	−0.03 %	8.76
	200	7817.92	66.58	7812.92	−0.06 %	75.00
	300	11211.67	194.95	11204.86	−0.06 %	232.64
Concave	100	4251.13	29.26	4255.19	0.10 %	32.73
	200	7491.18	115.43	7497.43	0.08 %	143.04
	300	10478.24	366.57	10478.69	0.00 %	467.54
Convex	100	3773.51	6.11	3773.54	0.00 %	5.08
	200	6971.11	28.05	6971.29	0.00 %	29.63
	300	9975.99	100.75	9976.28	0.00 %	100.94
Average		7355.42	102.13	7355.22	0.00 %	121.71

Table H3
Comparison of results between DRLCG with and without tabu strategy for instances of the first set.

type	n	DRLCG		DRLCG without tabu strategy for the pricing problem		
		Cost	t (s)	Cost	Gap (%)	t (s)
Linear	100	16284.72	0.79	16284.72	0.00 %	0.91
	200	31504.44	3.02	31504.44	0.00 %	3.81
	500	75948.89	26.69	75948.89	0.00 %	42.87
	1000	146715.28	281.39	146891.25	0.12 %	403.96
	2000	291304.58	1478.13	292316.81	0.35 %	1853.44
Concave	100	11186.69	0.87	11186.69	0.00 %	1.02
	200	21340.07	3.19	21340.07	0.00 %	4.43
	500	50627.39	31.40	50627.50	0.00 %	42.77
	1000	97005.63	361.32	96977.69	−0.03 %	445.62
	2000	190008.31	1386.43	191335.08	0.70 %	1773.59
Convex	100	23801.54	1.14	23801.54	0.00 %	0.90
	200	46459.35	3.32	46459.35	0.00 %	3.49
	500	113331.35	32.06	113331.35	0.00 %	38.36
	1000	220751.75	203.70	220751.67	0.00 %	249.38
	2000	441330.40	1511.84	443396.60	0.47 %	1622.91
Average		118506.69	355.02	118810.24	0.26 %	432.50

Table H4
Comparison of results between DRLCG with and without tabu strategy for instances of the second set.

type	n	DRLCG		DRLCG without tabu strategy for the pricing problem		
		Cost	t (s)	Cost	Gap (%)	t (s)
Linear	100	4228.06	11.45	4227.78	−0.01 %	11.45
	200	7817.92	66.58	7820.97	0.04 %	72.65
	300	11211.67	194.95	11260.42	0.43 %	198.10
Concave	100	4251.13	29.26	4255.49	0.10 %	30.73
	200	7491.18	115.43	7495.93	0.06 %	129.20
	300	10478.24	366.57	10484.51	0.06 %	408.97
Convex	100	3773.51	6.11	3773.51	0.00 %	6.35
	200	6971.11	28.05	6971.11	0.00 %	30.87
	300	9975.99	100.75	9975.74	0.00 %	108.39
Average		7355.42	102.13	7362.83	0.10 %	110.75

Table H5
Comparison of results between DRLCG with and without the pre-processing strategy for instances of the first set.

type	n	DRLCG		DRLCG without pre-processing		
		Cost	t(s)	Cost	Gap(%)	t(s)
Linear	100	16284.72	0.79	16284.72	0.00 %	1.67
	200	31504.44	3.02	31504.44	0.00 %	7.86
	500	75948.89	26.69	75949.03	0.00 %	66.33
	1000	146715.28	281.39	146715.42	0.00 %	530.60
	2000	291304.58	1478.13	294829.86	1.21 %	2639.07
Concave	100	11186.69	0.87	11186.54	0.00 %	1.56
	200	21340.07	3.19	21340.07	0.00 %	6.62
	500	50627.39	31.40	50627.49	0.00 %	54.80

(continued on next page)

Table H5 (*continued*)

type	n	DRLCG		DRLCG without pre-processing		
		Cost	t(s)	Cost	Gap(%)	t(s)
	1000	97005.63	361.32	96977.79	−0.03 %	484.54
	2000	190008.31	1386.43	191366.79	0.71 %	2172.92
Convex	100	23801.54	1.14	23801.54	0.00 %	2.09
	200	46459.35	3.32	46459.35	0.00 %	10.51
	500	113331.35	32.06	113331.49	0.00 %	103.13
	1000	220751.75	203.70	220752.47	0.00 %	650.10
	2000	441330.40	1511.84	485308.92	9.96 %	3493.71
Average		118506.69	355.02	121762.39	2.75 %	681.70

Table H6
Comparison of results between DRLCG with and without the pre-processing strategy for instances of the second set.

type	n	DRLCG		DRLCG without pre-processing		
		Cost	t(s)	Cost	Gap(%)	t(s)
Linear	100	4228.06	11.45	4228.06	0.00 %	14.82
	200	7817.92	66.58	7827.22	0.12 %	82.77
	300	11211.67	194.95	11258.33	0.42 %	217.83
Concave	100	4251.13	29.26	4260.68	0.22 %	34.98
	200	7491.18	115.43	7493.39	0.03 %	151.25
	300	10478.24	366.57	10483.08	0.05 %	549.95
Convex	100	3773.51	6.11	3773.51	0.00 %	7.30
	200	6971.11	28.05	6971.39	0.00 %	32.68
	300	9975.99	100.75	9975.60	0.00 %	114.58
Average		7355.42	102.13	7363.47	0.11 %	141.52

Appendix I. . Performance of TRFF in the DRLCG

.

Table I1
Comparison of results between DRLCG with TRFF, DRLCG with BF, and DRLCG with SSP for instances of the first set.

type	n	DRLCG with TRFF		DRLCG with BF			DRLCG with SSP		
		Cost	t(s)	Cost	Gap(%)	t(s)	Cost	Gap(%)	t(s)
Linear	100	16284.72	0.79	16284.86	0.00 %	0.78	16284.72	0.00 %	0.89
	200	31504.44	3.02	31504.58	0.00 %	3.27	31504.03	0.00 %	3.75
	500	75948.89	26.69	75948.89	0.00 %	28.82	75948.89	0.00 %	34.73
	1000	146715.28	281.39	146715.14	0.00 %	257.49	146714.72	0.00 %	288.96
	2000	291304.58	1478.13	291449.17	0.05 %	1550.88	294637.78	1.14 %	1593.39
Concave	100	11186.69	0.87	11186.79	0.00 %	0.79	11185.72	−0.01 %	0.83
	200	21340.07	3.19	21340.25	0.00 %	3.09	21339.43	0.00 %	3.36
	500	50627.39	31.40	50627.72	0.00 %	30.34	50626.92	0.00 %	29.52
	1000	97005.63	361.32	96760.43	−0.25 %	320.01	97135.06	0.13 %	277.48
	2000	190008.31	1386.43	190318.17	0.16 %	1388.70	194100.42	2.15 %	1257.87
Convex	100	23801.54	1.14	23801.54	0.00 %	0.86	23801.54	0.00 %	1.03
	200	46459.35	3.32	46459.35	0.00 %	3.79	46459.35	0.00 %	4.11
	500	113331.35	32.06	113331.31	0.00 %	40.35	113331.51	0.00 %	39.96
	1000	220751.75	203.70	220751.72	0.00 %	278.11	220751.58	0.00 %	231.47
	2000	441330.40	1511.84	444122.19	0.63 %	2056.04	441516.85	0.04 %	1664.54
Average		118506.69	355.02	118706.81	0.17 %	397.55	119022.57	0.44 %	362.13

Table I2
Comparison of results between DRLCG with TRFF, DRLCG with BF, and DRLCG with SSP for instances of the second set.

type	n	DRLCG with TRFF		DRLCG with BF			DRLCG with SSP		
		Cost	t(s)	Cost	Gap(%)	t(s)	Cost	Gap(%)	t(s)
Linear	100	4228.06	11.45	4228.06	0.00 %	12.89	4228.06	0.00 %	12.14
	200	7817.92	66.58	7825.56	0.10 %	78.30	7820.56	0.03 %	71.00
	300	11211.67	194.95	11275.42	0.57 %	205.16	11223.33	0.10 %	210.61
Concave	100	4251.13	29.26	4256.67	0.13 %	32.78	4263.75	0.30 %	31.88
	200	7491.18	115.43	7499.94	0.12 %	135.86	7510.47	0.26 %	140.78
	300	10478.24	366.57	10487.64	0.09 %	547.40	10514.21	0.34 %	512.30
Convex	100	3773.51	6.11	3773.54	0.00 %	6.38	3773.76	0.01 %	5.69

(*continued on next page*)

Table I2 (*continued*)

type	n	DRLCG with TRFF		DRLCG with BF			DRLCG with SSP		
		Cost	t(s)	Cost	Gap(%)	t(s)	Cost	Gap(%)	t(s)
	200	6971.11	28.05	7048.26	1.11 %	28.79	6971.82	0.01 %	21.13
	300	9975.99	100.75	9975.85	0.00 %	102.42	9976.81	0.01 %	66.07
Average		7355.42	102.13	7374.55	0.26 %	127.78	7364.75	0.13 %	119.07

Appendix J. . Performance of CDAC in the DRLCG

.

Table J1
Comparison of results between DRLCG with CDAC and DRLCG with maximal clique-based actor-critic for instances of the first set.

type	n	DRLCG with CDAC		DRLCG with maximal clique-based actor-critic		
		Cost	t(s)	Cost	Gap(%)	t(s)
Linear	100	16284.72	0.79	16284.72	0.00 %	0.86
	200	31504.44	3.02	31504.44	0.00 %	3.26
	500	75948.89	26.69	75948.89	0.00 %	28.61
	1000	146715.28	281.39	146714.58	0.00 %	254.83
	2000	291304.58	1478.13	290984.72	−0.11 %	1411.19
Concave	100	11186.69	0.87	11186.69	0.00 %	0.88
	200	21340.07	3.19	21340.07	0.00 %	3.14
	500	50627.39	31.40	50627.39	0.00 %	31.54
	1000	97005.63	361.32	97003.24	0.00 %	356.46
	2000	190008.31	1386.43	190653.88	0.34 %	1379.35
Convex	100	23801.54	1.14	23801.54	0.00 %	0.89
	200	46459.35	3.32	46459.35	0.00 %	3.46
	500	113331.35	32.06	113331.35	0.00 %	35.69
	1000	220751.75	203.70	220751.83	0.00 %	230.30
	2000	441330.40	1511.84	445674.88	0.98 %	1785.32
Average		118506.69	355.02	118817.84	0.26 %	368.39

Table J2
Comparison of results between DRLCG with CDAC and DRLCG with maximal clique-based actor-critic for instances of the second set.

type	n	DRLCG with CDAC		DRLCG with maximal clique-based actor-critic		
		Cost	t(s)	Cost	Gap(%)	t(s)
Linear	100	4228.06	11.45	4227.916667	0.00 %	8.22071993
	200	7817.92	66.58	7816.805556	−0.01 %	81.31731097
	300	11211.67	194.95	11212.08333	0.00 %	215.2255071
Concave	100	4251.13	29.26	4253.569444	0.06 %	35.04978483
	200	7491.18	115.43	7490.388889	−0.01 %	132.2141785
	300	10478.24	366.57	10479.98611	0.02 %	431.1532413
Convex	100	3773.51	6.11	3773.513889	0.00 %	7.098649151
	200	6971.11	28.05	6971.361111	0.00 %	35.14705613
	300	9975.99	100.75	9975.736111	0.00 %	109.3695474
Average		7355.42	102.13	7355.71	0.00 %	117.20

Appendix K. . Performance of the DRL in the DRLCG

.

Table K1
Comparison of results between the DRLCG and DQNCG for instances of the first set.

type	n	DRLCG		DQNCG		
		Cost	t(s)	Cost	Gap(%)	t(s)
Linear	100	16284.72	0.79	16284.72	0.00 %	0.77
	200	31504.44	3.02	31504.44	0.00 %	3.30
	500	75948.89	26.69	75948.89	0.00 %	28.62
	1000	146715.28	281.39	146715.14	0.00 %	325.27
	2000	291304.58	1478.13	290680.69	−0.21 %	1527.81
Concave	100	11186.69	0.87	11186.69	0.00 %	0.82
	200	21340.07	3.19	21340.08	0.00 %	3.15

(*continued on next page*)

Table K1 (*continued*)

type	n	DRLCG		DQNCG		
		Cost	t(s)	Cost	Gap(%)	t(s)
	500	50627.39	31.40	50627.26	0.00 %	31.12
	1000	97005.63	361.32	96871.71	−0.14 %	336.08
	2000	190008.31	1386.43	189912.38	−0.05 %	1419.25
Convex	100	23801.54	1.14	23801.54	0.00 %	0.82
	200	46459.35	3.32	46459.35	0.00 %	3.13
	500	113331.35	32.06	113331.35	0.00 %	29.70
	1000	220751.75	203.70	220751.75	0.00 %	207.49
	2000	441330.40	1511.84	442642.56	0.30 %	1504.81
Average		118506.69	355.02	118537.24	0.03 %	361.48

Table K2
Comparison of results between DRLCG and DQNCG for instances of the second set.

type	n	DRLCG		DQNCG		
		Cost	t(s)	Cost	Gap(%)	t(s)
Linear	100	4228.06	11.45	4228.19	0.00 %	13.12
	200	7817.92	66.58	7817.08	−0.01 %	79.44
	300	11211.67	194.95	11211.94	0.00 %	208.64
Concave	100	4251.13	29.26	4251.35	0.01 %	37.40
	200	7491.18	115.43	7490.75	−0.01 %	126.76
	300	10478.24	366.57	10478.90	0.01 %	409.74
Convex	100	3773.51	6.11	3773.51	0.00 %	9.73
	200	6971.11	28.05	6971.47	0.01 %	33.98
	300	9975.99	100.75	9975.78	0.00 %	108.35
Average		7355.42	102.13	7355.44	0.00 %	114.13

Table K3
Comparison of results between DRLCG with CDAC, DRLCG with CDHM, and DRLCG with HM.

type	n	DRLCG with CDAC		DRLCG with CDHM			DRLCG with HM		
		Cost	t (s)	Cost	Gap(%)	t (s)	Cost	Gap(%)	t (s)
Linear	100	4228.06	11.45	4228.75	0.02 %	14.63	4228.75	0.02 %	15.99
	200	7817.92	66.58	7819.17	0.02 %	78.28	7819.31	0.02 %	78.62
	300	11211.67	194.95	11212.64	0.01 %	199.94	11213.19	0.01 %	210.36
Concave	100	4251.13	29.26	4254.99	0.09 %	33.01	4252.19	0.02 %	32.91
	200	7491.18	115.43	7493.11	0.03 %	151.64	7495.93	0.06 %	127.34
	300	10478.24	366.57	10477.85	0.00 %	404.58	10488.65	0.10 %	447.62
Convex	100	3773.51	6.11	3773.51	0.00 %	7.03	3773.51	0.00 %	7.03
	200	6971.11	28.05	6971.43	0.00 %	32.09	6971.25	0.00 %	31.69
	300	9975.99	100.75	9976.06	0.00 %	104.07	9975.67	0.00 %	102.09
Average		**7355.42**	**102.13**	**7356.39**	**0.02 %**	**113.92**	**7357.61**	**0.03 %**	**117.07**

Data availability

Data will be made available on request.

References

Alves, C., & De Carvalho, J. V. (2007). Accelerating column generation for variable sized bin-packing problems. *European Journal of Operational Research, 183*(3), 1333–1352.

Aydın, N., Muter, İ., & Birbil, Ş.İ. (2020). Multi-objective temporal bin packing problem: An application in cloud computing. *Computers & Operations Research, 121*, Article 104959.

Baldacci, R., Coniglio, S., Cordeau, J. F., & Furini, F. (2024). A numerically exact algorithm for the bin-packing problem. *INFORMS Journal on Computing, 36*(1), 141–162.

Borges, Y. G., Schouery, R. C., & Miyazawa, F. K. (2024a). Mathematical models and exact algorithms for the Colored Bin Packing Problem. *Computers & Operations Research, 164*(4), Article 106527.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. arxiv preprint arxiv:1611.09940.

Beyaz, M., Dokeroglu, T., & Cosar, A. (2015). Robust hyper-heuristic algorithms for the offline oriented/non-oriented 2D bin packing problems. *Applied Soft Computing, 36*(11), 236–245.

Borges, Y. G., Schouery, R. C., & Miyazawa, F. K. (2024b). Mathematical models and exact algorithms for the Colored Bin Packing Problem. *Computers & Operations Research, 164*, Article 106527.

Caprara, A., & Toth, P. (2001). Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics, 111*(3), 231–262.

Cheng, H., Cong, Y., Jiang, W., & Pu, S. (2024). Learning to solve Class-Constrained Bin Packing Problems via Encoder-Decoder Model. *In: The Twelfth International Conference on Learning Representations*.

Cintra, G. F., Miyazawa, F. K., Wakabayashi, Y., & Xavier, E. C. (2008). Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. *European Journal of Operational Research, 191*(1), 61–85.

Dantzig, G. B., & Wolfe, P. (1960). Decomposition principle for linear programs. *Operations research, 8*(1), 101–111.

Dhahbi, S., Berrima, M., & Al-Yarimi, F. A. (2021). Load balancing in cloud computing using worst-fit bin-stretching. *Cluster Computing, 24*(4), 2867–2881.

Duman, E. N., Taş, D., & Çatay, B. (2022). Branch-and-price-and-cut methods for the electric vehicle routing problem with time windows. *International Journal of Production Research, 60*(17), 5332–5353.

Ekici, A. (2021). Bin packing problem with conflicts and item fragmentation. *Computers & Operations Research, 126*(2), Article 105113.

Ekici, A. (2022). Variable-sized bin packing problem with conflicts and item fragmentation. *Computers & Industrial Engineering, 163*, Article 107844.

Ekici, A. (2023). A large neighborhood search algorithm and lower bounds for the variable-Sized bin packing problem with conflicts. *European Journal of Operational Research, 308*(3), 1007–1020.
Feillet, D., Dejax, P., Gendreau, M., & Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks: An International Journal, 44*(3), 216–229.
Firat, M., Crognier, G., Gabor, A. F., Hurkens, C. A., & Zhang, Y. (2020). Column generation based heuristic for learning classification trees. *Computers & Operations Research, 116*(4), Article 104866.
Fleszar, K. (2023). A new MILP model and fast heuristics for the variable-sized bin packing problem with time windows. *Computers & Industrial Engineering, 175*(1), Article 108849.
Fortunato, S. (2010). Community detection in graphs. *Physics Reports, 486*(3), 75–174.
Gao, M., Chen, Y., Li, J., & Wahab, M. I. M. (2023). Hybrid branch-and-price-and-cut algorithm for the two-dimensional vector packing problem with time windows. *Computers & Operations Research, 157*(9), Article 106267.
Haouari, M., & Mhiri, M. (2024). Lower and upper bounding procedures for the bin packing problem with concave loading cost. *European Journal of Operational Research, 312*(1), 56–69.
Haouari, M., & Serairi, M. (2009). Heuristics for the variable sized bin-packing problem. *Computers & Operations Research, 36*(10), 2877–2884.
He, P., Li, K., & Kumar, P. R. (2022). An enhanced branch-and-price algorithm for the integrated production and transportation scheduling problem. *International Journal of Production Research, 60*(6), 1874–1889.
Hemmelmayr, V., Schmid, V., & Blum, C. (2012). Variable neighbourhood search for the variable sized bin packing problem. *Computers & Operations Research, 39*(5), 1097–1108.
Huang, L., Xiao, F., Zhou, J., Duan, Z., Zhang, H., & Liang, Z. (2023). A machine learning based column-and-row generation approach for integrated air cargo recovery problem. *Transportation Research Part B: Methodological, 178*, Article 102846.
Joshi, C. K., Laurent, T., & Bresson, X. (2019). An efficient graph convolutional network technique for the travelling salesman problem (arXiv:1906.01227).
Jia, H., & Shen, S. (2021). Benders cut classification via support vector machines for solving two-stage stochastic programs. *INFORMS Journal on Optimization, 3*(3), 278–297.
Kool, W., Van Hoof, H., & Welling, M. (2018). Attention, learn to solve routing problems! In: International Conference on Learning Representations, September.
Li, R., He, M., He, H., & Deng, Q. (2022). Heuristic column generation for designing an express circular packaging distribution network. *Operational Research, 22*(2), 1103–1126.
López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives, 3*, 43–58.
Liu, Q., Cheng, H., Tian, T., Wang, Y., Leng, J., Zhao, R., Zhang, H., & Wei, L. (2021). Algorithms for the variable-sized bin packing problem with time windows. *Computers & Industrial Engineering, 155*(5), Article 107175.
Meliani, Y., Hani, Y., Elhaq, S. L., & El Mhamedi, A. (2022). A tabu search based approach for the heterogeneous fleet vehicle routing problem with three-dimensional loading constraints. *Applied Soft Computing, 126*(13), Article 109239.
Mezghani, S., Haddar, B., & Chabchoub, H. (2023). The evolution of rectangular bin packing problem—A review of research topics, applications, and cited papers. *Journal of Industrial and Management Optimization, 19*(5), 3329–3361.
Morabit, M., Desaulniers, G., & Lodi, A. (2021). Machine-learning–based column selection for column generation. *Transportation Science, 55*(4), 815–831.
Newman, M. E. (2004). Detecting community structure in networks. *The European Physical Journal B, 38*(2), 321–330.
Pessoa, A., Sadykov, R., & Uchoa, E. (2021). Solving bin packing problems using VRPSolver models. *Operations Research Forum, 2*(2), 1–25.
Pessoa, A., Sadykov, R., Uchoa, E., & Vanderbeck, F. (2020). A generic exact solver for vehicle routing and related problems. *Mathematical Programming, 183*(5), 483–523.
Puchinger, J., & Raidl, G. R. (2007). Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research, 183*(3), 1304–1327.
Quesnel, F., Wu, A., Desaulniers, G., & Soumis, F. (2022). Deep-learning-based partial pricing in a branch-and-price algorithm for personalized crew rostering. *Computers & Operations Research, 138*, Article 105554.
Santos, R. C., Lopes, A. L., Sanches, A. C., Gomes, E. P., Silva, E. A. D., & Silva, J. L. D. (2023). Intelligent automated monitoring integrated with animal production facilities. *Engenharia Agrícola, 43*(2), 1–10.
Seman, L. O., Rigo, C. A., Camponogara, E., Bezerra, E. A., & dos Santos Coelho, L. (2023). Explainable column-generation-based genetic algorithm for knapsack-like energy aware nanosatellite task scheduling. *Applied Soft Computing, 144*(13), Article 110475.
Schepler, X., Rossi, A., Gurevsky, E., & Dolgui, A. (2022). Solving robust bin-packing problems with a branch-and-price approach. *European Journal of Operational Research, 297*(3), 831–843.
Tomita, E., Tanaka, A., & Takahashi, H. (2006). The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science, 363*(1), 28–42.
Wang, C., Guo, C., & Zuo, X. (2021). Solving multi-depot electric vehicle scheduling problem by column generation and genetic algorithm. *Applied Soft Computing, 112*(15), Article 107774.
Wang, T., Hu, Q., & Lim, A. (2022). An exact algorithm for two-dimensional vector packing problem with volumetric weight and general costs. *European Journal of Operational Research, 300*(1), 20–34.
Wei, L., Lai, M., Lim, A., & Hu, Q. (2020). A branch-and-price algorithm for the two-dimensional vector packing problem. *European Journal of Operational Research, 281*(1), 25–35.
Wei, L., Luo, Z., Baldacci, R., & Lim, A. (2020). A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. *INFORMS Journal on Computing, 32*(2), 428–443.
Yu, M., Nagarajan, V., & Shen, S. (2022). Improving column generation for vehicle routing problems via random coloring and parallelization. *INFORMS Journal on Computing, 34*(2), 953–973.
Zhang, H., Yao, S., Liu, Q., Wei, L., Lin, L., & Leng, J. (2023). An exact approach for the constrained two-dimensional guillotine cutting problem with defects. *International Journal of Production Research, 61*(9), 2986–3003.
Zhang, J., Magnouche, Y., Bauguion, P., Martin, S., & Beck, J. C. (2024). Computing bipath multicommodity flows with constraint programming–based branch-and-price-and-cut. *INFORMS Journal on Computing, 36*(1), 141–162.