Deep reinforcement learning for solving the single container loading problem

Yakin Hajlaoui, Amel Jaoua & Safa Bhar Layeb

To cite this article: Yakin Hajlaoui, Amel Jaoua & Safa Bhar Layeb (2022): Deep reinforcement learning for solving the single container loading problem, Engineering Optimization, DOI: 10.1080/0305215X.2021.2024177

To link to this article: https://doi.org/10.1080/0305215X.2021.2024177

Published online: 11 May 2022.

ENGINEERING OPTIMIZATION
https://doi.org/10.1080/0305215X.2021.2024177

Deep reinforcement learning for solving the single container loading problem

Yakin Hajlaoui[a,b], Amel Jaoua [b] and Safa Bhar Layeb [b]

[a]GERAD & Département de Mathématiques et de Génie Industriel, Polytechnique Montréal, Montréal, Québec, Canada; [b]LR-OASIS, National Engineering School of Tunis, University of Tunis El Manar, Tunis, Tunisia

ABSTRACT
In this article, a new deep Reinforcement Learning (RL) model is proposed to solve the Single Container Loading Problem (SCLP) as well as the SCLP with full support. For that purpose, a multilayer neural network architecture is used. Computational experiments, conducted on benchmark instances with homogeneous boxes, revealed that the proposed model yields fairly good results compared to those found with state-of-the-art heuristics. Experiments have also shown good generalization capability of the proposed deep RL model to deal with both homogeneous and heterogeneous classes of instances. Nevertheless, this learning-based optimization approach still has an optimality gap compared to the well-designed heuristics of the operations research literature. Finally, the benefit of training the model under different levels of variability has been analysed. Results revealed that, for better performance, and if a company does not face high demand volatility, it is recommended to train the model under a low level of variability.

ARTICLE HISTORY
Received 16 June 2020
Accepted 27 December 2021

KEYWORDS
Single container loading problem; multilayer neural network; deep reinforcement learning

1. Introduction

Finding optimal packaging and increasing the capacity utilization of containers are key to improving logistics systems. Not surprisingly, these topics have caught the interest of both practitioners and researchers for several decades and many researchers have focused on solving so-called Container Loading Problems (CLPs) (Castellucci, Toledo, and Costa 2019). Specifically, one of the highly challenging CLPs is the Single Container Loading Problem (SCLP), which is formally defined as follows. Given a set $I = \{1, \dots, m\}$ of three-dimensional rectangular boxes, where each box i, $i \in I$, is characterized by a length l_i, a width w_i, a height h_i and a volume $v_i = l_i w_i h_i$, let a three-dimensional rectangular container be defined by its prefixed length L, width W and height H. The SCLP requires loading the entire or part of the m boxes into the container, in order to maximize its volume utilization. Moreover, each loaded box should be packed orthogonally into the container without any overlapping with the other boxes. Not surprisingly, the SCLP is a notoriously intractable NP-hard combinatorial optimization problem.

In today's context of multimodal freight transportation, optimizing the loading of packages into containers for overseas shipping is an often encountered issue (Olsson, Larsson, and Quttineh 2020). Dependent on the type of items to ship, some companies may have to load packages with identical dimensions, *i.e.* homogeneous boxes, whereas others have to deal with a large variety of packages, *i.e.*

CONTACT Safa Bhar Layeb safa.layeb@enit.utm.tn

Supplemental data for this article can be accessed here. https://doi.org/10.1080/0305215X.2021.2024177

heterogeneous boxes. Also, one of the key requirements to avoid damaging a cargo in real life loading situations is respecting the packing stability and, more precisely, the so-called *full support* condition. Actually, full support implies that every loaded box inside the container should be fully supported from below (Araya, Guerrero, and Nuñez 2017). When this condition is taken into account, the SCLP variant is the so-called Single Container Loading Problem with Full Support (SCLP-FS).

Therefore, the objective of the present work is to propose a new deep Reinforcement Learning (RL) approach to solve the SCLP as well as the SCLP-FS. During the last decades, deep RL techniques have had great success on a variety of *NP*-hard optimization problems from resource allocation (Vengerov 2007) to flowshop scheduling (Gupta, Majumder, and Laha 2019). According to Gupta, Majumder, and Laha (2019), deep RL has gained huge attraction over classical operations research techniques because, once trained, it can solve problems in almost real-time. In fact, deep RL combines the reinforcement learning framework with the learning capability of deep Neural Networks (NNs) based on many hidden layers, to improve the performance of control policies (Mousavi, Schukat, and Howley 2017). In this article, a deep NN architecture for the SCLP is proposed. The developed deep RL model for the basic SCLP as well as the SCLP-FS is implemented and experiments performed on benchmark instances from the literature. In order to show the capability of the training to face an uncertain number of boxes, experimentation with different levels of variability are conducted.

The remainder of this article is organized as follows. An overview of the recent literature is presented in Section 2. Section 3 describes a mathematical formulation for the SCLP as well as a generic deep reinforcement learning model. Section 4 details the implementation of the deep RL model, for solving the considered variants of the SCLP, and provides an illustrative toy example. Section 5 reports the results of the computational experimentation. Finally, Section 6 derives conclusion and indicates directions for future research.

2. Related work

By reason of its *NP*-hard nature, solving the SCLP to optimality poses redoubtable challenges. Some exact methods are based on developing and solving mathematical formulations (Scheithauer 2017). In this regard, Fasano (1999) proposed a linear programming formulation for the three-dimensional orthogonal packing problem. Then Padberg (2000) extended Fasano's model and subjected it to polyhedral analysis. In the same vein, Allen, Burke, and Mareček (2012) failed to solve instances having up to 20 boxes using this formulation with modern solvers. Afterwards, they derived a new space-indexed formulation that shows better performance. Junqueira, Morabito, and Yamashita (2012) introduced three integer linear programming models for the cubic container loading problem with realistic constraints on cargo stability and load bearing conditions. Computational experimentation, using CPLEX® 11.0 as solver, revealed that the proposed models are able to handle only instances of a quite moderate size. More recently, Kurpel *et al.* (2020) amended the model of Junqueira, Morabito, and Yamashita (2012) to find optimal solutions for several CLPs with practical constraints.

Most of the literature on the SCLP involves heuristic approaches. They are herein categorized into the following three, *not necessarily disjoint*, state-of-the-art classes.

2.1. Conventional heuristics

These include methods based on the principles of construction and/or improvement mainly inspired by the work of Bischoff and Ratcliff (1995) and the so-called maximal space representation of Lim, Rodrigues, and Wang (2003). Accordingly, Parreño *et al.* (2008) proposed their maximal-space algorithm that combines randomized and deterministic packing strategies. The same authors hybridized a variable neighbourhood search approach in their later work (Parreño *et al.* 2010). Layeb, Jabloun, and Jaoua (2017) proposed a greedy two-step look-ahead heuristic that selects free spaces deterministically followed by a block search. They also integrated load-bearing, weight limits and positioning constraints. Araya and Riff (2014) proposed a block-building based algorithm that uses a greedy-based function to evaluate nodes in the beam search tree. Their work was then enhanced by introducing a new evaluation function for ranking blocks (Araya, Guerrero, and Nuñez 2017).

2.2. Metaheuristics

Many genetic algorithms have been developed that generate fairly competitive results on benchmark datasets, see for example Gonçalves and Resende (2012) and Xiang *et al.* (2018). Liu *et al.* (2011) proposed a hybrid tabu search approach that integrates both tabu search and loading heuristics. A hybrid bee algorithm was developed by Dereli and Das (2011) for the SCLP with many similarities to evolutionary mechanisms. Along the same lines, a well-designed joint hybrid artificial bee colony algorithm was recently proposed by Bayraktar, Ersöz, and Kubat (2021). Saraiva, Nepomuceno, and Pinheiro (2019) developed a two-phase block-loading hybrid metaheuristic. Computational experimentation has shown good performance for SCLP instances with weakly heterogeneous boxes.

2.3. Tree search methods

Similar to constructive procedures, these approaches select the next box to be placed in the container by exploring the search space using a suitable size-limited tree search or by using graph search methods (Pisinger 2002). Fanslau and Bortfeldt (2010) generalized a block building approach within a tree search framework. To decrease the searching space, a fit degree algorithm was proposed by He and Huang (2011). Zhu *et al.* (2012) identified the so-called six key elements and proposed a greedy two-step lookahead framework that uses both the tree search algorithm (Fanslau and Bortfeldt 2010) and the maximal space algorithm (Parreño *et al.* 2008). Furthermore, Zhu and Lim (2012) introduced an iterative-doubling greedy-lookahead tree search heuristic for the SCLP as well as the SCLP-FS. By enhancing the depth-first search method, an effective block-loading algorithm with multi-layer search was developed in D. Zhang, Peng, and Leung (2012). Recently, Da Silva *et al.* (2020) proposed a new wall-based approach with a greedy improvement phase that takes into account a realistic set of practical constraints.

For the sake of convenience, literature on the related SCLP variants published during the last decade has been compiled in a summary table available in the online supplemental data, which can be accessed at https://doi.org/10.1080/0305215X.2021.2024177. Thus, the following points should be highlighted.

- Although the literature includes well-engineered state-of-the-art resolution approaches, alternative avenues are still under investigation. By way of example, but not limited to the following, Mladenović *et al.* (2019) and Olsson, Larsson, and Quttineh (2020) used classical heuristics to automate the planning of container loading within decision support system frameworks. Sahun (2020) pointed out the high relevance of using both expert system and artificial intelligence tools for potentially advantageous improvements in a real-world loading context. Recently, Kilincci and Medinoglu (2021) investigated solving hybrid integer nonlinear programming models within a real case study in the plastics industry.
- In recent years, reinforcement learning has emerged as the trendy avenue for solving operations research problems. As stated by Vesselinova *et al.* (2020), the main motivation behind this trend in using deep reinforcement learning for solving combinatorial optimization problems is the ability of a learned model to adapt to perturbation in the problem setting without the need to re-optimize algorithms. For an extended review of reinforcement learning applications for solving combinatorial optimization problems, the interested reader could consult the two recent surveys of Mazyavkina *et al.* (2020) and Vesselinova *et al.* (2020). According to these works, deep reinforcement learning has never been formally investigated for solving the SCLP. Thus, the objective

of the present work is to show how this combinatorial optimization problem, *i.e.* the SCLP, may be modelled and solved using this promising deep reinforcement learning approach.

3. Problem formulation

In this section, a mathematical model is first detailed for solving the deterministic SCLP. Then, the implemented deep RL model is depicted.

3.1. The mathematical model

Let $\mathcal{A}_C = \{X, Y, Z\}$ be the three-dimensional Cartesian reference frame associated with the container. Accordingly, the container dimension along axis A, $A \in \mathcal{A}_C$, is denoted by D_A. More precisely, $D_X = L$, $D_Y = W$, and $D_Z = H$. In addition, let $d_{i,a}$ be the dimension size of box i, $i \in I$, corresponding to its principal axis side o, $o \in \{1, 2, 3\}$. Thus, $d_{i,o} \in \{l_i, w_i, h_i\}, \forall\, i \in I, o \in \{1, 2, 3\}$. At this stage, it is worth remembering that each box could be freely orthogonally orientated. In line with its cuboid form, each box could have six possible orientations within the three-dimensional container. Now, define $\chi_{i,o,A}$, $i \in I$, $o \in \{1, 2, 3\}$, $A \in \mathcal{A}_C$, as a binary variable that takes on a value of one if box i is placed with its side axis o parallel to axis A of the container, and zero otherwise. These variables were first introduced by Fasano (1999) to capture the orthogonal packing requirement. Furthermore, to reflect the positioning and the non-overlapping assumptions (Padberg 2000), consider a continuous non-negative decision variable $\varepsilon_{i,A}$ that represents the coordinates of the centre of gravity of box i, $i \in I$, with respect to axis A, $A \in \mathcal{A}_C$. The gravity centre of each box is indeed assumed to be its geometric centre. Let $\zeta_{i,j,A}$, $i, j \in I$, $i \neq j$, $A \in \mathcal{A}_C$, denote the binary decision variable that takes the value one if box i is placed before box j in the direction A, and zero otherwise.

Using these definitions, a Mixed Integer Linear Programming (MILP) formulation for the SCLP can be stated as follows:

$$\text{SCLP}: \quad \text{Maximize} \sum_{i \in I} \sum_{A \in \mathcal{A}_C} v_i \chi_{i,1,A} \tag{1}$$

subject to:

$$\sum_{A \in \mathcal{A}_C} \chi_{i,1,A} \leq 1, \quad \forall\, i \in I, \tag{2}$$

$$\sum_{A \in \mathcal{A}_C} \chi_{i,1,A} = \sum_{A \in \mathcal{A}_C} \chi_{i,2,A}, \quad \forall\, i \in I, \tag{3}$$

$$\sum_{A \in \mathcal{A}_C} \chi_{i,1,A} = \sum_{o \in \{2,3\}} \chi_{i,o,A}, \quad \forall\, i \in I,\ \forall\, A \in \mathcal{A}_C, \tag{4}$$

$$\sum_{o \in \{1,2,3\}} \frac{d_{i,o}}{2} \chi_{i,o,A} \leq \varepsilon_{i,A}, \quad \forall\, i \in I,\ \forall\, A \in \mathcal{A}_C, \tag{5}$$

$$\varepsilon_{i,A} \leq \sum_{o \in \{1,2,3\}} \left(D_A - \frac{d_{i,o}}{2} \right) \chi_{i,o,A}, \quad \forall\, i \in I,\ \forall\, A \in \mathcal{A}_C, \tag{6}$$

$$D_A \zeta_{i,j,A} + \sum_{o \in \{1,2,3\}} \frac{d_{i,o}}{2} \chi_{i,o,A} - \sum_{o \in \{1,2,3\}} \left(D_A - \frac{d_{j,o}}{2} \right) \chi_{j,o,A} \leq \varepsilon_{i,A} - \varepsilon_{j,A},$$
$$\forall\, i, j \in I,\ i < j,\ \forall\, A \in \mathcal{A}_C, \tag{7}$$

$$\varepsilon_{i,A} - \varepsilon_{j,A} \leq \sum_{o \in \{1,2,3\}} \left(D_A - \frac{d_{i,o}}{2} \right) \chi_{i,o,A} - \sum_{o \in \{1,2,3\}} \frac{d_{j,o}}{2} \chi_{j,o,A} - D_A \zeta_{i,j,A},$$

$$\forall\, i,j \in I,\ i < j,\ \forall\, A \in \mathcal{A}_C, \tag{8}$$

$$\sum_{A\in\mathcal{A}_C} (\zeta_{i,j,A} + \zeta_{j,i,A}) \le \sum_{A\in\mathcal{A}_C} \chi_{i,1,A}, \quad \forall\, i,j \in I,\ i < j, \tag{9}$$

$$\sum_{A\in\mathcal{A}_C} (\zeta_{i,j,A} + \zeta_{j,i,A}) \le \sum_{A\in\mathcal{A}_C} \chi_{j,1,A}, \quad \forall\, i,j \in I,\ i < j, \tag{10}$$

$$\sum_{A\in\mathcal{A}_C} \chi_{i,1,A} + \sum_{A\in\mathcal{A}_C} \chi_{j,1,A} \le 1 + \sum_{A\in\mathcal{A}_C} (\zeta_{i,j,A} + \zeta_{j,i,A}), \quad \forall\, i,j \in I,\ i < j, \tag{11}$$

$$\chi_{i,o,A} \in \{0,1\}, \quad \forall\, i \in I,\ \forall\, o \in \{1,2,3\},\ \forall\, A \in \mathcal{A}_C, \tag{12}$$

$$\zeta_{i,j,A} \in \{0,1\}, \quad \forall\, i,j \in I, i \ne j,\ \forall\, A \in \mathcal{A}_C, \tag{13}$$

$$\varepsilon_{i,A} \ge 0, \quad \forall\, i \in I,\ \forall\, A \in \mathcal{A}_C. \tag{14}$$

The objective function (1) requires maximizing the total volume utilization of the container. Note that, if

$$\sum_{A\in\mathcal{A}_C} \chi_{i,1,A} = 0,$$

then box i, $i \in I$, is not placed in the container. Thus, Constraints (2) enforce the first side, *i.e.* $o = 1$, of each loaded box i, $i \in I$, to be parallel to only one of the three axes of the container. Constraints (3) express that, once box i, $i \in I$, is loaded, *i.e.*

$$\sum_{A\in\mathcal{A}_C} \chi_{i,1,A} = 1,$$

then its second side should be parallel to one of the axes of the container. Constraints (4) assert the mutual parallelism between the sides of each loaded box i, $i \in I$, and the axes of the container. To be precise, if box i, $i \in I$, is loaded, then each of its sides should be parallel to at most one of the three axes of the container. Therefore, Constraints (2)–(4) express orthogonal packing conditions. Constraints (5), in conjunction with (6), ensure appropriate coordinates of the centre of gravity of each loaded box i, $i \in I$, which guarantee its entire placement in the container. Constraints (7), in conjunction with (8), fulfil the non-overlapping condition while enforcing minimum distances between the placed boxes. The bundle Constraints (9) and (10) state that, if one box is loaded before or after another box within the container, then both boxes should obviously be in the container. Constraints (11) express the fact that, if two boxes are loaded into the container, then one of the boxes should be placed before the other along an axis of the container. Finally, Constraints (12) and (13) impose the integrality of the χ- and ζ-variables, and Constraints (14) define the non-negativity of the ε-variables.

Due to its polynomial structure, the model (1)–(14) could be solved using a general-purpose mixed integer programming solver. Not surprisingly, this analytical model would quickly become intractable in practical cases, because of its quite considerable number of variables and constraints. It goes without saying that, if additional constraints have to be taken into account, the model would most likely turn out to be nonlinear (Lim *et al.* 2013).

3.2. The deep reinforcement learning model

In line with reinforcement learning principles, the agent and the environment interact along a sequence of discrete time-steps $t = 0, 1, 2, 3, \ldots$ based on trial-and-error interactions as shown in Figure 1. At each time-step t, the agent receives some representation of the environment's state $s_t \in S$, where S is the set of possible states, and selects an action $a_t \in A(s_t)$, where $A(s_t)$ is the set of available

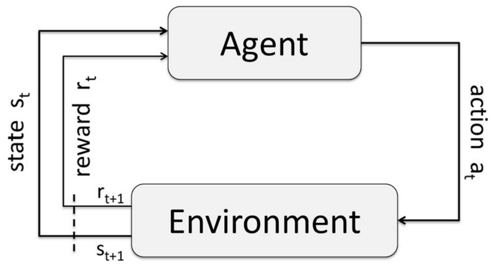


Figure 1. Agent–environment interaction in reinforcement learning.

actions in state s_t. As a consequence of the selected action, the state s_t will be updated to s_{t+1} and the agent receives a numerical reward signal r_t.

The agent's decision making procedure is characterized by a policy expressed as $\pi(a_t, s_t, w) = P(a_t \mid s_t, w)$, where w is a parameter vector including the weights in the deep neural network. The so-called *policy gradient* method is applied herein. Actually, this approach states that the policy is explicitly represented by its own function approximation and is updated according to the gradient of the expected reward $E_w(R)$ expressed as

$$E_w(R) = \sum_t P(a_t \mid s_t, w) \cdot r_t. \tag{15}$$

As in the gradient descent algorithm, the update rule of the policy parameter w, in the direction that maximizes $E_w(R)$, is

$$w := w + \alpha \times \nabla_w(E_w(R)), \tag{16}$$

where α is an adequate step size parameter. The reader interested in reinforcement learning techniques is referred to the comprehensive survey of Xu, Zuo, and Huang (2014).

In the case of the SCLP, to make the agent learn a '*good*' loading policy, *i.e.* that maximizes the volume utilization of the container, the policy gradient method is applied after each M experiences of loading processes. M is the so-called *batch size* of experiences collected over successive episodes into a reply memory dataset, where an experience is the set of accurate states, actions and rewards at each time-step of the loading process. Updating the deep neural network stops when the agent converges to a good policy. Thus, the process of learning a good policy is called *training*. To model the SCLP as a deep RL problem, it should be considered as a process of actions or decisions $(a_1, \ldots, a_t, \ldots, a_n)$ where a_t is the chosen action at time-step t for a given state s_t. The state should be modelled as an input variable that expresses the state of the container. The loading process will stop if there are no more available actions to choose. The implementation details will be specified in Section 4.

Actually, within the deep RL model, the loading process is an arrangement of loading actions $(a_1, \ldots, a_t, \ldots, a_n)$, where a_t is the chosen action at time-step t for a given state s_t. Algorithm ?? presents the training algorithm of the deep reinforcement learning model that calls the loading procedure detailed later in Section 4.

At each time-step, the agent will choose an action that is an order to place a generated block b in free space f with orientation o. Recall that the deep neural network provides the policies $\pi(a_t, s_t, w)$, $a_t \in A(s_t)$. Then, the selected action a_t corresponds to the optimal policy

$$\pi_t^* = \underset{a_t \in A(s_t)}{\arg\max}\{\pi(a_t, s_t, w)\}. \tag{17}$$

During the loading procedure, some actions are undefined, set to zero and called wrong actions as they represent unfeasible placements. If the currently selected action is a wrong action, the loading

Algorithm 1: Deep reinforcement learning algorithm

```
1 $episode\_number = 0$;
2 $batch\_size = M$;
3 While True:
4     Apply the Loading Process Algorithm (Algorithm 2, which is available in the
      online supplemental data);
5     Assign a reward for each chosen action and add them to $rewards\_list$;
6     $episode\_number = episode\_number + 1$;
7     Apply the Policy Gradient Algorithm to the deep NN after each $batch\_size$
      episode with the recorded data;
```

process will stop and the action will be assigned a negative reward. Otherwise, the loading process will continue to perform actions, update the list of states and the list of available actions, and record the different states and actions chosen for the data until the container is fully loaded, *i.e.* there is no more action available to be selected.

4. Implementation details

In this section, the implementation details of the deep RL to solve the SCLP are given. To be precise, the proposed architecture of the neural network is depicted. Then, specifications of how to add the full base support constraints are presented. Finally, a toy example is given to illustrate this resolution mechanism. Algorithms 2–6 discussed in this section are available in the online supplemental data.

4.1. Deep RL for the SCLP

As stated earlier in Section 3, the solving approach model is based on the loading process mechanism specified in Algorithm 2. To be precise, the implemented loading process consists of choosing, for each state s_t, a set of actions $A(s_t)$ computed using Algorithm 3. The set of actions $A(s_t)$ and the state s_t are fed to the deep neural network to select an action $a_t = (block\ b_t, space\ f_t)$. Such an action means that a block b_t represented by its coordinates $[b_t \cdot x_1, b_t \cdot y_1, b_t \cdot z_1, b_t \cdot x_2, b_t \cdot y_2, b_t \cdot z_2]$ is placed inside a space f_t represented by its coordinates $[f_t \cdot x_1, f_t \cdot y_1, f_t \cdot z_1, f_t.x_2, f_t \cdot y_2, f_t \cdot z_2]$. It is worth mentioning that this loading process is based on the placement of blocks rather than boxes. A block is a set of boxes that is placed inside a minimum bounding cuboid. According to Lim, Rodrigues, and Wang (2003) and Araya, Guerrero, and Nuñez (2017), this building approach using blocks instead of boxes has reported the best results for the SCLP. Thus, in the present work, the Maximum Generated Block Algorithm (Algorithm 5) is used to construct such blocks of boxes.

As shown in Figure 2, each cuboid inside the container, whether it is a block b or a free space f, is modelled with its coordinates (x_1, y_1, z_1) and (x_2, y_2, z_2), designating respectively the coordinates of the corner of the cuboid closest to and furthest from the origin corner of the container.

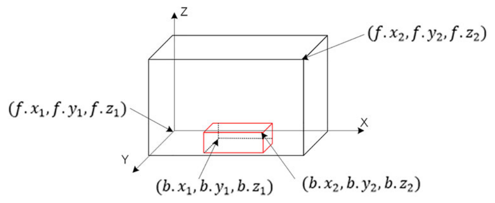


Figure 2. Cuboids coordinates representation.

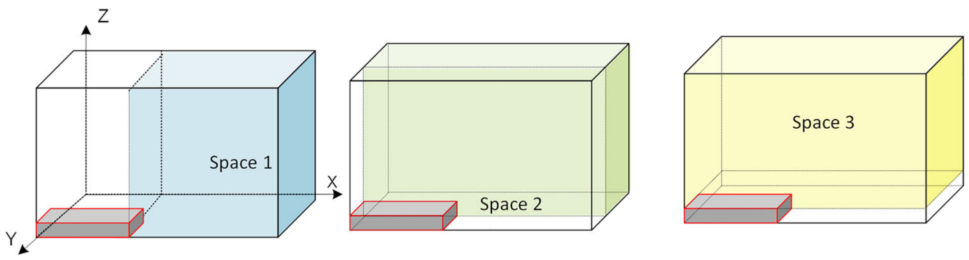


Figure 3. Cuboids coordinates representation.

As presented in Algorithm 2 at each step, once a block is placed, a new free space list is computed using the Update Space Algorithm (Algorithm 4). This update free space mechanism aims to represent the residual space in the container by a list of overlapping maximal cuboids, *i.e.* empty space cuboids that cannot be extended. This method of representing residual space is known as *cover representation* and was first proposed by Lim, Rodrigues, and Wang (2003). As displayed in Figure 3, once a block is placed, three new overlapping cuboids are generated using a guillotine cut via the x-axis, followed by the y-axis, and then the z-axis. Next, based on Algorithm 6, the residual space in the container is computed taking into account this new list of overlapped cuboids. In the present work, the state space S is characterized through the application of this cover representation mechanism to the placed block at time-step t. In fact, in the multilayer neural network architecture implemented herein, the input nodes hold the elements of the vector s_t, *i.e.* the state of the container at time-step t, which is expressed as

$$s_t = [t, b_{t-1} \cdot x_1, b_{t-1} \cdot y_1, b_{t-1} \cdot z_1, b_{t-1} \cdot x_2, b_{t-1} \cdot y_2, b_{t-1} \cdot z_2]. \tag{18}$$

Specifically, seven input nodes are implemented. The first represents the time-step and the other six refers to the coordinates of the last placed block. It is worth mentioning that, since each step is associated with a new space list, this state modelling fully characterizes the environment at every step.

In the implemented deep NN architecture, the output nodes correspond to the available actions a_t, *i.e.* feasible placements that could be executed at time-step t. As mentioned earlier, Algorithm 3 describes the construction of the set of available actions. To be precise, at each time-step t of the loading process, the agent should choose one action among these available actions. To model the actions of the output nodes, the maximum generated block heuristic (Zhu *et al.* 2012) is used, as described in Algorithm 5. This later takes as input a space f, a box b, a number of boxes n_b that must not be exceeded, and an orientation c and constructs the block b with a maximum number of boxes of the same type that the space f could hold in the orientation c.

4.2. Deep RL for the SCLP-FS

On a practical level, the proposed solutions to the SCLP should be stable. Therefore, the SCLP-FS was targeted by going further in adapting the aforementioned deep RL algorithm to handle full base support constraints. It is worth stating that the stability of a placed block b is tested through the following condition: if a block b is in an unstable position, then there exists at least one space f such that

$$b \cdot z_1 \geq f \cdot z_2, \tag{19}$$

and that verifies at least one of the following inequalities:

$$b \cdot x_2 > f \cdot x_1, \tag{20}$$

$$b \cdot x_1 < f \cdot x_2, \tag{21}$$

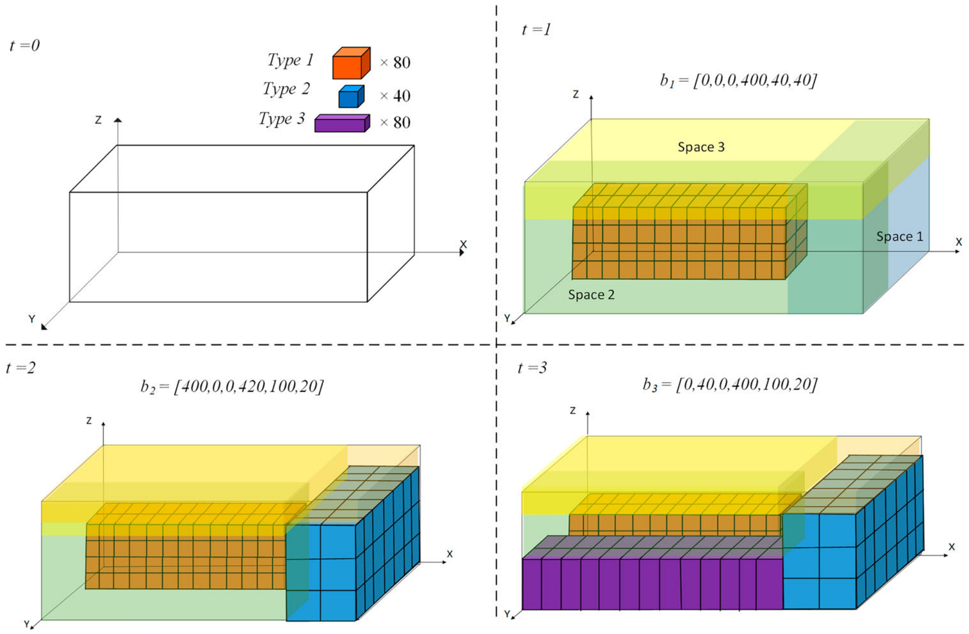


Figure 4. Toy example.

$$b \cdot y_2 > f \cdot y_1, \tag{22}$$

$$b \cdot y_1 < f \cdot y_2. \tag{23}$$

Condition (19) ensures that block b should be above space f. Inequalities (20)–(23) imply that space f should overlap block b along the x-axis or the y-axis, respectively.

The stability condition is thereby integrated into the model described in Section 4.1. To be precise, the loading procedure is amended with this condition to test whether an action leads to an unstable block or not. This test is used in the loading process algorithm and checks whether a configuration leads to an unstable block or not. It stops the process and returns a negative reward if the block is unstable, treating the move as a wrong move.

4.3. Illustrative toy example

In this subsection, a toy example is presented to illustrate the implemented resolution process of the SCLP based on the deep NN mechanism. For that purpose, begin by considering the basic example of filling a container with three different types of box, such that the dimensions of the container are $(L, W, H) = (420, 100, 50)$. There are 80 boxes of type 1, 40 boxes of type 2 and 80 boxes of type 3. The dimensions of boxes of type 1, 2 and 3 are $(20, 20, 20)$, $(10, 10, 10)$ and $(40, 30, 5)$, respectively. Figure 4 provides an evolution of the loading process through an example of three steps. At time-step $t = 0$, the state is initialized as $s_0 = [0, 0, 0, 0, 0, 0, 0]$. Initially, the container is empty and the set of spaces comprises a unique free space f_0 represented by the coordinates of the container $[0, 0, 0, 420, 100, 50]$. Given the current state and the available boxes, a set of actions is computed with Algorithm 3 that are then fed to the deep NN to select the next action.

At time-step $t = 1$, as illustrated in Figure 4, the embedded action consists of generating a block b_1 using 80 boxes of type 1. This block is made of the 80 boxes placed on the x-, y-, z-axes as follows: $20\times2\times2$. As a result, the state $s_1 = [1, 0, 0, 0, 400, 40, 40]$ according to Expression (18).

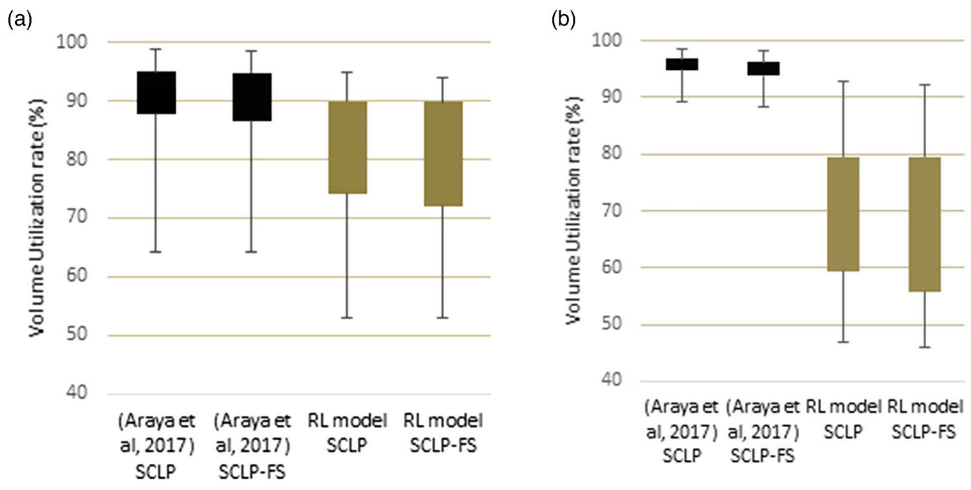


Figure 5. Boxplots for the homogeneous set BR0 (a) and the heterogeneous set BR1 (b).

At the next time-step $t = 2$, based on this state s_1, a new space list is defined using Algorithm 4. In fact, at each step based on the last placed block, a new space is defined, then with the available boxes, a new set of actions is computed with Algorithm 3. In the example herein, the residual space in the container is computed based on the three new overlapping cuboids, *i.e.* Space 1, Space 2 and Space 3, as shown in Figure 4. Using Algorithm 4, a new free space f will be generated for each residual space. For example, for the residual Space 1, in Figure 4, the free space coordinates are $[400, 0, 0, 420, 100, 50]$. Then, based on this list of residual spaces and the available boxes, Algorithm 3 generates a new list of possible actions. An embedded action, shown in Figure 4 at time-step $t = 2$, is to place a second block b_2 on the free Space 1. This block b_2 is composed of the 40 boxes of type 2 placed on the x-, y-, z-axes as follows: $2\times10\times2$. This action leads to the new state $s_2 = [2, 400, 0, 0, 420, 100, 20]$.

Then, for the time-step $t = 3$, given this state s_2, a new free space list is generated again with Algorithm 4. Also, a set of actions is defined with Algorithm 3. As depicted in Figure 4 at time-step $t = 3$, the embedded action is to place a third block b_3 on a newly generated free space f, the coordinates of which are $[0, 40, 0, 400, 100, 50]$. This block b_3 is composed of the 80 boxes of type 3 placed on the x-, y-, z-axes as follows: $10\times2\times4$. This placement leads to the state $s_3 = [3, 0, 40, 0, 400, 100, 20]$. The process will stop if the there are no more boxes to place or if the container is fully loaded. In the next section, numerical experiments will be conducted using this implemented mechanism of deep NN to solve the SCLP.

5. Experiments and results

The proposed deep RL algorithms were implemented using the Python® 3.6 programming language and the TensorFlow library. All the computational experiments were carried out on an Intel® Core™ i7 personal computer with a rack-mounted server and 8.0 GB of RAM under the Windows® 10 operating system. To evaluate the performance of the proposed deep RL model, experiments were first conducted to solve the SCLP, secondly the SCLP-FS was considered, and finally uncertainty about the number of boxes was introduced to assess the ability of the model to perform on unseen instances.

5.1. Results for the SCLP

Depending on the type of box, different classes of instance with different features have been proposed in the literature. These instances can be classified into two broad categories: homogeneous sets with

Table 1. Instance characteristics for homogeneous boxes.

Dataset	Number of boxes	Container length, L	Container width, W	Container height, H	Box length, l	Box width, w	Box height, h
A1-10	$\left\lfloor \frac{(L\times W\times H)}{(l\times w\times h)} \right\rfloor$	10	10	10	$[0.25L, 0.75L]$	$[0.25W, 0.75W]$	$[0.25H, 0.75H]$
A1-20	$\left\lfloor \frac{(L\times W\times H)}{(l\times w\times h)} \right\rfloor$	20	20	20	[0.25L, 0.75L]	[0.25W, 0.75W]	[0.25H, 0.75H]
BR0	Random	587	233	220	Random	Random	Random

identical box dimensions and heterogeneous sets with several types of box having different dimensions. Thus, the reliability of the deep RL model will first be explored for solving the homogeneous instances, and then the heterogeneous category will be investigated.

5.1.1. Experiments with homogeneous boxes

Under this first category, the most common benchmark instances are the basic set of instances A1, proposed by Junqueira, Morabito, and Yamashita (2012), and the more complex one that consists of the BR0 instances, proposed by Davies and Bischoff (1999) with a larger number of boxes. Instance characteristics of each dataset are given in Table 1. For the sake of simplicity, the two subsets of A1, namely A1-10 and A1-20, consider only cubic containers, *i.e.* $L = W = H$. For each box i, the dimensions of boxes (l_i, w_i, h_i) are randomly generated by a Uniform distribution from the corresponding interval. As the boxes are homogeneous, let (l, w, h) denote the corresponding dimensions, *i.e.* $(l, w, h) = (l_i, w_i, h_i), \forall\, i \in I$. The number of boxes is set equal to $\lfloor (L\times W\times H)/(l\times w\times h) \rfloor$, which means that the problem is not constrained with respect to the availability of boxes.

For the first set A1, the results obtained from the deep RL approach and that found through exact resolution, *i.e.* optimal solutions reached when the proposed mathematical model (1)–(14) is solved with CPLEX®, are given in the online supplemental data. Obviously, the deep RL model can solve all instances. Meanwhile, the MILP model was unable to deliver optimal solutions and solve instances from 13 boxes. To assess the quality of solutions, the gap between the filling rate of the deep RL model and the best known solution found by Kurpel *et al.* (2020) is displayed and calculated as

$$\text{Gap} = \frac{(\text{Best} - \text{known filling rate}) - (\text{deep RL filling rate})}{\text{Best} - \text{known filling rate}} \times 100.$$

It is noteworthy that even though the deep RL filling rate is on average 12.40% worse than the best known solutions, for some instances such as for A1-10-2 the deep RL model reaches a good solution of 93.99%. Also, for A1-20-01, the RL model found a solution nearly equal to the optimal one with a Gap of only 1.45%.

Once the capability of the RL model to solve basic instances with a small number of boxes has been illustrated, the next experiments will investigate the more realistic BR0 instances drawn from a different distribution than A1, as displayed in Table 1. Since the minimum number of boxes in BR0 is 41, the exact method using the proposed MILP model can no longer solve these large-size instances for the SCLP. To the best of the authors' knowledge, and according to many recent works (Saraiva, Nepomuceno, and Pinheiro 2019; Da Silva *et al.* 2020), the results found by Araya, Guerrero, and Nuñez (2017) based on an heuristic approach represent the current state-of-the-art for this problem. Numerical results for the 100 instances of BR0 are detailed in the online supplemental data in order to analyse the performance on each instance individually. Accordingly, even though the deep RL model found solutions with an average gap of 12.71%, for some instances this gap is less than 5%. Precisely for 23 instances, the model had reached a fairly good filling rate. Also, this average gap remained almost stable for the A1 small instances, with an average gap of 12.40%, and for the BR0 larger instances, with an average gap of 12.71%. Additionally, the number of boxes in BR0 reached 1169, whereas the maximum number of boxes tested under A1 was 27. That means that the proposed deep RL model scaled well as the number of boxes increased. Also, it is worth mentioning that, even though the sets A1 and BR0 were generated according to different data distributions, the RL results were found using the same hyper-parameters. Actually, when machine learning is used to solve combinatorial optimization problems, the scalability is an important topic. In this case, computational results revealed that the running time of the deep RL model do not increase dramatically for the larger BR0 instances.

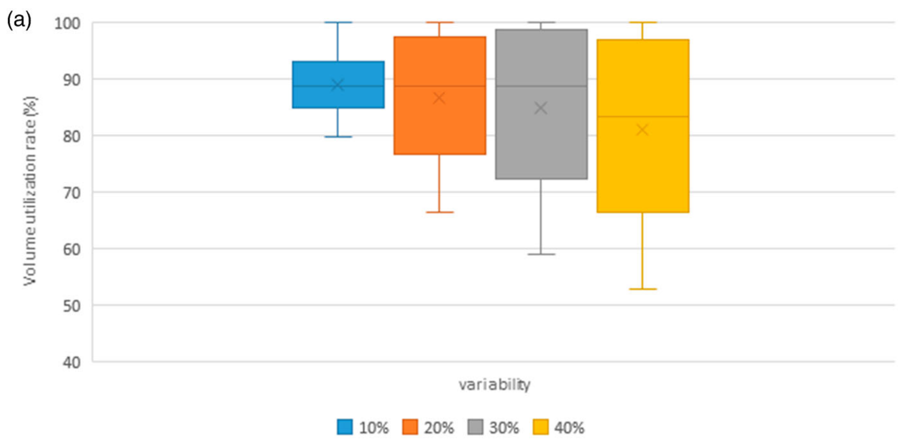
5.1.2. Experiments with heterogeneous boxes

In order to assess empirically the capability of the model to solve the SCLP with heterogeneous boxes, another structure of benchmark instances provided by Bischoff and Ratcliff (1995), namely BR1, was considered. This dataset involves three different types of box with up to 476 boxes to load. The corresponding best-known solutions were found by Araya, Guerrero, and Nuñez (2017) and Saraiva, Nepomuceno, and Pinheiro (2019). According to Saraiva, Nepomuceno, and Pinheiro (2019), for these difficult BR1 benchmark instances, there is no particular algorithm that stands out from the others for all instances. Therefore, it is important to analyse the performance of the proposed model on each instance. Computational results, available in the online supplemental data, show that the deep RL model can still solve all instances from BR1, although the solution quality deteriorates as it gives a high average gap of more than 27% to the best-known solutions. Only for eight instances does the model give fairly good solutions with a gap of less than 5%. However, for some instances such as BR1-14 and BR1-78, the deep RL solutions are competitive with those recently found by Saraiva, Nepomuceno, and Pinheiro (2019). It is worthy of note that the training time significantly increased, reaching 9895 seconds for instance BR1-67, whereas other heuristic algorithms found the reported filling rate within a maximum runtime of 500 seconds. This issue of high computation time is a well-known problem when deep NNs having many hidden layers, leading to a huge number of parameters to learn. According to Mousavi, Schukat, and Howley (2017), this is an active research field to which the emergence of advanced parallel processing technologies could contribute considerably.

5.2. Results for the SCLP-FS

The objective of this second subsection is to investigate the capability of the deep RL model to solve the SCLP while taking into account the full base support constraints. For that purpose, the model presented in Section 4.2 was applied at first to solving the 100 homogeneous instances from BR0 and then the other 100 heterogeneous instances of BR1. Then, the computational results were compared to those of Araya, Guerrero, and Nuñez (2017). For the sake of conciseness, the results of these experiments on 200 instances are shown in the boxplots of Figures 5(a) and 5(b).

The objective herein is to compare the effectiveness of the deep RL when adding full support constraints. Figure 5(a) shows that, as for Araya, Guerrero, and Nuñez (2017), solving the BR0 instances while taking into consideration full support constraints does not degrade the performance of the deep RL model. In fact, the variation in the average mean for BR0, under full support constraints, remains almost the same, *i.e.* less than 1%. This means that, in general, the loading solution found before adding full support constraints was already stable. For the heterogeneous dataset BR1, there was a slight deterioration of the deep RL results. Figure 5(b) displays a slightly larger boxplot when the SCLP-FS is solved. This box size is the interquartile range: equal to upper quantile (Q3) minus the lower quantile (Q1). This larger size of the boxplot when adding full support constraints is due to the fact that the RL model assigns negative rewards to all unstable placements, which decreases the search space. Even though the performance of the deep RL model slightly deteriorates when solving the SCLP-FS for BR1 instances, the effect of the boxes' heterogeneity remains important. Actually, the deep RL model is more sensitive to boxes' heterogeneity than to full support constraints; and this is easily detected when comparing the overlapping in the boxplots shown in Figures 5(a) and 5(b). In Figure 5(a), boxplots given by the RL model overlap with that given by Araya, Guerrero, and Nuñez (2017); however, in Figure 5(b) with heterogeneous boxes, the solutions of the RL model for both problems deteriorate considerably.



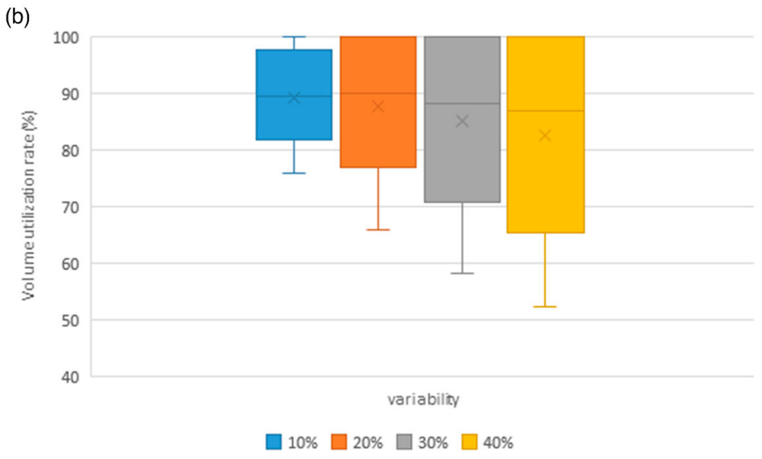
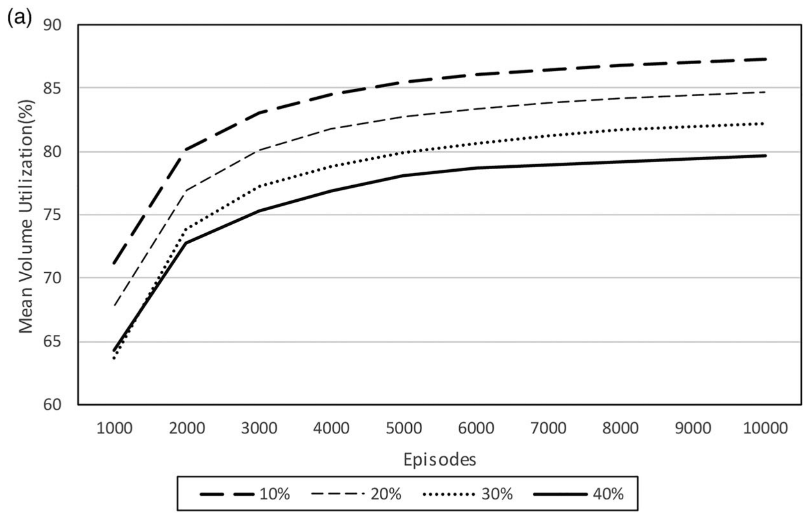


Figure 6. Box-and-wisker plots for BR0-02 (a) and for BR0-05 (b).

5.3. *Effect of boxes number variability*

The purpose of this subsection is to investigate the effect of training the model with a variable number of boxes for the SCLP-FS. In order to address this issue, experiments were conducted while considering four different scenarios with an increasing level of variability, *i.e.* Coefficients of Variation (CVs) equal to 10%, 20%, 30% and 40%, respectively. Recall that the CV is the standard deviation divided by the mean. Then, for each instance, the number of boxes is now assumed to follow a Uniform distribution $U([a, b])$. The parameters a and b, of each probability distribution, are calculated according to the mean as the deterministic number of boxes in the original instance and the variance deduced from the corresponding level of variability. Detailed statistics of the mean fill rate of the two instances BR0-02 and BR0-05 are provided in the online supplemental data and the corresponding plots are shown in Figures 6(a) and 6(b). It is worthy of mention that the same results have been found for all instances tested. Actually, it is obvious that an increase in the variability level induces higher dispersion (Q3-Q1) in the volume utilization, *i.e.* the filling rate. This is demonstrated by the container dimensions provided in the benchmark instances which are generally designed to accommodate a number of boxes near to the mean . However, it is important to notice how the Q1 deteriorates considerably for both instances, *i.e.* the CV increases from 10% to 40%. For the instance BR0-02, the Q1 decrease rate is equal to 18.25% and for the instance BR0-05 it is equal to 16.53%. This means that training the model with a higher number of boxes leads to poorer performance.



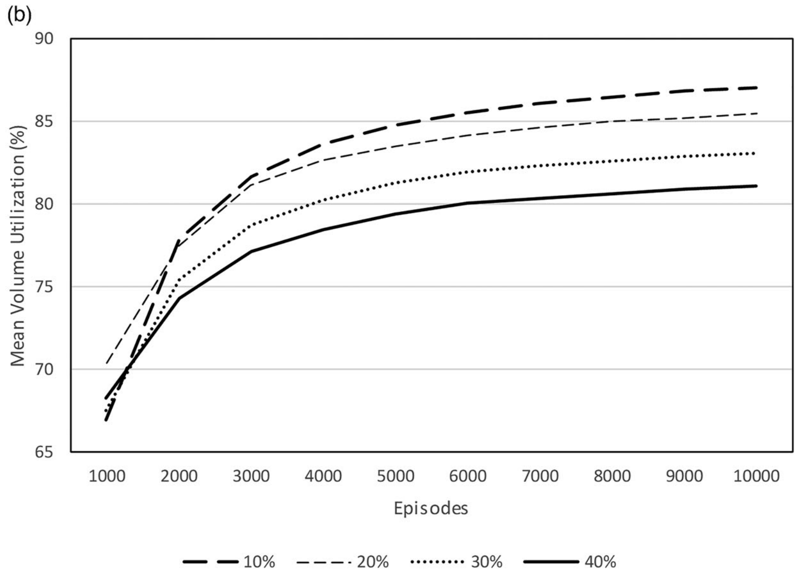


Figure 7. Mean volume utilization under different variabilities for BR0-02 (a) and mean volume utilization under different variabilities for BR0-05 (b).

Then, the effect of different CV levels on the training curves was investigated. Figures 7(a) and 7(b) provide the corresponding plots. More precisely, Figures 7(a) and 7(b) illustrate that the deep RL exhibits the same training curve as for the deterministic case discussed in Section 5.2. Actually, the model converges even though it is trained with a highly variable number of boxes. However, when the variability is high (CV = 40%), it is noticeable that the mean volume utilization decreases. From a practical point of view, this means that, if the company does not face high and unpredictable demand volatility, which implies high variability of the number of boxes, it is more profitable to train the deep RL model with a low level of variability, *i.e.* less than 20%.

In summary, the experiments conducted have shown the capability of the proposed deep RL model to solve challenging SCLPs. In terms of problem size, the model was able to find good solutions for small- as well as large-size instances, *i.e.* with a high number of boxes. Furthermore, computational results have shown the good generalization capability of the proposed model, since it was able to solve instances generated according to different data distributions. In fact, the same deep RL model was used, with a single set of hyper-parameters, to solve both homogeneous and then heterogeneous classes of instances. This generalization ability is a highly desired property of machine learning models, since it overcomes both the need to re-optimize algorithms as well as the deficiencies of tedious parameter tuning. As stated in Bengio, Lodi, and Prouvost (2021), Li *et al.* (2021) and Wang and Tang (2021), even though in terms of optimality these recently proposed learning-based optimization approaches are still less effective than heuristic algorithms, they have become highly attractive owing to their generalization ability. With such an ability, their implementation in real-world applications becomes more straightforward.

6. Conclusion

This work has presented a new deep RL model to solve the basic SCLP as well as the SCLP-FS. At first, the RL model was used to solve both small and large homogeneous instances of the SCLP. Computational experiments performed on benchmark instances indicated that the model gives fairly good results compared to those found with well-known state-of-the-art heuristics. However, the results showed that, in terms of optimality, the quality of the deep RL solutions cannot compete with the well-designed heuristics proposed in the operations research literature. Actually, this limitation of RL models has been encountered in many recent works aiming to solve combinatorial optimization using solely RL models (Vesselinova *et al.* 2020; Wang and Tang 2021). Then, the deep RL model was adapted to solve the more constrained SCLP-FS. The experiments on benchmark instances demonstrated that the proposed deep RL approach still yields fairly good results on homogeneous datasets. Finally, in order to benefit from offline training, the deep RL model was used to solve the SCLP-FS under an uncertain number of boxes. Experimentation with different levels of variability revealed that training the model with an uncertain number of boxes is profitable, since it will not require neither higher computational efforts nor hyper-parameter tuning. Nevertheless, for better performance, it is preferable to train the model with a low level of variability.

An interesting future research direction would be to enhance the quality of the solutions in terms of optimality, and for that purpose it is possible to combine operations research heuristics with machine learning. In order to establish such a combination, the example of a promising framework proposed by Lu, Zhang, and Yang (2019) has achieved new state-of-the-art results for the widely investigated capacitated vehicle routing problem.

Disclosure statement

No potential conflict of interest was reported by the author(s)

Data availability statement

The data that support the findings of this study are openly available at https://doi.org/10.6084/m9.figshare.17304767.v1.

ORCID

Amel Jaoua http://orcid.org/0000-0003-3260-8048
Safa Bhar Layeb http://orcid.org/0000-0003-2536-7872

References

Allen, S. D., E. K. Burke, and J. Mareček. 2012. "A Space-Indexed Formulation of Packing Boxes Into a Larger Box." *Operations Research Letters* 40 (1): 20–24.
Araya, I., K. Guerrero, and E. Nuñez. 2017. "VCS: A New Heuristic Function for Selecting Boxes in the Single Container Loading Problem." *Computers & Operations Research* 82: 27–35.

Araya, I., and M. C. Riff. 2014. “A Beam Search Approach to the Container Loading Problem.” *Computers & Operations Research* 43: 100–107.

Bayraktar, T., F. Ersöz, and C. Kubat. 2021. “Effects of Memory and Genetic Operators on Artificial Bee Colony Algorithm for Single Container Loading Problem.” *Applied Soft Computing* 108: Article ID 107462. doi:10.1016/j.asoc.2021.107462.

Bengio, Y., A. Lodi, and A. Prouvost. 2021. “Machine Learning for Combinatorial Optimization: A Methodological Tour D’Horizon.” *European Journal of Operational Research* 290 (2): 405–421.

Bischoff, E. E., and M. S. W. Ratcliff. 1995. “Issues in the Development of Approaches to Container Loading.” *Omega* 23 (4): 377–390.

Castellucci, P. B., F. M. Toledo, and A. M. Costa. 2019. “Output Maximization Container Loading Problem with Time Availability Constraints.” *Operations Research Perspectives* 6: 100–126.

Da Silva, E. F., A. A. S. Leão, F. M. B. Toledo, and T. Wauters. 2020. “A Matheuristic Framework for the Three-Dimensional Single Large Object Placement Problem with Practical Constraints.” *Computers & Operations Research* 124: Article ID 105058. doi:10.1016/j.cor.2020.105058.

Davies, A. P., and E. E. Bischoff. 1999. “Weight Distribution Considerations in Container Loading.” *European Journal of Operational Research* 114 (3): 509–527.

Dereli, T., and G. S. Das. 2011. “A Hybrid ‘Bee(s) Algorithm’ for Solving Container Loading Problems.” *Applied Soft Computing* 11 (2): 2854–2862. doi:10.1016/j.asoc.2010.11.017.

Fanslau, T., and A. Bortfeldt. 2010. “A Tree Search Algorithm for Solving the Container Loading Problem.” *INFORMS Journal on Computing* 22 (2): 222–235.

Fasano, G. 1999. “Cargo Analytical Integration in Space Engineering: A Three-Dimensional Packing Model.” In *Operational Research in Industry*, 232–246. London: Palgrave Macmillan.

Gonçalves, J. F., and M. G. Resende. 2012. “A Parallel Multi-Population Biased Random-Key Genetic Algorithm for a Container Loading Problem.” *Computers & Operations Research* 39 (2): 179–190.

Gupta, J. N., A. Majumder, and D. Laha. 2019. “Flowshop Scheduling with Artificial Neural Networks.” *Journal of the Operational Research Society* 71 (10): 1619–1637. doi:10.1080/01605682.2019.1621220.

He, K., and W. Huang. 2011. “An Efficient Placement Heuristic for Three-Dimensional Rectangular Packing.” *Computers & Operations Research* 38 (1): 227–233.

Junqueira, L., R. Morabito, and D. S. Yamashita. 2012. “Three-Dimensional Container Loading Models with Cargo Stability and Load Bearing Constraints.” *Computers & Operations Research* 39 (1): 74–85.

Kilincci, O., and E. Medinoglu. 2021. “An Efficient Method for the Three-Dimensional Container Loading Problem by Forming Box Sizes.” *Engineering Optimization*. 16pp. Advance online publication. doi:10.1080/0305215X.2021.1913734.

Kurpel, D. V., C. T. Scarpin, J. E. P. Junior, C. M. Schenekemberg, and L. C. Coelho. 2020. “The Exact Solutions of Several Types of Container Loading Problems.” *European Journal of Operational Research* 284 (1): 87–107.

Layeb, S. B., O. Jabloun, and A. Jaoua. 2017. “New Heuristic for the Single Container Loading Problem.” *International Journal of Economics & Strategic Management of Business Processes* 8 (1): 1–7.

Li, B., G. Wu, Y. He, M. Fan, and W. Pedrycz. 2021. “An Overview and Experimental Study of Learning-Based Optimization Algorithms for Vehicle Routing Problem.” arXiv:2107.07076.

Lim, A., H. Ma, C. Qiu, and W. Zhu. 2013. “The Single Container Loading Problem with Axle Weight Constraints.” *International Journal of Production Economics* 144 (1): 358–369.

Lim, A., B. Rodrigues, and Y. Wang. 2003. “A Multi-Faced Buildup Algorithm for Three-Dimensional Packing Problems.” *Omega* 31 (6): 471–481.

Liu, J., Y. Yue, Z. Dong, C. Maple, and M. Keech. 2011. “A Novel Hybrid Tabu Search Approach to Container Loading.” *Computers & Operations Research* 38 (4): 797–807.

Lu, H., X. Zhang, and S. Yang. 2019, September. “A Learning-Based Iterative Method for Solving Vehicle Routing Problems.” In *Proceedings of the Eighth International Conference on Learning Representations (ICLR 2020)*. ICLR. https://openreview.net/forum?id=BJe1334YDH.

Mazyavkina, N., S. Sviridov, S. Ivanov, and E. Burnaev. 2020. “Reinforcement Learning for Combinatorial Optimization: A Survey.” arXiv:2003.03600.

Mladenović, S., S. Zdravković, S. Vesković, S. Janković, Ž. Đorđević, and N. Đalić. 2019. “Development of a Novel Freight Railcar Load Planning and Monitoring System.” *Symmetry* 11 (6): 756. doi:10.3390/sym11060756.

Mousavi, S. S., M. Schukat, and E. Howley. 2017. “Deep Reinforcement Learning: An Overview.” In *Proceedings the SAI Intelligent Systems Conference (IntelliSys) 2016*, 426–440. Vol. 16 of *Lecture Notes in Networks and Systems*. Cham, Switzerland: Springer. doi:10.1007/978-3-319-56991-8_32.

Olsson, J., T. Larsson, and N. H. Quttineh. 2020. “Automating the Planning of Container Loading for Atlas Copco: Coping with Real-Life Stacking and Stability Constraints.” *European Journal of Operational Research* 280 (3): 1018–1034.

Padberg, M. 2000. “Packing Small Boxes Into a Big Box.” *Mathematical Methods of Operations Research* 52 (1): 1–21.

Parreño, F., R. Alvarez-Valdés, J. F. Oliveira, and J. M. Tamarit. 2010. “Neighborhood Structures for the Container Loading Problem: A VNS Implementation.” *Journal of Heuristics* 16 (1): 1–22.

Parreño, F., R. Alvarez-Valdés, J. M. Tamarit, and J. F. Oliveira. 2008. "A Maximal-Space Algorithm for the Container Loading Problem." *INFORMS Journal on Computing* 20 (3): 412–422.
Pisinger, D. 2002. "Heuristics for the Container Loading Problem." *European Journal of Operational Research* 141 (2): 382–392.
Sahun, Y. S. 2020. "Perspective Directions of Artificial Intelligence Systems in Aircraft Load Optimization Process." In *Handbook of Research on Artificial Intelligence Applications in the Aviation and Aerospace Industries*, 419–437. Lancaster, PA: IGI Global.
Saraiva, R. Dias, N. Nepomuceno, and P. Rogério Pinheiro. 2019. "A Two-Phase Approach for Single Container Loading with Weakly Heterogeneous Boxes." *Algorithms* 12 (4): Article ID 67. doi:10.3390/a12040067.
Scheithauer, Guntram. 2017. *Introduction to Cutting and Packing Optimization: Problems, Modeling Approaches, Solution Methods*. Vol. 263 of the *International Series in Operations Research & Management Science* book series. Cham, Switzerland: Springer International Publishing. doi:10.1007/978-3-319-64403-5.
Vengerov, D. 2007. "A Reinforcement Learning Approach to Dynamic Resource Allocation." *Engineering Applications of Artificial Intelligence* 20 (3): 383–390.
Vesselinova, N., R. Steinert, D. F. Perez-Ramirez, and M. Boman. 2020. "Learning Combinatorial Optimization on Graphs: A Survey with Applications to Networking." *IEEE Access* 8: 120388–120416.
Wang, Q., and C. Tang. 2021. "Deep Reinforcement Learning for Transportation Network Combinatorial Optimization: A Survey." *Knowledge-Based Systems* 233: Article ID 107526. doi:10.1016/j.knosys.2021.107526.
Xiang, X., C. Yu, H. Xu, and S. X. Zhu. 2018. "Optimization of Heterogeneous Container Loading Problem with Adaptive Genetic Algorithm." *Complexity* 2018: 1–12.
Xu, X., L. Zuo, and Z. Huang. 2014. "Reinforcement Learning Algorithms with Function Approximation: Recent Advances and Applications." *Information Sciences* 261: 1–31.
Zhang, D., Y. Peng, and S. C. Leung. 2012. "A Heuristic Block-Loading Algorithm Based on Multi-Layer Search for the Container Loading Problem." *Computers & Operations Research* 39 (10): 2267–2276.
Zhu, W., and A. Lim. 2012. "A New Iterative-Doubling Greedy-Lookahead Algorithm for the Single Container Loading Problem." *European Journal of Operational Research* 222 (3): 408–417.
Zhu, W., W. C. Oon, A. Lim, and Y. Weng. 2012. "The Six Elements to Block-Building Approaches for the Single Container Loading Problem." *Applied Intelligence* 37 (3): 431–445.