Operations Research Forum (2024) 5:20
https://doi.org/10.1007/s43069-024-00301-3

ORIGINAL RESEARCH

gym-flp: A Python Package for Training Reinforcement Learning Algorithms on Facility Layout Problems

Benjamin Heinbach[1] · **Peter Burggräf**[1] · **Johannes Wagner**[1]

Received: 27 March 2021 / Accepted: 29 January 2024 / Published online: 5 March 2024

Abstract
Reinforcement learning (RL) algorithms have proven to be useful tools for combinatorial optimisation. However, they are still underutilised in facility layout problems (FLPs). At the same time, RL research relies on standardised benchmarks such as the Arcade Learning Environment. To address these issues, we present an open-source Python package (gym-flp) that utilises the OpenAI Gym toolkit, specifically designed for developing and comparing RL algorithms. The package offers one discrete and three continuous problem representation environments with customisable state and action spaces. In addition, the package provides 138 discrete and 61 continuous problems commonly used in FLP literature and supports submitting custom problem sets. The user can choose between numerical and visual output of observations, depending on the RL approach being used. The package aims to facilitate experimentation with different algorithms in a reproducible manner and advance RL use in factory planning.

Keywords Combinatorial optimisation · Artificial intelligence · OpenAI · Production management · Factory planning · Simulation

1 Introduction

Facility layout problems (FLP) are an important class of optimisation problems in operations research (OR). The FLP constitutes a fundamental aspect of operations research and industrial engineering, addressing the strategic arrangement of compo-

✉ Benjamin Heinbach
benjamin.heinbach@uni-siegen.de

Peter Burggräf
peter.burggraef@uni-siegen.de

Johannes Wagner
johannes.wagner@uni-siegen.de

[1] International Production Engineering and Management, University of Siegen, Siegener Straße 152, Kreuztal 57223, Germany

nents within a facility to optimise efficiency, reduce costs, and amplify productivity. The optimal layout design significantly impacts material flow, transportation costs, and worker performance [1].

The FLP is NP-hard and has been tackled using an extensive array of modelling and resolution techniques [1–4]. In recent years, the use of Machine Learning (ML) techniques has experienced a surge for other important tasks in manufacturing, e.g. [5–10]. Special consideration has been conveyed to Reinforcement Learning (RL), a specific sub-class of ML, in manufacturing and supply chain research [11–14]. For combinatorial optimisation problems, provided sufficient training resources and a supervised initialisation, RL approaches bear the potential to generalise on new instances if the training was performed on a small distribution of the problem to exploit its structure [15].

Building upon these insights, we infer that facility layout planning systems based on RL will achieve a level of proficiency comparable to that of human layout planners. A fundamental premise to consider is that the distinctive attributes of DRL, such as the potent feature extraction and function approximation capabilities offered by Convolutional Neural Networks (CNN), can potentially be harnessed and extended to the realm of facility layout planning. Thus, when being trained on a sufficiently large distribution of FLPs, the definition of which is yet to be made, trained DRL agents might be able to make proposals for factory layouts while eliminating the need for training new RL agents and modelling problems, thus facilitating and speeding up facility layout planning.

To address this, we introduce an open-source Python package providing a standardised interface that allows researchers to leverage commonly used FLP benchmark problems for training and testing RL algorithms in a comparable and reproducible manner. Its purpose is not to compete with existing heuristics in FLP research but to provide a curated set of both discrete and continuous well-studied FLP problems known in literature to decrease the up-front modelling effort and to make it accessible to an FLP research community interested in investigating the practical utility of RL approaches in facility layout design.

The contribution of this paper is threefold: we service the FLP community with a toolbox that (1) provides open-sourced environments for well-known benchmark FLP problems and (2) presents an interface to these environments to reduce implementation efforts, so as to (3) enable comprehensive benchmarking of FLP implementations against readily available RL algorithms. Thus, we hope to assist in showing whether RL can prove to be useful for FLP research and practitioners.

2 Facility Layout Problems

The central objective of Facility Layout Problems involves determining the optimal spatial arrangement of workstations, departments, machines, or equipment within a facility. This arrangement profoundly influences factors such as material handling expenses, production time, worker movement, and overall facility utilisation. FLPs encompass identifying the most favourable positioning of these components while adhering to constraints such as space limitations, safety regulations, and workflow

prerequisites [16, 17]. Typically, one central element of an FLP is to minimise the transport intensity between facilities, given their pairwise flow relationships and distances. This transport intensity is commonly referred to as Material Handling Cost (MHC). Thus, the general optimisation objective of an FLP regarding the MHC is given as follows:

$$minimize : MHC = \sum_{i}^{n} \sum_{\substack{j \\ i<j}}^{n} f_{ij} d_{ij} c_{ij} \tag{1}$$

where f_{ij} denotes the flow relationship between facilities i and j, d_{ij} is the distance between i and j, usually measured from their centre point, and c_{ij} is a scaling factor to account for different modes of transportation in between facilities.

A central aspect of solving Facility Layout Problems revolves around the mathematical formulation of the problem itself as the choice of formulation governs the complexity of the solution. FLPs can be classified into discrete and continuous formulations based on the treatment of spatial arrangement variables. In *discrete* FLPs, the space is divided into a finite number of predefined locations, often termed grid points or nodes. Each component or workstation is assigned to one of these discrete locations. In *continuous* FLPs, the placement of components is treated as a continuous variable, allowing flexibility in layout design. Components can be placed at any point in the continuous space [1].

Discrete FLPs are often modelled as Quadratic Assignment Problem (QAP). The QAP deals with assigning a set of facilities to a set of locations in a way that minimises the sum of weighted distances between facility pairs [18]. This combinatorial optimisation problem finds applications in fields ranging from manufacturing to telecommunications.

Continuous FLPs, in turn, are modelled with various degrees of freedom: the Flexible Bay Structure (FBS) notation has been created by Tong [19]. It allows the departments to be located only in parallel bays with varying widths. Bays are bounded by straight aisles on both sides, and departments are not allowed to span over multiple bays [20]. The width of the bays is determined by the cumulated area demand of the facilities assigned to each one using the equation below [21]. The Slicing Tree Structure (STS) decomposes irregular shapes into simpler rectangles, easing the layout process. It involves iteratively partitioning a larger rectangular area into smaller rectangles through horizontal and vertical cuts. This method aims to find an arrangement of smaller rectangles that optimises the given layout objective [22]. Lastly, the Open Field Layout Problem (OFP) involves designing the layout for facilities with vast open spaces, such as warehouses or large manufacturing areas. The objective is to optimise material movement and accessibility without the restrictions or constraints that would be imposed by such arrangements as a single row or loop layout. Instead, a key concern of the OFP is that facilities are to be arranged free of overlaps [23].

Historically, various strategies have been employed to tackle Facility Layout Problems, reflecting diverse problem formulations and solution methods [2]. Classical techniques, such as the systematic layout planning (SLP) approach, emphasise human factors, expertise, and experience to create efficient layouts. In the SLP approach,

qualitative and quantitative considerations are combined to generate layout alternatives. However, such methods might fall short in complex environments with numerous variables and constraints [24].

Other prominent approaches are mathematical optimisation techniques like integer programming, genetic algorithms, simulated annealing, and particle swarm optimisation. These methods leverage computational power to systematically search for optimal or near-optimal solutions, taking into account various constraints and objectives simultaneously. These algorithms have proven effective in generating layouts that minimise transportation costs, minimise material handling time, and maximise facility throughput [25].

More recent approaches make use of ML tools. Despite some evidence of using Artificial Neural Networks in the facility planning approach, see for instance [26–28], a recent study has shown that ML methods, and RL specifically, have seen comparably little use in FLP research [4]. This is remarkable since RL has been employed to solve several real-life industry-related problems such as electric energy storage systems [29], job-shop scheduling [5, 6, 30, 31], autonomous guided vehicle routing [32], or the travelling salesman problem [33]. Contrary to this lack of application evidence, recent work [34–40] demonstrates the need for and interest in the application of RL to FLPs.

3 Reinforcement Learning

Reinforcement Learning has gained prominence due to significant research advancements in creating learning agents capable of excelling in virtual arcade game environments [41]. RL, most notably its deep learning variant for higher order problems, Deep Reinforcement Learning (DRL), has been demonstrated to outperform expert human players in board games [42] and computer games by learning merely from visual input [41].

In RL, an agent learns to maximise a reward signal by exploratively interacting with its environment. The goal of the agent is to learn a policy that maps states of the environment to actions in such a way as to maximise the expected discounted cumulative rewards over time:

$$E_\pi[R_t] = E_\pi[\sum_{i=1}^{\infty} \gamma^i r_{t+i}] \tag{2}$$

where $\gamma \in (0, 1]$ is a discount factor that governs whether rewards at later training progress are treated myopic or farsighted.

A reinforcement learning problem can be formally defined as a Markov decision process (MDP) which mathematically translates problems into a sequential decision-making process. An MDP consists of:

- A set of states, S, that represent the possible configurations of the environment.
- A set of actions, A, that the agent can take in each state.

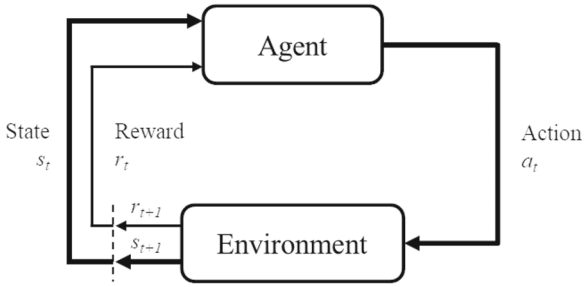

Fig. 1 Representation of the MDP [43]

- A reward function, $R : S \times A- > R$, which assigns a reward to each state-action pair.
- A transition function, $T : S \times A- > S$, which specifies the next state that results from taking a given action in a given state.

The agent begins in an initial state and at each time step t, selects an action a according to a policy π and transitions to a new state. The reward at each time step is determined by the state-action pair and the resulting state is determined by the transition function [43]. A representation of the MDP is shown in Fig. 1.

In reinforcement learning, there are two main approaches to learning a policy: model-based and model-free.

Model-based RL involves learning a model of the environment, which allows the agent to make predictions about the consequences of its actions. With this model, the agent can plan its actions using a search algorithm such as value iteration or policy iteration. The advantage of model-based RL is that it can learn an optimal policy more efficiently since it can use its model to predict the outcomes of actions and plan accordingly. However, it can be difficult to learn an accurate model of the environment, especially if the environment is complex or the state space is large. Model-free RL, on the other hand, does not involve learning a model of the environment. Instead, the agent directly learns a policy by interacting with the environment and receiving rewards. Model-free reinforcement learning is simpler to implement and can learn directly from raw sensory data, but it can take longer to learn an optimal policy compared to model-based methods [43].

Two other fundamental approaches within RL are value-based and policy-based methods. Value-based methods focus on estimating the value of taking various actions in different states. They aim to learn a value function, such as the Q-function in Q-learning, which assigns a value to each state-action pair. Agents then make decisions by selecting actions with the highest estimated value. In contrast, policy-based methods aim to directly learn the optimal policy, which is a mapping from states to actions. Instead of estimating the value of actions, policy-based methods adjust the policy itself to maximise the expected cumulative reward [43].

Furthermore, reinforcement learning algorithms can be categorised into on-policy and off-policy methods. On-policy algorithms learn and improve the policy they currently follow. This means that the data used for learning must be collected using the current policy, which can limit exploration. Off-policy algorithms, on the other hand,

allow an agent to learn from data generated by a different policy, enabling more efficient exploration and potentially better sample efficiency. Popular examples of off-policy algorithms include Q-learning and off-policy actor-critic methods [44].

4 Leveraging Reinforcement Learning for Facility Layout Problems

In the contemporary landscape of optimisation challenges, the paradigm of RL emerges as a compelling alternative to tackle the intricacies of FLPs. With an inherent capacity to learn, adapt, and optimise over time, RL brings to the fore a dynamic approach that resonates with the multi-faceted nature of FLPs.

A hallmark of RL is its intrinsic inclination toward exploration and exploitation. This trait holds profound relevance in the context of FLPs, where sub-optimal solutions might often be disguised as local optima. RL algorithms can navigate this challenge by leveraging exploration mechanisms, thereby unearthing novel layout configurations that can lead to substantial performance enhancements. Adaptive heuristics embedded within RL techniques continuously refine strategies, ensuring a balance between tried-and-tested methods and novel explorations.

Yet another distinctive characteristic of RL, more precisely DRL, is the utilisation of Convolutional Neural Networks (CNNs) as described by [45]. CNNs excel at extracting spatial features from structured data. FLPs involve spatial configurations of various components such as layout maps or images depicting the facility. By integrating CNNs into the Reinforcement Learning framework for FLPs, CNNs can learn to generalise spatial patterns from one facility layout to others, enhancing the transferability of learned policies. This can be particularly valuable in scenarios where similar layout structures occur across different facilities.

One key challenge to be addressed is to design the problem representations and underlying Markov Decision Processes (MDP) in a way that they can accommodate different sorts of problem types and become less sensitive to problem sizes. By submitting fixed-size input information to the RL agent, i.e. an image with the same amount of pixels in both directions regardless of the number of facilities, one is able to assume control over the state space dimensionality.

The underlying hypothesis driving the development of this software tool is that the synergy between spatial feature extraction and policy optimisation can lead to more accurate, adaptive, and efficient solutions. CNNs empower RL agents to navigate the complexities of spatial arrangement while accounting for visual cues and correlations inherent in facility layouts.

The Python package presented herein aims to achieve this synergy by providing a visual representation of the FLP to the RL agent. We do so by encoding the flow intensity matrix, both row and colour-wise, in the colour channels of an image. The flow intensity matrix contains information on both the flows and distances from Eq. 2 where rows can be interpreted as flow sources and columns as sinks, respectively. By encoding the flow intensity information, the RL algorithm should be able to abstract structural information of the problem. That is, the machine with the lowest transport flows both in- and out-bound will be shown as dim red (e.g. RGB=(40, 0, 0)) whereas machines with higher transport relationships will move closer to purple or even white

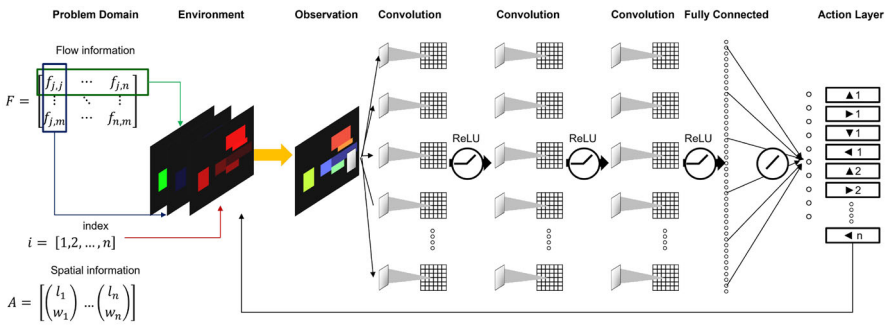

Fig. 2 Illustration of the concept of embedding FLP information in visual representations to make them digestible by Convolutional Neural Networks. Image adapted from [41, 47]

((40, 255, 255) or (255, 255, 255)). Figure 2 visualises this concept, the details of which are outlined in Section 5.3. This functionality, coupled with the respective reward assignment, should enable an RL agent to—at least—reproduce basic heuristics, such as 'place the machine with highest flows in the centre and arrange all others around it', similar to the triangular placement method of [46].

As [48] have pointed out, a key objective for using DRL in manufacturing research and practice is to reduce manual involvement. With facility layout planning typically being a highly manual process, the exceptional representation learning capabilities of CNNs combined with the adaptive heuristics inherent to RL techniques can position them as a transformative force in contemporary FLP research.

5 Tying it Together: The gym-flp Library for Using RL on FLPs

5.1 The Backbone: OpenAI Gym

In this section, we introduce the Python package `gym-flp` that can be used to train Reinforcement Learning agents on common or custom FLP instances. This software library is at https://github.com/BTHeinbach/gym-flp.

The library we propose in this work is based on OpenAI Gym, a toolkit designed for developing and comparing RL algorithms. Gym provides researchers with access to benchmarks for training agents in the form of *environments* [49]. These environments include control problems, arcade games, and robotics. The availability of such benchmarks has played a significant role in advancing the field of RL [50]. In fact, open-source libraries in general play a vital role in evaluating innovations in the field of algorithms using benchmarks on a variety of problems [51].

While originally developed for RL research, there are several examples of how OpenAI Gym has been used to address real-world research questions, such as controlling power systems on grid [52, 53] or on building level [54], job-shop scheduling problems [55], simulated robotic motion (e.g. pick-and-place operations) [56], industrial process control (e.g. valve settings for gas turbines or pitch angles and rotor speeds for wind turbines) [50], or communication technology optimisation [57].

The potential for OpenAI Gym in OR has also been recognised: [58] published OR-Gym, a Gym-based library that features the common OR problem types knapsack, bin packing, supply chain, vehicle routing, news vendor, portfolio optimisation, and travelling salesman problems. Similar to the contribution presented herein, they aim to encourage further development and integration of RL into optimisation and the OR community while also opening the RL community to many of the problems and challenges that the OR community has been wrestling with for decades. Nonetheless, despite being a special problem in OR research, their library does not support FLPs.

The examples given above provide substantial evidence that OpenAI Gym is a prudent choice for creating frameworks for RL usage in FLP research. For our intended purposes, we reversed the Gym logic by putting the development focus on the environments rather than the algorithms by encouraging the use of collections of common RL algorithms, such as Stable Baselines [59] or Ray [60]. Effectively, we assert that this can speed up RL research on FLPs since, as a first step, researchers can resort to selecting a resolution technique from a well-established suite of algorithms to use on their problems before having to develop new or amend existing algorithms.

5.2 Package Structure

The implemented environments in this work are Python classes that inherit from the OpenAI Gym base class `gymEnv` and follow the same conventions regarding required class methods. This is because certain RL algorithm frameworks include a method for checking that the inputs and outputs of the environment are compatible with the agents. Figure 3 shows the outline of the package topography, the explanations of which follow thereafter.

5.3 Implementation Details

For an FLP to be solved using RL, its components need to be translated into the key concepts of an MDP as introduced in Section 3.

5.3.1 Observation Spaces

There are two modes for designing the observation space: `rgb_array` and `human`. The mode can be specified when initialising the environment.

In `'human'` mode, the observation for a discrete Facility Location Problem (FLP) is a permutation vector of size n, sampled randomly without replacement, where n is the problem size of the instance. The permutation vector represents the available locations (numbered consecutively) and the assigned machines. The assumption is that every facility can be assigned to any location. Further restrictions can be implemented by those interested in this functionality.

For continuous FLPs, the observation includes centre point coordinates (x, y), length (l), and width (w) information for each facility in $i = 1...n$, resulting in an observation space of $4 * n$. The range of possible values for an observation is deter-

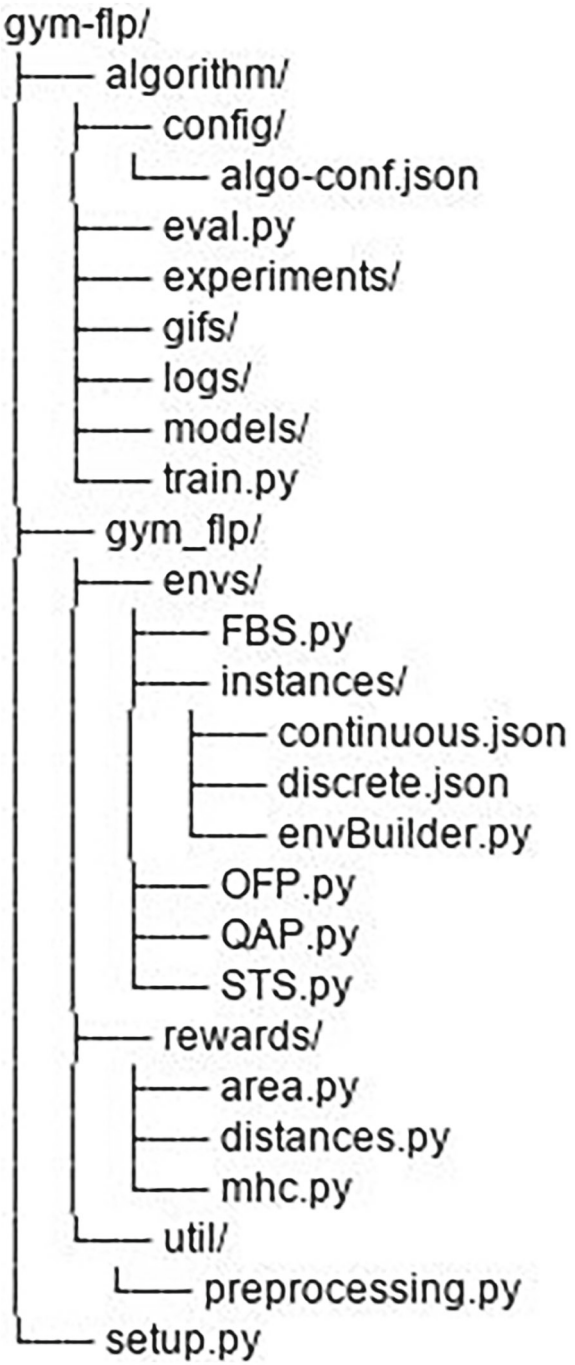

Fig. 3 gym-flp package structure. Module init files were omitted

mined using the `Box` space and is based on problem instance information in the `__init__` function.

One of the main motivations of the authors is to teach RL algorithms to rearrange factory plants using visual input alone, similar to the early advances in RL research on arcade games [45]. To facilitate this, we provide the `rgb_array` mode, in which observations are treated as images with dimensions corresponding to the sizes of the plants. The allowed values for the third dimension are limited to the range [0, 255], and the observation is normalised (divided by 255 to have values in [0, 1]) when using Convolutional Neural Network (CNN) policies. The baseline algorithms implemented in Stable-Baselines3 and RLlib use predefined filter sizes, so the plant sizes may need to be adjusted accordingly on occasion. For example, the `CnnPolicy` in Stable-Baselines3 requires observed images to be at least 36 by 36 pixels in size.

To facilitate feature extraction in CNN-based policies and make all facilities on the plane identifiable, we encode certain information in the RGB colour space of the observation images. The index i of a facility is encoded on the red channel, while the row sums of the flow matrix F (representing how much a facility is a flow source) are embedded in the green channel and the column sums of F (i.e. the sinks) are written into the blue channel. The underlying idea is that facilities with higher green and blue values have stronger flow relationships and may be located at the centre of all facilities, potentially allowing an RL algorithm to learn this rule through intuition or heuristics.

Figure 4 shows a visual representation of the `rgb_array` observations for each environment.

Note that the observation mode has an impact on the policy used by the algorithm. For instance, the use of human mode forbids using `CnnPolicy` as convolutional filters do not work on one-dimensional arrays.

5.3.2 Action Spaces

Defining proper action spaces is likely to be as crucial in RL as reward engineering. This section explains the predefined implementations, yet we encourage all interested researchers to amend the actions for their respective purposes.

Aside from `Box` used for the observation spaces, `gym.Spaces` provides several other spaces, e.g. `Discrete`, `MultiDiscrete`, `Dict`, `Tuple`, `Graph`, `MultiBinary`, `Sequence` and `Text`. To define the possible range of actions, `gym-flp` currently uses `Box`, `Discrete`, and `MultiDiscrete`.

Table 1 shows which spaces are supported in the respective environments.

In `qap-v0`, the action space represents a pairwise exchange mechanism, where two facilities i and j will swap their currently assigned locations. We also supply a swap (i, j) to provide an action to remain in the current state. Since the swap (i, j) is identical to (j, i), the size of the action space is given as $n - n * 0.5 + 1$.

`fbs-v0` incorporates five discrete actions: *Permute* swaps two random positions in permutation vector, *Bit Swap* flips the value (0/1) of one random element in the bay break vector. With *Bay Exchange*, two randomly selected bays exchange their facilities. *Inverse* inverts the order of facilities in a randomly selected bay, and *Idle* does nothing, again to provide the agent with an action that allows it to remain in a state it deems optimal.

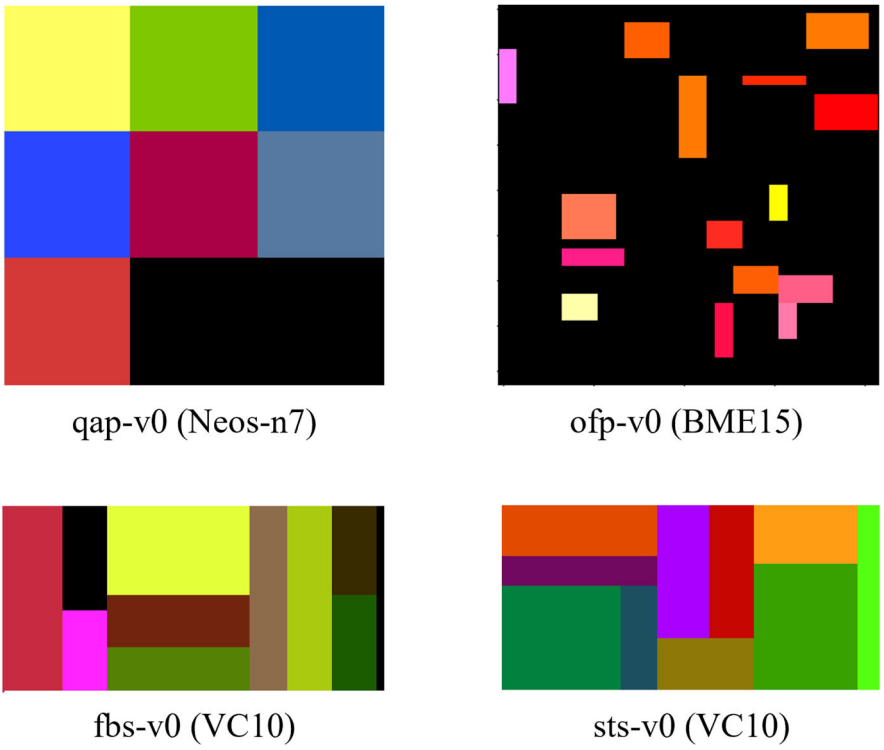

Fig. 4 Examples for environment renderings. The text in brackets describes the instance used

`sts-v0` uses five discrete actions as well. In addition to *Permute* and *Idle*, the action space uses *Slice Swap* (swap two random positions in slicing order), *Bit Swap* (change *slicing* orientation at random position) and *Shuffle* (Create new random slicing order).

To date, the action spaces for `ofp-v0` are the most comprehensive. The `Discrete` variant is a 1-D array of size $4 * n + 1$, where each facility can take any step in the north, east, south, or west direction at a step size that defaults to 1 but can be passed differently upon initialisation. The single additional action is again used as a way to retain the current state. The `MultiDiscrete` action space folds the four actions into a $5 \times n$ 2-D array. Here, all facilities are moved simultaneously in any of the four

Table 1 Summary of currently supported action spaces per environment

Environment	Discrete	Multi-Discrete	Box	Box (simultaneous)
qap-v0	✓	-	-	-
fbs-v0	✓	-	-	-
ofp-v0	✓	✓	✓	✓
sts-v0	✓	-	-	-

directions. The idle action is appended as the fifth option to every dimension. Lastly, there are two options using a `Box` space. Both of them use Cartesian coordinates on the factory plant ranging from point $(0, 0)$ to (Y, X), with Y and X corresponding to the plant width and length, respectively. The `Box` spaces differ in that facilities can be moved sequentially or simultaneously. This decision is passed upon initialisation using the boolean argument `multi`.

5.3.3 Reward Computation

The reward engine comprises up to three different components, depending on the environment and action space used. Since QAP, FBS and STS are collision-free and within defined plant bounds, these environments only handle material handling cost (MHC).

The flows between facilities are stored in the package. The distance metric for continuous problems can be set upon initialisation with the options rectilinear, Euclidean and squared-Euclidean, where rectilinear is the default value. MHC computation is invoked as an instance of the sub-module `rewards.mhc`.

An intricacy to consider is that common RL approaches attempt to maximise cumulative reward whereas the goal in FLP is to minimise transport cost. At the same time, the reward signal should enable generalisation and avoid reward gaming (i.e. the exploitation of an unintended loop-hole) as described in [34]. To address these points, we have set up the reward component for MHC as a moving target. In every step, we compare the MHC at time step t $MHC_{s(t+1)}$ with the best known MHC MHC_{best} of the current episode. A reward of 1 is assigned if the new MHC is lower, or 0 otherwise, see Eq. 3. Then, MHC_{best} is overwritten accordingly.

$$r_{MHC} = \begin{cases} 1, & \text{if } MHC_{s(t+1)} < MHC_{best} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

For the OFP environment, layout feasibility needs to be taken into account. In contrast to the environments where locations are determined, OFP needs to incorporate non-overlapping constraints. We, therefore, define the penalty term $p_{collision}$ (see Eq. 4) that collects a penalty of 1 if one facility intersects the union of the remaining ones. We choose a value of 2 to prevent a sparse reward signal for actions that improved MHC (+1) but resulted in a collision.

$$p_{collision} = \begin{cases} 2 & \text{if } F_i \cap \left(\bigcup_{j=1}^{i-1} F_j \cup \bigcup_{j=i+1}^{n} F_j \right) \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Finally, the variants that rearrange facilities with a movement with a given step size (`Discrete` and `MultiDiscrete`) are at risk of moving facilities beyond the plant boundaries resulting in entering a state beyond the state space. Such actions are penalised in two ways: the current episode is terminated (similar to a game-over situation in arcade games) and a penalty $p_{off-grid}$ of 10 is assigned so as to overrule possible positive rewards from MHC and to be free of collisions, see Eq. 5.

$$p_{off-grid} = \begin{cases} 10 & \text{if } s_{t+1} \notin S \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

In consequence, the total reward per step (in OFP) is given as:

$$r = r_{MHC} - p_{collision} - p_{off-grid} \tag{6}$$

Reward engineering is said to be one of the most important tasks in RL research. Therefore, all researchers using `gym-flp` are encouraged to redefine the reward computation according to their respective needs and suitable for their problems in question. For instance, additional reward or penalty terms can be defined inside `gym-flp`.

5.3.4 Agent

This package uses Stable Baselines 3 [59] as the framework for RL and the algorithms it currently supports (as of version 1.8.0). Thus, the algorithms available off-the-shelf in `gym-flp` are

- Deep Q-Networks (DQN) [45]
- Proximal Policy Optimization (PPO) [61]
- Advantage Actor Critic (A2C) [62]
- Deep Deterministic Policy Gradients (DDGP) [63]
- Twin-Delayed DDPG (TD3) [64], and
- Soft Actor-Critic (SAC) [65]

Prior to training, hyper-parameters for the algorithm are loaded from the algo-conf.json file located inside `./algorithm/config/`. The hyper-parameters are the baseline values as stored in the Stable Baselines 3 classes. These can be altered by

(a) Manually changing the parameters in the configuration file by navigating to the source folder
(b) Passing the parameter values along with train.py, see Section 6.2

When choosing option b), parameters can be passed as a single value for one-off training (see Section 6.2.4) or as two values describing the lower and upper bound for an optimisation run (see Section 6.2.5).

5.3.5 Environment

The environment is the heart of the package and contains the state transition dynamics for the observation and action spaces defined above. The environment follows the Gym convention and thus contains the functions `__init__()`, `reset()`, `step()`, and `render()`.

Environments in Gym are typically *episodic*, i.e. they usually possess a terminal state that aborts an episode and resets the environment. One such terminal state in the context of FLPs could be the optimal state of the layout given the objective function. However, the implementation assumes that no optimum is known a priori. Therefore,

the gym-flp environments are designed as *continuous* tasks. To nonetheless enable the use of episode-based callbacks and evaluations, episodic behaviour needs to be mimicked. We have therefore included the following termination criteria:

- An action leads to a state outside of the state space (applies mostly to OFP)
- No improvement of MHC has been made in a number of consecutive steps

The first criterion prevents the agent from learning actions that produce infeasible layouts and are penalised accordingly, see Section 5.3.3. The second criterion assumes that the agent has reached a region close to a (local) optimum.

6 Usage

This section provides basic examples of how to use the described package with Reinforcement Learning approaches. Following some installation hints, we provide execution commands for training and tuning scripts provided with the package.

6.1 Installation

The procedures below were tested on a Windows 10 OS and assume the usage of an Anaconda, Pycharm or Visual Code distribution. gym-flp requires Python with a version starting from 3.7. The following ways exist to install the package.

6.1.1 PyPi Installation

Gym-flp supports PyPi installation which is the most straightforward means of installation. The package can be conveniently installed by opening a terminal window in the Python home directory and running the command:

pip install gym−flp

The use of virtual environments, such as virtualenv, conda or pipenv is highly encouraged.

6.1.2 Installation from GitHub

The next option is to directly install the package from the GitHub source. This can be achieved by opening a command terminal and running the command:

pip install git+git://github.com/BTHeinbach/gym−flp.git

6.1.3 Cloning GitHub Repository

If all else fails, navigate to the GitHub repository under https://github.com/BTHeinbach/gym-flp and download the .zip package to a destination folder of your choosing. Next, open up a terminal, navigate to the destination folder using the command cd and type the command below (Note the full-stop after the space). This will install the package in editable development mode, allowing easier changes to it.

pip install −e .

The examples below demonstrate the usage and results.

6.2 Basic Experiment Workflow

According to the experiment workflow presented in Fig. 5, in the current version of gym-flp, researchers have the following experimental design options:

- Common problem instance or custom problem
- One-off or optimisation
- Meta-Study or algorithm tuning

The core of the package are the 'train' and 'evaluate' blocks. Train will call evaluate from within to test the agent on the new instance of the environment. But, both can be called individually from a command line interface. Calling 'evaluate' independently can be useful to test an agent with different random seeds.

The training scripts will create a model according to the script name and the arguments passed. Finally, when training is completed, the program will run one episode in the environment until termination and log reward and MHC per step for evaluation. The Tensorboard logs, final and (if applicable) intermediate models, experiment JSON files, and GIFs from the evaluation run are stored in respective sub-folders in `algorithm` by default.

The algorithm scripts make use of Stable Baselines 3 and especially the built-in callback functionality. An evaluation callback will assess the training performance every 10,000 steps for 10 episodes and save a checkpoint of the model if a new best mean reward has been achieved. In the script's evaluation process, the model at the end of training and the best model are used simultaneously.

Since RL algorithms are generally prone to be sample-inefficient and training can take a long time, especially on large problem sets or if parallelisation is not possible, we included the `StopTrainingOnNoModelImprovement` callback which observes the results of the evaluation callback and will abort the training run if no new best model could be found within a predefined time frame (default: three consecutive evaluations).

6.2.1 Experiment Input Options Overview

Gym supports passing optional arguments to the environment. We make use of this opportunity to provide the FLP researcher with more flexibility and less need to access the environments' code for changes. These arguments are technically optional, yet some of them are critical for the behaviour of the environment, and failing to pass them may result in errors being thrown or otherwise unexpected behaviour. Figure 6 summarises the implementation details from Section 5.3.

For training RL algorithms, `gym-flp` provides executable scripts under the package directory *algorithms* that can be executed via IDE or using a command line interface such as a regular terminal in Windows. The available parameters along with their permissible values are shown below. The default value is highlighted in boldface.

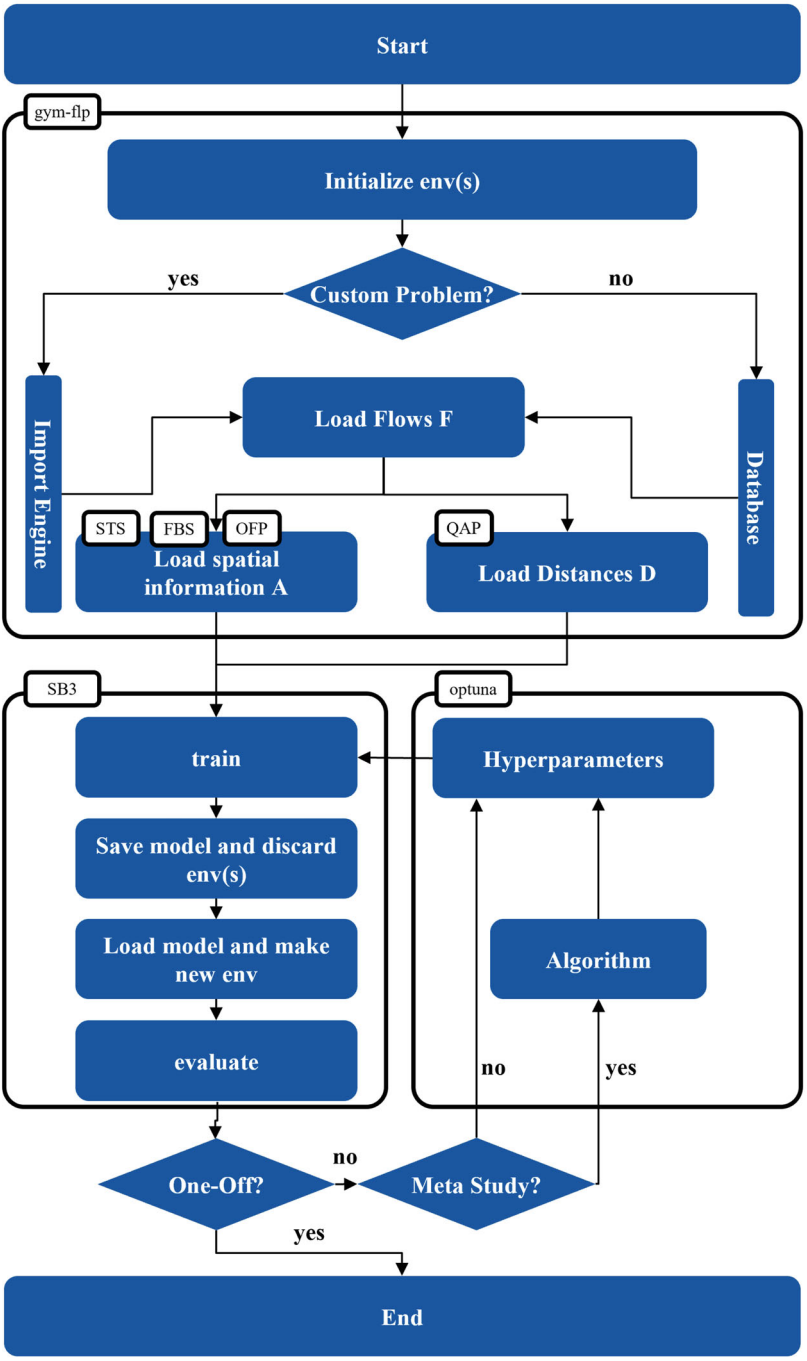

Fig. 5 Basic gym-flp workflow with decision gates

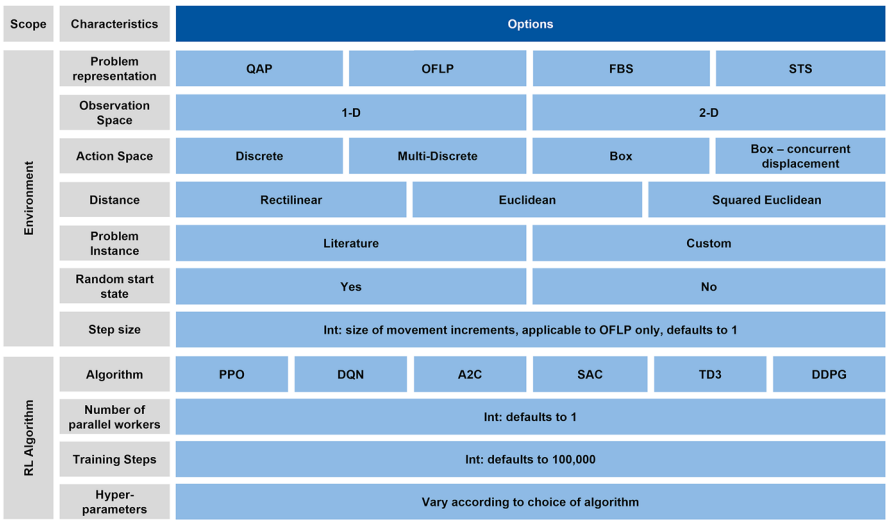

Fig. 6 Morphological Box of experimentation options supported as of gym-flp 0.2.0

- 'mode' (*human* or ***rgb_array***): controls whether observations are output as 1D or 2D arrays.
- 'instance' (**instance name as string**): the instance to be used. If no instance is passed or if it is misspelled, the environment will prompt for new input and provide an environment-specific list of available problem sets.
- 'env' ($qap-v0$, $fbs-v0$, $sts-v0$ or $\boldsymbol{ofp-v0}$): the environment id to train in
- 'distance' (***rectilinear***, $euclidean$ or $squared-euclidean$): the distance metric to be used for MHC calculations
- 'step_size': controls displacement length for discrete steps in $ofp-v0$. Without effect in other environments. Defaults to 1.
- 'box': if passed, will create an actions space of type `Box`. If omitted, a `Discrete` space is created instead.
- 'multi': if passed, will create a variant of the action space defined by the argument 'box' that supports simultaneous displacements.
- 'train_steps': integer value for the number of steps to take in the environment. Defaults to 100.000.
- 'num_workers': integer value for the number of parallel environments (where the algorithm provides it). Defaults to 1.
- 'algo': Name of the algorithm used for logging. If omitted, the program will use the script name instead
- 'randomize': If set, `reset()` will assign facilities to random locations. If omitted, machines will be distributed in a reproducible manner

6.2.2 Using Problem Sets from Literature

To perform studies that compare the performance of RL approaches to that of solutions presented in contemporary literature, we provide the flow (and distance or area) information for problems commonly used in literature. We implemented 138 QAPs available from the QAPLIB [66] and a plethora of continuous problems which were taken from the compilation of [67].

It is important to note that all continuous problems are available for the respective implementations as explained in Section 5.3, but their suitability may vary. For example, problem sets where the available plant space matches the sum of facility areas may not be suitable for the greenfield scenario, as the environment may fail to find a collision-free layout. On the other hand, instances where area requirements greatly undershoot plant space availability may result in empty spaces in the RGB image representations, which can hinder computation efficiency when using image recognition policies.

For all problem sets, we deliberately chose not to include currently known lowest bounds or optimal solutions from academic literature as we consider these a 'moving target' with no true added value for this package. The reason is that we intend to also solve real industry problems without any solution known a priori. However, we note and explicitly encourage any interested researcher to adapt the open-source package by, for instance, implementing an episodic approach that terminates upon reaching the state representing the best known layout.

The available problem sets are tabulated in the `readme.md` of the package.

6.2.3 Working with Custom Problems

New FLP instances can be loaded at run-time by providing flows and distances or dimensions (depending on which type of problem representation is chosen). The gym-flp import engine accepts text files of the types .json, .txt and .prn.

When passing the value 'custom' to the argument 'instance' upon starting the training, the user will be prompted to provide the desired problem size and the file to read. This happens twice during training (training and evaluation environment) and during testing (final model and best model environment). The second input allows users to provide an evaluation instance that is different from the training instance to test the agent's performance on new information.

In the background, the EnvBuilder processor will attempt to make sure the input file is not ill-structured. The input engine expects the flow information to come first. Ideally, text files are supplied as numeric values only with connections, e.g. $f_{0,0} \ldots f_{n,n}$ separated by a white space, given as one line per machine, separated by a line feed. JSON files, on the other hand, are more complicated to engineer, but easier to parse by the engine. A .json should be structured as per the listing below. Alternatively,

the connections j for each i, i.e. the columns of the matrices, can be given as a list following the key 'i'. The input engine will further attempt to deduce all necessary information required for the `gym-flp` environment. That is, if only area data are provided with the input file, the engine will make facilities square or rectangular with integer side lengths. If no plant dimensions are supplied, the engine will return an area that is twice the size of what is occupied by the specified facilities. Furthermore, the program will raise an exception and terminate if it detects distance information in inputs supplied to a continuous environment, or spatial information fed into a discrete environment, respectively.

For further details on structuring custom input files, readers are referred to the `gym-flp` repository, where downloadable examples are provided.

```
{
 'f':
  {
   '0':
    {'0': f_{0,0},
     ...
    'n': f_{0,n}}
   'n':{'0': f_{n,0},
    ...
    'n': f_{n,n}}
  }
}
```

6.2.4 Example 1: One-Off Training

This example represents an atomic version of algorithm training with a single training run followed by one evaluation run. We train once with 100,000 and once with 1,000,000 steps to compare the effect of increasing the training budget. To train using PPO we invoke the respective scripts from a terminal as follows:

python train.py −−algo ppo −−distance euclidean

and

python train.py −−algo ppo −−distance euclidean −−train_steps 1000000

The results of 100,000 training steps can be found in Fig. 7a. One can observe a steady improvement of MHC for about 80 *evaluation* steps. Afterwards, it is likely that the agent had not yet processed a sufficient amount of training observations, leading it to propose a series of actions that do not yield improvements and eventually run into a termination criterion. With an increased training budget (Fig. 7b), the agent achieves even lower MHC values. In addition to this, we see a plateau starting at around 100 evaluation steps after which the evaluation stops due to not improving for five consecutive steps.

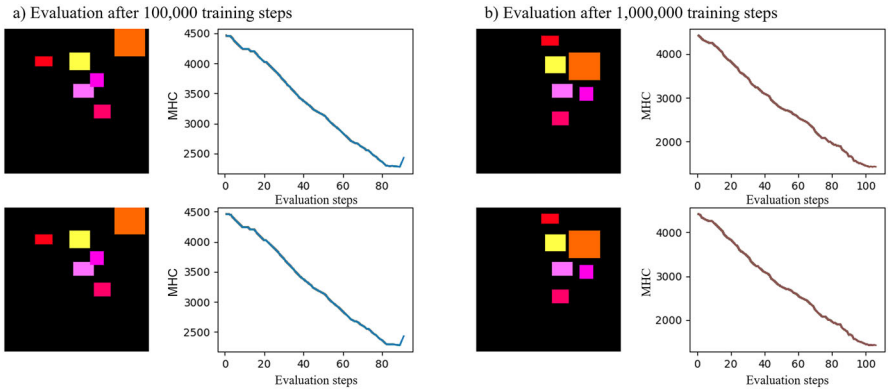

Fig. 7 Results of an evaluation run in ofp-v0 for 100,000 and 1,000,000 training steps in the instance P6

6.2.5 Example 2: Tuning Hyper-parameters with Optuna

The `gym-flp` training script leverages Optuna [68]. Optuna can, for instance, be used to optimise hyper-parameters of agents before submitting them to longer trials.

To do so, any parameter present in the algorithm configuration file can be passed to the training script with two values which will be interpreted as lower and upper bound in that order. The program runs 20 iterations by default, other values can be passed with `tune_runs`. The optimisation procedure is set up to use the relative improvement of MHC (MHC at the start—MHC at the finish) per each trial and attempts to maximise this value. The listing below shows an example for optimising the PPO hyper-parameter `learning_rate`.

```
python train.py --algo ppo --instance P12 --learning\_rate 1e6 1e2
```

It should be noted that RL training, especially with larger problem sets, tends to require many training steps before convergence sets in (e.g. the common instance P6 shows converging behaviour after around 1 million training steps). While it is possible to pass optimisation bounds for all hyper-parameters simultaneously to mimic a full-factorial design of experiment (DoE), it is highly unlikely that this approach will yield any meaningful results in a satisfactory time frame.

6.2.6 Example 3: Meta-study

Optuna is further useful to conduct comprehensive studies to examine the effects of using different algorithms or, as originally intended with `gym-flp`, changes to environment design. The listing below demonstrates an experiment that performs one training run for instance P12 with 1,000,000 training steps for the three algorithms PPO, DQN and A2C (in the given order). The results of the evaluation runs for all three algorithms can be seen in Fig. 8.

```
python train.py --instance P12 --train\_steps 1e6 --algo ppo dqn a2c
```

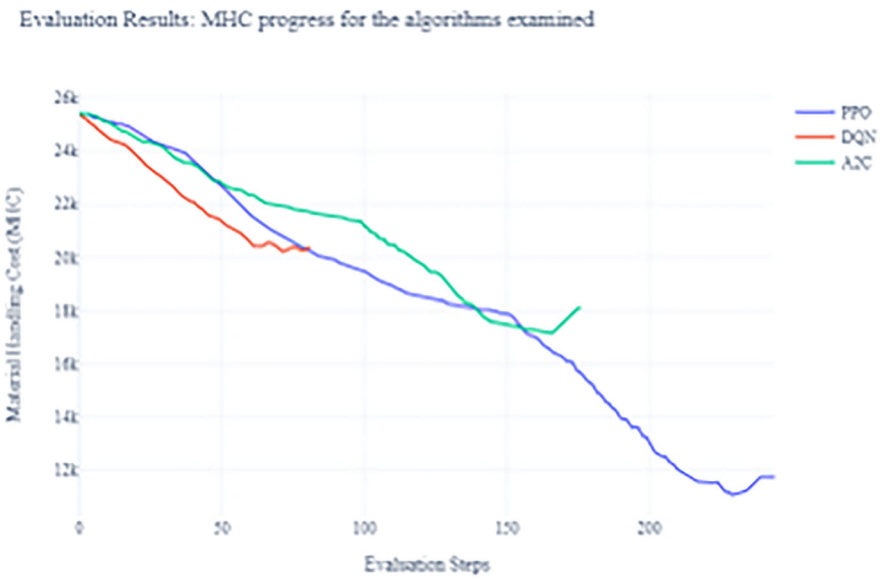

Fig. 8 Evaluation result of training three RL algorithms on the instance P12 for 1M steps each. It can be observed that PPO yields the best results given identical hyper-parameters as it achieves the lowest MHC

Of course, this approach can be used in conjunction with Section 6.2.5. In this case, the Optuna trial will begin by suggesting the input training values and consecutively submitting them to the algorithms passed along as a list. If any of the hyper-parameters passed are not supported by the algorithms, the input values will not have any effect. No warnings are thrown in this case, so special care must be taken when designing experiments.

6.2.7 Example 4: Beyond the Package

Instead of using the experiment workflow that is built-in into `gym-flp`, users can of course write their own scripts for training. This is especially recommended when using RL frameworks other than Stable Baselines 3, such as Ray or a custom implementation. To achieve this, the environment side of `gym-flp` can be used stand-alone. A generic script as a starting point is given below in Algorithm 1.

In the context of operations research, it can be useful to replace steps 9 and 20 by other algorithmic approaches from the field. Taking the OFP as an example, the internal observation can be accessed at any time, too, simply by calling env.internal_state. The internal observation corresponds to the state output in `human` mode and is a vector of size $4 \times n$ that holds the coordinates and dimensions of each facility (see Section 5.3.1). That information could be encoded differently outside `gym-flp`, e.g. as chromosomes to make it useable with genetic algorithms.

Algorithm 1 An algorithm with caption

Initialise gym and gym-flp
$M \leftarrow$ Initialise instance of RL model class
$env \leftarrow$ instance of class GymEnv ▷ Use env=gym.make(...)
$T \leftarrow$ Number of train steps (int)
$s_0 \leftarrow$ env.reset() ▷ obtain initial observation
for $t = 1, T$ **do**
    Sample random action a from action space A
    Pass a to env and receive tuple **return** (o, r, d, i) ▷ env.step(a)
    Pass tuple to RL algorithm (Update replay buffer, value function, network weights, ...) and update M
end for
Save RL model, delete env
Start evaluation run:
Make new env
$eval\_env \leftarrow$ instance of class GymEnv
$o \leftarrow$ eval_env.reset() ▷ obtain initial observation
$M \leftarrow$ Load RL model
$d \leftarrow False$
$mhc \leftarrow$ empty list
while $d \neq True$ **do**
    $a \leftarrow$ predict a for o from M
    Pass a to $eval\_env$ and receive tuple **return** (o, r, d, i)
    Append i to mhc
end while
Plot/Interpret mhc

7 Concluding Remarks

In this paper, we present a Python library intended to assist operations researchers in advancing the usage of Reinforcement Learning techniques for Facility Layout Problems. As RL has gained traction as a methodology for related combinatorial optimisation problems, we firmly believe that this is a direction worth exploring. As an initial stepping stone, we implemented three commonly used FLP problem definitions (QAP, FBS, and STS) plus an open-field layout problem (OFP) including the material handling cost computation. The package follows the conventions used in the OpenAI Gym framework allowing the FLP instances to serve as future benchmark problems for RL in FLP research.

We consider this package in its current version as a starting point for other keen FLP researchers who can freely modify our environments by accessing their local copy of `gym-flp.py` and make amendments to, e.g., reward signals or even add whole new environments to it. We encourage interested researchers to explore the functionality of the package and to engage in, among others, the following activities:

- Filing issues where reproducible code problems arise
- Adding additional RL algorithms by providing an agent branch
- Writing further unit tests using test branches
- Introducing more FLP components by submitting a feature branch (drop-off points, row-based layouts, other reward mechanisms, etc.)

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability The package data that support the findings of this study are available from GitHub, https://github.com/BTHeinbach/gym-flp.

Declarations

Conflict of Interest The authors declare no competing interests.

References

1. Drira A, Pierreval H, Hajri-Gabouj S (2007) Facility layout problems: a survey. Annu Rev Control 31(2):255–267. https://doi.org/10.1016/j.arcontrol.2007.04.001
2. Hosseini-Nasab H, Fereidouni S, Ghomi Fatemi, Taghi Seyyed Mohammad et al (2018) Classification of facility layout problems: a review study. Int J Adv Manuf Technol 94(1–4):957–977. https://doi.org/10.1007/s00170-017-0895-8
3. Pérez-Gosende P, Mula J, Díaz-Madroñero M (2021) Facility layout planning. An extended literature review. Int J Prod Res 59(12):3777–3816. https://doi.org/10.1080/00207543.2021.1897176
4. Burggräf P, Wagner J, Heinbach B (2021) Bibliometric study on the use of machine learning as resolution technique for facility layout problems. IEEE Access 9:22569–22586. https://doi.org/10.1109/ACCESS.2021.3054563
5. Burggräf P, Wagner J, Koke B (2018) Artificial intelligence in production management: a review of the current state of affairs and research trends in Academia. 2018 International Conference on Information Management and Processing (ICIMP 2018): Jan. 12–14, 2018, London, UK. IEEE, Piscataway, NJ, pp 82–88. https://doi.org/10.1109/ICIMP1.2018.8325846
6. Burggräf P, Wagner J, Koke B et al (2020) Performance assessment methodology for AI-supported decision-making in production management. Procedia CIRP 93:891–896. https://doi.org/10.1016/j.procir.2020.03.047
7. Burggräf P, Wagner J, Koke B et al (2020) Approaches for the prediction of lead times in an engineer to order environment–a systematic review. IEEE Access 8:142434–142445. https://doi.org/10.1109/ACCESS.2020.3010050
8. Burggräf P, Wagner J, Koke B et al (2019) Sensor retrofit for a coffee machine as condition monitoring and predictive maintenance use case. Human Practice. Digital Ecologies. Our Future: 14. Internationale Tagung Wirtschaftsinformatik (WI 2019): Tagungsband. Universitätsbibliothek Siegen, pp 62–66. https://aisel.aisnet.org/wi2019/track01/papers/5/
9. Wuest T, Weimer D, Irgens C et al (2016) Machine learning in manufacturing: advantages, challenges, and applications. Prod Manuf Res 4(1):23–45. https://doi.org/10.1080/21693277.2016.1192517
10. Dogan A, Birant D (2021) Machine learning and data mining in manufacturing. Expert Syst Appl 166:114060. https://doi.org/10.1016/j.eswa.2020.114060. https://www.sciencedirect.com/science/article/abs/pii/S095741742030823X
11. Bahrpeyma F, Reichelt D (2022) A review of the applications of multi-agent reinforcement learning in smart factories. Front Robot AI 9:1027340. https://doi.org/10.3389/frobt.2022.1027340. https://www.frontiersin.org/articles/10.3389/frobt.2022.1027340/full
12. Panzer M, Bender B (2022) Deep reinforcement learning in production systems: a systematic literature review. Int J Prod Res 60(13):4316–4341. https://doi.org/10.1080/00207543.2021.1973138

13. Rolf B, Jackson I, Müller M et al (2023) A review on reinforcement learning algorithms and applications in supply chain management. Int J Prod Res 61(20):7151–7179. https://doi.org/10.1080/00207543.2022.2140221
14. Del Real Torres A, Andreiana DS, Ojeda Roldán Á et al (2022) A review of deep reinforcement learning approaches for smart manufacturing in industry 4.0 and 5.0 framework. Appl Sci 12(23):12377. https://doi.org/10.3390/app122312377. https://www.mdpi.com/2076-3417/12/23/12377
15. Bengio Y, Lodi A, Prouvost A (2021) Machine learning for combinatorial optimization: A methodological tour d'horizon. Eur J Oper Res 290(2):405–421. https://doi.org/10.1016/j.ejor.2020.07.063
16. Zuniga ER, Moris MU, Syberfeldt A et al (2020) A simulation-based optimization methodology for facility layout design in manufacturing. IEEE Access 8:163818–163828. https://doi.org/10.1109/ACCESS.2020.3021753
17. Tompkins J, White JA, Bozer YA (2010) Facilities planning, 4th edn. Wiley, Hoboken, NJ
18. Koopmans TC, Beckmann M (1957) Assignment problems and the location of economic activities. Econometrica 25(1):53. https://doi.org/10.2307/1907742
19. Tong X (1991) SECOT: a sequential construction technique for facility design. University of Pittsburgh, Pittsburgh, PA. https://elibrary.ru/item.asp?id=5805928. Accessed 16 Feb 2024
20. Konak A, Kulturel-Konak S, Norman BA et al (2006) A new mixed integer programming formulation for facility layout design using flexible bays. Oper Res Lett 34(6):660–672. https://doi.org/10.1016/j.orl.2005.09.009
21. Haktanirlar Ulutas B, Kulturel-Konak S (2012) An artificial immune system based algorithm to solve unequal area facility layout problem. Expert Syst Appl 39(5):5384–5395. https://doi.org/10.1016/j.eswa.2011.11.046
22. Tam KYR (1992) A simulated annealing algorithm for allocating space to manufacturing cells. Int J Prod Res 30(1):63–87. https://doi.org/10.1080/00207549208942878
23. Niroomand S, Hadi-Vencheh A, Şahin R et al (2015) Modified migrating birds optimization algorithm for closed loop layout with exact distances in flexible manufacturing systems. Expert Syst Appl 42(19):6586–6597. https://doi.org/10.1016/j.eswa.2015.04.040. https://www.sciencedirect.com/science/article/pii/s0957417415002821?casa_token=iavfj2vewwsaaaaa:qispvydrz7gpionc_mjuodgdxjif3ge1ijutuhrqdqpnc5mrfhj7bxkcpwcqalzeecruopmy
24. Yang T, Su CT, Hsu YR (2000) Systematic layout planning: a study on semiconductor wafer fabrication facilities. Int J Oper Prod Manag 20(11):1359–1371. https://doi.org/10.1108/01443570010348299. https://www.emerald.com/insight/content/doi/10.1108/01443570010348299/full/
25. Anjos MF, Vieira MV (2021) Facility layout: mathematical optimization techniques and engineering applications, 1st edn. EURO Advanced Tutorials on Operational Research, Springer International Publishing and Imprint Springer, Cham. https://doi.org/10.1007/978-3-030-70990-7
26. Ueda K, Fujii N, Hatono I et al (2002) Facility layout planning using self-organization method. CIRP Ann 51(1):399–402. https://doi.org/10.1016/S0007-8506(07)61546-7. https://www.sciencedirect.com/science/article/pii/s0007850607615467
27. Tsuchiya K, Bharitkar S, Takefuji Y (1996) A neural network approach to facility layout problems. Eur J Oper Res 89(3):556–563. https://doi.org/10.1016/0377-2217(95)00051-8. https://www.sciencedirect.com/science/article/pii/0377221795000518
28. García-Hernández L, Pérez-Ortiz M, Araúzo-Azofra A et al (2014) An evolutionary neural system for incorporating expert knowledge into the UA-FLP. Neurocomputing 135:69–78. https://doi.org/10.1016/j.neucom.2013.01.068. https://www.sciencedirect.com/science/article/pii/s0925231213011430
29. Weitzel T, Glock CH (2018) Energy management for stationary electric energy storage systems: a systematic literature review. Eur J Oper Res 264(2):582–606. https://doi.org/10.1016/j.ejor.2017.06.052
30. Shi D, Fan W, Xiao Y et al (2020) Intelligent scheduling of discrete automated production line via deep reinforcement learning. Int J Prod Res 58(11):3362–3380. https://doi.org/10.1080/00207543.2020.1717008
31. Kuhnle A, Röhrig N, Lanza G (2019) Autonomous order dispatching in the semiconductor industry using reinforcement learning. Procedia CIRP 79:391–396. https://doi.org/10.1016/j.procir.2019.02.101
32. Malus A, Kozjek D, Vrabič R (2020) Real-time order dispatching for a fleet of autonomous mobile robots using multi-agent reinforcement learning. CIRP Ann 69(1):397–400. https://doi.org/10.1016/j.cirp.2020.04.001

33. Khalil E, Dai H, Zhang Y et al (2017) Learning combinatorial optimization algorithms over graphs. In: Guyon I, Von Luxburg U, Bengio S et al (eds) Advances in Neural Information Processing Systems, vol 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/d9896106ca98d3d05b8cbdf4fd8b13a1-Paper.pdf
34. Unger H, Börner F (2021) Reinforcement learning for layout planning – modelling the layout problem as MDP. In: Dolgui A, Bernard A, Lemoine D et al (eds) Advances in production management systems, IFIP Advances in Information and Communication Technology, vol 632. Springer, Cham, pp 471–479. https://doi.org/10.1007/978-3-030-85906-0_52
35. Klar M, Glatt M, Aurich JC (2021) An implementation of a reinforcement learning based algorithm for factory layout planning. Manuf Lett 30:1–4. https://doi.org/10.1016/j.mfglet.2021.08.003
36. Klar M, Hussong M, Ruediger-Flore P et al (2022) Scalability investigation of double deep q learning for factory layout planning. Procedia CIRP 107:161–166. https://doi.org/10.1016/j.procir.2022.04.027
37. Klar M, Glatt M, Aurich JC (2023) Performance comparison of reinforcement learning and metaheuristics for factory layout planning. CIRP J Manuf Sci Technol 45:10–25. https://doi.org/10.1016/j.cirpj.2023.05.008. https://www.sciencedirect.com/science/article/pii/s1755581723000718
38. Ikeda H, Nakagawa H, Tsuchiya T (2022) Towards automatic facility layout design using reinforcement learning. Communication Papers of the 17th Conference on Computer Science and Intelligence Systems, vol 32. PTI, pp 11–20. https://doi.org/10.15439/2022f25
39. Unger H, Börner F, Fischer D (2024) Reinforcement learning for layout planning – automated pathway generation for arbitrary factory layouts. In: Silva FJG, Ferreira LP, Sá JC, et al (eds) Flexible Automation and Intelligent Manufacturing: Establishing Bridges for More Sustainable Manufacturing Systems. Springer Nature Switzerland and Imprint Springer, Cham, Lecture Notes in Mechanical Engineering, pp 1031–1039. https://doi.org/10.1007/978-3-031-38165-2_118. https://link.springer.com/chapter/10.1007/978-3-031-38165-2_118
40. Heinbach B, Burggräf P, Wagner J (2023) Deep reinforcement learning for layout planning - an MDP-based approach for the facility layout problem. Manuf Lett 38:40–43. https://doi.org/10.1016/j.mfglet.2023.09.007
41. Mnih V, Kavukcuoglu K, Silver D et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533. https://doi.org/10.1038/nature14236. https://www.nature.com/articles/nature14236?wm=book_wap_0005
42. Silver D, Huang A, Maddison CJ et al (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484–489. https://doi.org/10.1038/nature16961. https://www.nature.com/articles/nature16961?mrk_cmpg_source=sm_tw_pp
43. Sutton RS, Barto AG (2018) Reinforcement learning: an introduction, 2nd edn. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, Massachusetts
44. Dong H, Ding Z, Zhang S (2020) Deep reinforcement learning: fundamentals, research and applications, 1st edn. Springer eBook Collection, Springer Singapore and Imprint Springer, Singapore. https://doi.org/10.1007/978-981-15-4095-0
45. Mnih V, Kavukcuoglu K, Silver D et al (2013) Playing Atari with deep reinforcement learning. Preprint at http://arxiv.org/abs/1312.5602
46. Schmigalla H (1970) Methoden zur optimalen Maschinenanordnung. VEB Verlag Technik
47. Patel D, Hazan H, Saunders DJ et al (2019) Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to Atari breakout game. Neural Networks: the Official Journal of the International Neural Network Society 120:108–115. https://doi.org/10.1016/j.neunet.2019.08.009. https://www.sciencedirect.com/science/article/pii/s0893608019302266
48. Li C, Zheng P, Yin Y et al (2023) Deep reinforcement learning in smart manufacturing: a review and prospects. CIRP J Manuf Sci Technol 40:75–101. https://doi.org/10.1016/j.cirpj.2022.11.003. https://www.sciencedirect.com/science/article/pii/s1755581722001717
49. Brockman G, Cheung V, Pettersson L et al (2016) OpenAI gym. Preprint at http://arxiv.org/abs/1606.01540
50. Hein D, Depeweg S, Tokic M et al (2018) A benchmark environment motivated by industrial control problems. 2017 SSCI proceedings: 2017 IEEE SSCI, Honolulu, Hawaii, UA. IEEE, Piscataway, NJ. https://doi.org/10.1109/ssci.2017.8280935
51. Serra T, O'Neil RJ (2020) MIPLIBing: seamless benchmarking of mathematical optimization problems and metadata extensions. SN Operations Research Forum 1(3):14. https://doi.org/10.1007/s43069-020-00024-1

52. Li F, Du Y (2018) From AlphaGo to power system AI: what engineers can learn from solving the most complex board game. IEEE Power Energ Mag 16(2):76–84. https://doi.org/10.1109/mpe.2017.2779554
53. Vázquez-Canteli JR, Kämpf J, Henze G et al (2019) Citylearn v1.0. In: Zhang M (ed) BuildSys '19: Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation: November 13-14, 2019, New York, NY, USA. The Association for Computing Machinery, New York, New York, pp 356–357. https://doi.org/10.1145/3360322.3360998
54. Spangher L, Gokul A, Palakapilly J et al (2020) Officelearn: an OpenAI Gym environment for reinforcement learning on occupant-level building's energy demand response. https://www.climatechange.ai/papers/neurips2020/56/paper.pdf. Accessed 16 Feb 2024
55. Waschneck B, Reichstaller A, Belzner L et al (2018) Optimization of global production scheduling with deep reinforcement learning. Procedia CIRP 72:1264–1269. https://doi.org/10.1016/j.procir.2018.03.212
56. Zamora I, Lopez NG, Vilches VM et al (2016) Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and gazebo. Preprint at https://arxiv.org/pdf/1608.05742.pdf
57. Gawłowicz P, Zubow A (2018) ns3-gym: extending OpenAI Gym for networking research. Preprint at http://arxiv.org/pdf/1810.03943v2
58. Hubbs CD, Perez HD, Sarwar O et al (2020) OR-Gym: a reinforcement learning library for operations research problems. https://doi.org/10.48550/arXiv.2008.06319. Accessed 16 Feb 2024
59. Raffin A, Hill A, Ernestus M et al (2019) Stable baselines3. https://github.com/DLR-RM/stable-baselines3. Accessed 16 Feb 2024
60. Philipp Moritz, Robert Nishihara, Stephanie Wang et al (2018) Ray: a distributed framework for emerging AI applications. 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), pp 561–577. https://www.usenix.org/conference/osdi18/presentation/moritz
61. Schulman J, Wolski F, Dhariwal P et al (2017) Proximal policy optimization algorithms. Preprint at http://arxiv.org/abs/1707.06347
62. Mnih V, Badia AP, Mirza M et al (2016) Asynchronous methods for deep reinforcement learning. In: Balcan MF, Weinberger KQ (eds) Proceedings of The 33rd International Conference on Machine Learning, Proceedings of Machine Learning Research, vol 48. PMLR, pp 1928–1937. https://arxiv.org/pdf/1602.01783
63. Lillicrap TP, Hunt JJ, Pritzel A et al (2015) Continuous control with deep reinforcement learning. Preprint at http://arxiv.org/abs/1509.02971
64. Fujimoto S, van Hoof H, Meger D (2018) Addressing function approximation error in actor-critic methods. In: Dy J, Krause A (eds) Proceedings of the 35th International Conference on Machine Learning, Proceedings of Machine Learning Research, vol 80. PMLR, pp 1587–1596. https://proceedings.mlr.press/v80/fujimoto18a.html
65. Haarnoja T, Zhou A, Abbeel P et al (2018) Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Dy J, Krause A (eds) Proceedings of the 35th International Conference on Machine Learning, Proceedings of Machine Learning Research, vol 80. PMLR, pp 1861–1870. https://proceedings.mlr.press/v80/haarnoja18b.html
66. Burkard RE, Karisch SE, Rendl F (1997) QAPLIB - a quadratic assignment problem library. J Global Optim 10(4):391–403. https://doi.org/10.1023/A:1008293323270
67. La Scalia G, Micale R, Enea M (2019) Facility layout problem: bibliometric and benchmarking analysis. Int J Ind Eng Comput 10(4):453–472. https://doi.org/10.5267/j.ijiec.2019.5.001
68. Akiba T, Sano S, Yanase T et al (2019) Optuna. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, New York, NY, USA, p 2623. https://doi.org/10.1145/3292500.3330701