ASME Journal of Mechanical Design
Online journal at:
https://asmedigitalcollection.asme.org/mechanicaldesign

Lorenzo De Santanna
Department of Mechanical Engineering,
Politecnico di Milano,
Milan 20156, Italy
e-mail: lorenzo.desantanna@polimi.it

Giacomo Guidotti
Department of Mechanical Engineering,
Politecnico di Milano,
Milan 20156, Italy
e-mail: giacomo.guidotti@polimi.it

Gianpiero Mastinu
Department of Mechanical Engineering,
Politecnico di Milano,
Milan 20156, Italy
e-mail: gianpiero.mastinu@polimi.it

Massimiliano Gobbi[1]
Department of Mechanical Engineering,
Politecnico di Milano,
Milan 20156, Italy
e-mail: massimiliano.gobbi@polimi.it

Multi-Objective Optimal Design Based on Reinforcement Learning

This article describes and applies a new optimization method based on multi-objective programming and reinforcement learning. The new method, called MORL-DB (multi-objective reinforcement learning dominace based), introduces the concept of Pareto dominance into the reinforcement learning framework. MORL-DB employs deep deterministic policy gradient (DDPG) with a reward function based on Pareto optimality. At first, the MORL-DB method is tested by solving the Viennet's benchmark problem, then it is applied to the Osyczka and Kundu benchmark problem. Finally, it is used to compute the Pareto front for the vertical dynamics of the quarter vehicle model in terms of two design variables and three objective functions. The results of these three case studies are then compared with the ones obtained using the parameter space investigation method and a nondominated sorting genetic algorithm. The comparison highlights the ability of MORL-DB to generate a high number of optimal solutions with a low number of objective function evaluations.
[DOI: 10.1115/1.4069046]

Keywords: reinforcement learning, DDPG, multi-objective optimization, design optimization, quarter vehicle model

1 Introduction

Nowadays, artificial intelligence is transforming many industrial sectors [1]. In particular, reinforcement learning (RL) techniques have demonstrated remarkable results in the control of nonlinear dynamic systems. For instance, mobile robots utilize neural network controllers that exploit the aforementioned techniques [2].

One of the few areas where RL techniques are still rarely used is the field of design optimization, for which the low number of published works is far from being a complete and autonomous branch of the technical literature. Sharpe et al. [3] use different case studies to compare several RL-based algorithms capable of mapping optimal or feasible regions of the design space. Ororbia and Warn [4] represent structural design problems as Markov decision processes and solve them with RL. In several case studies, the authors demonstrated that RL can find better solutions with fewer objective function evaluations with respect to a genetic algorithm. Similar results are obtained by Lee et al. [5], who apply RL methods in the context of microfluidic device design for flow sculpting. Wang et al. [6] apply a multi-objective deep reinforcement learning method in the context of the optimization of a wind turbine blade. They managed to obtain a better approximation of the Pareto front in terms of hypervolume with respect to nonsorted genetic algorithm (NSGA) methods. Van Moffaert and Nowé [7] tested different RL multi-objective algorithms in two different benchmark problems. In the considered case studies, single-policy algorithms failed to approximate the entire Pareto front, while multipolicy approaches successfully achieved the goal at the cost of a large number of objective function evaluations. Raina et al. [8] employ an autoencoder-based visual imitation learning process in the context of a truss design problem, in which the algorithm managed to outperform the human designers used to train the networks. In another study, Raina et al. [9] solve the truss design problem with a hierarchical architecture that, initially, evaluates the area of the structure that needs to be modified and then applies an action according to a probability distribution. He et al. [10] applied a multi-agent reinforcement learning algorithm to the optimization of a chemical process used in the textile industry and managed to obtain results closer to their target with respect to the ones obtained with traditional approaches.

Despite the great potential demonstrated by RL in this context, the most popular methods for solving multi-objective design optimizations still remain gradient-based or meta-heuristic algorithms [11]. In particular, genetic algorithm [12] and the particle swarm optimization [13] are extensively used.

The aim of this article is to apply RL in a multi-objective design optimization framework. The concept is to use a method that initially explores the design domain in a random fashion and gradually learns where the optimal design solutions are located. The performance of this method has been tested both on Viennet's benchmark problem [14] and on Osyczka and Kundu's benchmark problem [14], and on the quarter vehicle model (QVM), for which the performance is described by three conflicting objective functions inside a bidimensional design domain. The results are then compared to the ones of NSGA-II [15] and parameter space investigation (PSI) [11] to highlight the advantages and drawbacks of the new procedure with respect to the state-of-the-art methods.

[1]Corresponding author.
Contributed by the Design Automation Committee of ASME for publication in the Journal of Mechanical Design. Manuscript received October 28, 2024; final manuscript received May 6, 2025; published online August 4, 2025. Assoc. Editor: Ikjin Lee.

Section 2 describes the deep deterministic policy gradient (DDPG) algorithm [6] and how it is adapted to multi-objective design optimization. In Sec. 3, PSI and NSGA-II methods are briefly introduced. Section 4 presents the three case studies. The results are analyzed and compared in Sec. 5.

2 Reinforcement Learning-Based Multi-Objective Optimization

In this section, the new so-called MORL-DB (multi-objective reinforcement learning dominace based) method is described. Almost all of the algorithm descriptions currently available in the technical literature use the time variable to organize the evolution of steps [16]. In our description, the temporal dimension is avoided, as the MORL-DB method is applied in the context of design optimization.

2.1 Reinforcement Learning. Figure 1 shows the standard RL setup, which is composed of an environment E and an agent A interacting with each other. E is described by its state s_n. The agent receives at each step n (also known as episode n) of the algorithm an observation o_n of E, which could be a partial description of E. In our case, it is assumed that the agent fully observes E and so state and observation are coincident, $o_n = s_n$. For this reason, only states will be considered. The agent interacts with E by executing an action a_n, which causes the state s_n of E to change to s_{n+1} according to the governing laws of E. Moreover, A is granted a scalar reward $r_n(s_n, a_n, s_{n+1})$ dependent on the state transition from s_n to s_{n+1} and the performed action a_n at each algorithm step [2].

2.2 Deep Deterministic Policy Gradient. The DDPG is described as "a model-free, off-policy actor-critic algorithm using deep function approximators that can learn policies in high dimensional, continuous action spaces" [16]. This definition means that DDPG does not require any prior knowledge of the environment (model-free), aims at optimizing policy different from the one used to choose the action (off-policy), and is based on the symbiotic cooperation of two artificial neural networks that are generally referenced as actor and critic, respectively.

The actor–critic structure is obtained by combining a deep Q-learning algorithm [17], which can only deal with discrete action spaces, and a deterministic policy gradient algorithm [18], which is quite effective in general, but unsuited for complex control tasks.

The actor and the critic together form the agent. The actor is the artificial neural network in charge of deciding which action to take given the environment state s_n. The decision process of the actor is called policy π, and it is modeled by a function that goes from the state space to the action space, such that $a_n = \pi(s_n)$.

A policy, to be improved, must be measured in its outcomes. Starting from a state s_n and from an action a_n (which leads to the next state s_{n+1} and a reward $r(s_n, a_n, s_{n+1})$), the sum of discounted future rewards obtained following a policy π is called expected return $J(s_n)$:

$$J(s_n) = \mathbb{E}_{r_i, s_{i+1} \sim E, a_i \sim \pi}\left[\sum_{i=n}^{N} \gamma^{(i-n)} r(s_i, a_i, s_{i+1})\right] \tag{1}$$

where the notation of $\mathbb{E}_{r_i, s_{i+1} \sim E, a_i \sim \pi}$, also employed in the next formulae, indicates that the state s_{i+1} and the reward r_i are of the assigned environment E and the action a_i comes from the policy π, N is the number of future steps considered in the sum, and $\gamma \in (0, 1)$ is a discount factor that can be tuned to vary the weight of faraway states with respect to the closest. In the DDPG algorithm, the goal is to obtain an optimal deterministic policy $\mu(s_n)$ maximizing the expected return $J(s_n)$ from a state s_n.

The critic is an artificial neural network that approximates the action-value function $Q^{\pi}(s_n, a_n)$, also known as Q-function. The Q-function evaluates an action-state pair (s_n, a_n) by computing the expected return $J(s_n)$ after taking the action a_n at the state s_n and afterwards obeying a policy π. In case of the deterministic policy $\mu(s_n)$ as in DDPG, the Q-function Q^{μ} is given by

$$Q^{\mu}(s_n, a_n) = \mathbb{E}_{r_n, s_{n+1} \sim E}[r(s_n, a_n) + \gamma Q^{\mu}(s_{n+1}, \mu(s_{n+1}))] \tag{2}$$

The parameters of the critic artificial neural network Θ^Q are trained to approximate $Q^{\mu}(s_n, a_n)$. To quantify how close it is to the optimal function, the mean-squared Bellman error function $L(\Theta^Q)$ of the critic is employed. It is given by

$$L(\Theta^Q) = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\Theta^Q))^2 \tag{3}$$

where $Q(s_i, a_i|\Theta^Q)$ is the critic neural network and y_i is computed with the following equation:

$$y_i = r_i + \gamma Q'^{\mu'}(s_{i+1}, \mu'(s_{i+1}|\Theta^{\mu'})|\Theta^{Q'}) \tag{4}$$

where $\mu'(s_n|\Theta^{\mu'})$ and $Q'(s_n, a_n|\Theta^{Q'})$ are, respectively, the actor and critic target neural networks. These target neural networks differ from $Q(s_n, a_n|\Theta^Q)$ and $\mu(s_n|\Theta^{\mu})$, and they are used to stabilize the training process. The relationship between the neural networks of the actor and the critic and their respective target neural networks is shown later.

The actor network is trained to approximate the optimal policy $\mu(s_n) = argmax_{a_n} Q^{\mu}(s_n, a_n)$. Therefore, the parameters Θ^{μ} of the actor network are updated at each learning step of the training process by maximizing the deterministic policy gradient, which is computed in a minibatch of previous state transitions (s_n, a_n, r_n, s_{n+1})

$$\nabla_{\Theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_a Q(s_i, a|\Theta^Q)|_{a=\mu(s_i)} \nabla_{\Theta^{\mu}} \mu(s_i|\Theta^{\mu}) \tag{5}$$

The minibatch is extracted from an experience buffer, whose length is limited. If the number of episodes exceeds the experience buffer length, the first training episodes are deleted from the buffer while the last ones are kept.

A "soft update" according to Eq. (6) is performed on the target networks at each learning step.

$$\Theta' \leftarrow \tau\Theta + (1 - \tau)\Theta' \tag{6}$$

In Eq. (6), Θ are the artificial neural network parameters of the actor or the critic after the training and Θ' are the ones of the target neural networks. This "soft update" helps at stabilizing the training process of the artificial neural networks since the target values change slowly if $\tau \ll 1$.

The chances of finding the optimal policy during the training process are increased by better exploring the state space of the environment and the action space of the agent. To do so, noise sampled from a noise process $\mathcal{N}$ is added to the actor policy during training.

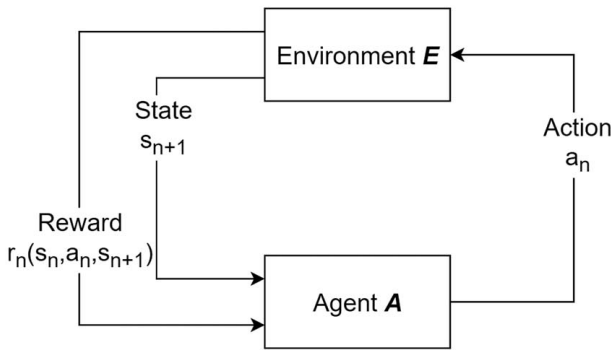

Fig. 1 Agent–environment interaction loop

The resulting policy is called exploration policy $\tilde{\mu}(s_n)$ and is given by

$$\tilde{\mu}(s_n) = \mu(s_n|\Theta^\mu) + \mathcal{N} \tag{7}$$

During training, the noise is progressively reduced so that at the end of training, each action is more deterministic. In this article, a Gaussian model is used for the mathematical description of noise. At each training episode, the Gaussian noise $\mathcal{N}$ is sampled as

$$\mathcal{N} = m + r\sigma_n \tag{8}$$

where m is the mean of the Gaussian process (in this article $m = 0$), r is a vector sampled from the standard normal distribution with the same dimension as the action, and σ_n is the standard deviation (SD) of the Gaussian process. The standard deviation σ_n is updated after each episode with the following equation:

$$\sigma_{n+1} = \sigma_n(1 - D) \tag{9}$$

where σ_{n+1} is the standard deviation of the noise for the next episode, σ_n is the standard deviation of the noise for the current episode, and D is the decay rate, which is a constant parameter to be set before training.

Another feature of the training process is that it does not start immediately after the first step, but there has to be a number of warm-up steps during which the policy is not used and the actions are randomly generated in order to explore as much as possible the environment.

All the parameters that define any configurable part of the learning process of the DDPG agents are called hyperparameters [16].

The steps of the DDPG algorithm are summarized in Fig. 2.

2.3 Application of Reinforcement Learning to Multi-objective Design Optimization. To adapt reinforcement learning techniques to multi-objective design optimization, several specific modifications are included. First, each state coincides with a design solution. Thus, a state change caused by an action is a design solution change.

Second, each training episode involves the succession of the steps that are depicted in Fig. 3.

At the beginning of each episode, a random sample is generated in the neighborhood of the center of the design domain. This allows the same design solution to be reached at each episode with the same variation of design variables, i.e., the same action. Moreover, this ensures that the maximum value of the action does not depend on the initial state. If initial design solutions were very close to the boundaries of the design domain, the action space would need to be constrained to prevent the design solution from exceeding the domain limits.

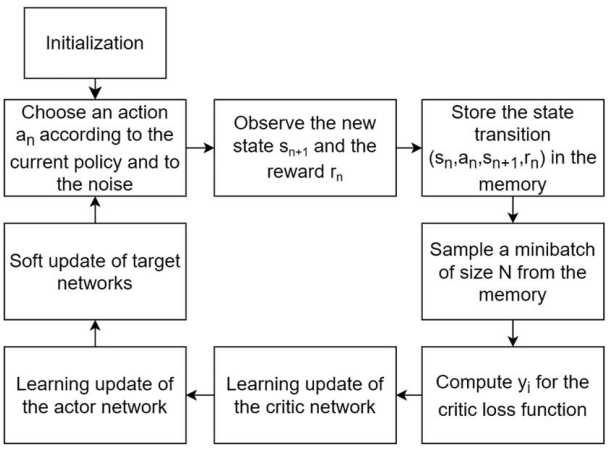

Fig. 2 Flowchart of the DDPG algorithm. At the beginning of each episode, the actor chooses an action according to the current policy from the initial state s_n. This action is then modified by the noise. In the next step, the obtained action leads to a new state s_{n+1} and to a reward r_n. Then, this state transition is stored in the memory. In the end, a minibatch of previous experiences is extracted from the memory and it is used to train the actor, the critic, and their respective target networks.

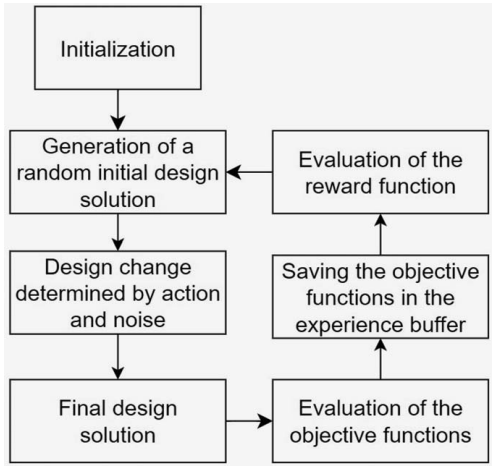

Fig. 3 Training algorithm

Once the initial design is generated, the actor reads this design solution and uses it to take action. Then, the action is summed to noise and to the random design solution to determine the final design solution. Once the final design solution has been calculated, the objective functions are computed and memorized together with the design solution. Then, the reward function is calculated by evaluating the Pareto optimality of the current design solution with respect to the ones previously obtained. After this step, the model is updated and the process is repeated starting from the random initial design generation. The training is stopped after a given number of episodes.

At the end of the training, the Pareto front in both the objective functions and design variable spaces is extracted from the values of the samples stored in memory. Therefore, the training process also coincides with the estimation of the Pareto front.

The problem of applying RL to multi-objective optimization lies in the need for a scalar reward, which has to represent the quality of a vector of objective functions. This problem can be solved by means of multipolicy or single-policy algorithms [7]. In multipolicy algorithms, multiple agents, which have different rewards, are trained simultaneously [10]. This greatly increases the complexity of the method as more agents have to be trained. On the other hand, single-policy methods are simpler but require the definition of a scalarization function to represent the best trade-offs between the objective functions. In the literature, this problem is generally solved by means of a weighted average of the objective functions [7]. Weighted sum requires the determination of optimal solutions for different weight combinations and thus to repeat the training of the agent. The scalarization method proposed in this article focuses instead on the concept of Pareto optimality alone. In this way, a high reward is assigned to all members of the Pareto front without the need to change the reward function during optimization. In particular, the reward function is equal to 1 if the design solution is not dominated by any design solution already obtained during training; otherwise, it is equal to 0. The nondominated Pareto solutions d^* are the design solutions belonging to the design domain D, such that it does not exist a design solution $\hat{d}$ such that [19]

$$\begin{cases} f_j(\hat{d}) \le f_j(d^*) & \forall\, j \\ \exists\ l\colon\ f_l(\hat{d}) < f_l(d^*) \end{cases} \tag{10}$$

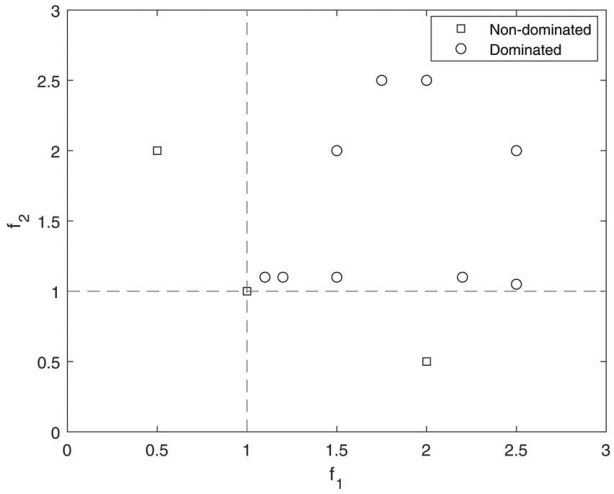

Fig. 4 Example of dominated (circles) and non-dominated solutions (squares)

To make this concept clearer, an example is shown in Fig. 4. The goal is to minimize f_1 and f_2. In Fig. 4, both squares and circles represent possible solutions in the objective function space. Circles have worse performance with respect to squares. Circles are called dominated solutions and represent poor design solutions. Each square represents a dominant solution, which is optimal in the sense than any other square reduces one objective function at the cost of increasing the other. The Pareto front is made up of dominant solutions that represent, for the given set of solutions, the best compromises achievable among the objective functions.

If the multi-objective optimization problem contains constraint functions, the reward of the design solutions that do not respect all the constraint functions is computed as

$$\text{Reward} = \frac{-1}{1 + n_{\text{constraints}} - n_{\text{violated}}(d)} \tag{11}$$

where d is a design solution, $n_{\text{constraints}}$ is the number of constraint functions, and $n_{\text{violated}}(d)$ is the number of constraint functions violated by the design solution d.

The optimization method discussed so far is depicted in the flowchart of Fig. 5.

3 Comparative Analysis

MORL-DB is compared to PSI and NSGA-II methods to evaluate its performance. In this section, these traditional methods are presented. In addition, the performance metrics used in the comparison are introduced.

3.1 Parameter Space Investigation. The first method which is compared to MORL-DB is PSI. It is based on the generation of uniformly distributed design solutions, which are then evaluated to create an approximation of the Pareto front in the objective function space [20].

PSI explores the entire design domain without focusing on a specific region and is not subject to premature convergence. For these reasons, it is a very robust approach. Furthermore, it is easy to setup, since, once the design domain boundaries and the number of designs to test are fixed, the algorithm has no dependence on any other hyperparameter. These advantages are the key reasons behind the selection of this method.

3.2 Genetic Algorithm. A second method used to compute the Pareto front is the NSGA-II [15,21].

It is an evolutionary algorithm inspired by natural selection processes, and it is based on the following steps:

(1) Generation of random individuals
(2) Evaluation of the fitness of the population
(3) Representation of each individual with a binary string
(4) A probability of reproduction is assigned to each individual according to the fitness
(5) A couple of parents generate an offspring by combining their strings
(6) Bits of the offspring are changed with a given probability of mutation
(7) The offspring is decoded in order to find the corresponding design solution
(8) Evaluation of the fitness of the population
(9) Discard the less fit individuals

The steps between 3 and 9 are repeated until the entire population is Pareto-optimal or until one of the stopping criteria is met.

The settings of the NSGA-II algorithm have been chosen after a sensitivity analysis and are listed in Table 1.

The genetic algorithm was selected for its demonstrated ability to solve multi-objective optimization problems. Its key qualities include computational speed and the ability to generate a set of Pareto-optimal solutions.

3.3 Performance Metrics. In order to compare the three methods in computing the Pareto front approximations, several metrics are introduced.

First of all, computational time is used since an ideal optimization method must take as little time as possible to solve an optimization problem. In addition, the number of objective function evaluations and the number of Pareto points generated are also taken into account, as the aim is to obtain a very good approximation of the Pareto front with the lowest number of objective function evaluations. Since in MORL-DB the optimization process coincides with the training, it has a stochastic nature. Therefore, to evaluate the impact of stochasticity on the results, each case study is analyzed by running MORL-DB five times. After completing all five runs, the distribution of optimal designs in the objective function space can be compared across the five runs, and the mean and standard deviation of the optimal design solutions obtained over the five runs can be computed.

In MORL-DB, the number of objective function evaluations corresponds to the number of training episodes, since the objective functions are evaluated once per training episode. The number of episodes is a user-defined hyperparameter. For PSI, the number of evaluations is given by the number of simulated designs, also set by the user. Finally, as far as the genetic algorithm is concerned, the total number of evaluations is equal to the product of the number of generations and the number of individuals per generation. To ensure a fair comparison between the performances of the methods, the population size in the genetic algorithm and the number of designs simulated by the PSI were chosen to obtain similar results to MORL-DB in terms of both the number of nondominated solutions and the quality of the solutions.

Finally, an optimization method must generate a uniform point distribution. To quantify this characteristic, the Spacing metric (SP), which is the standard deviation of the distance between the points of the Pareto front, is used. It can be computed as

$$d_i = \min_{i,j \neq i} \left(\sum_{k=1}^{K} |f_i^k - f_j^k| \right) \tag{12}$$

$$\hat{d} = \frac{1}{n} \sum_{i=1}^{n} d_i \tag{13}$$

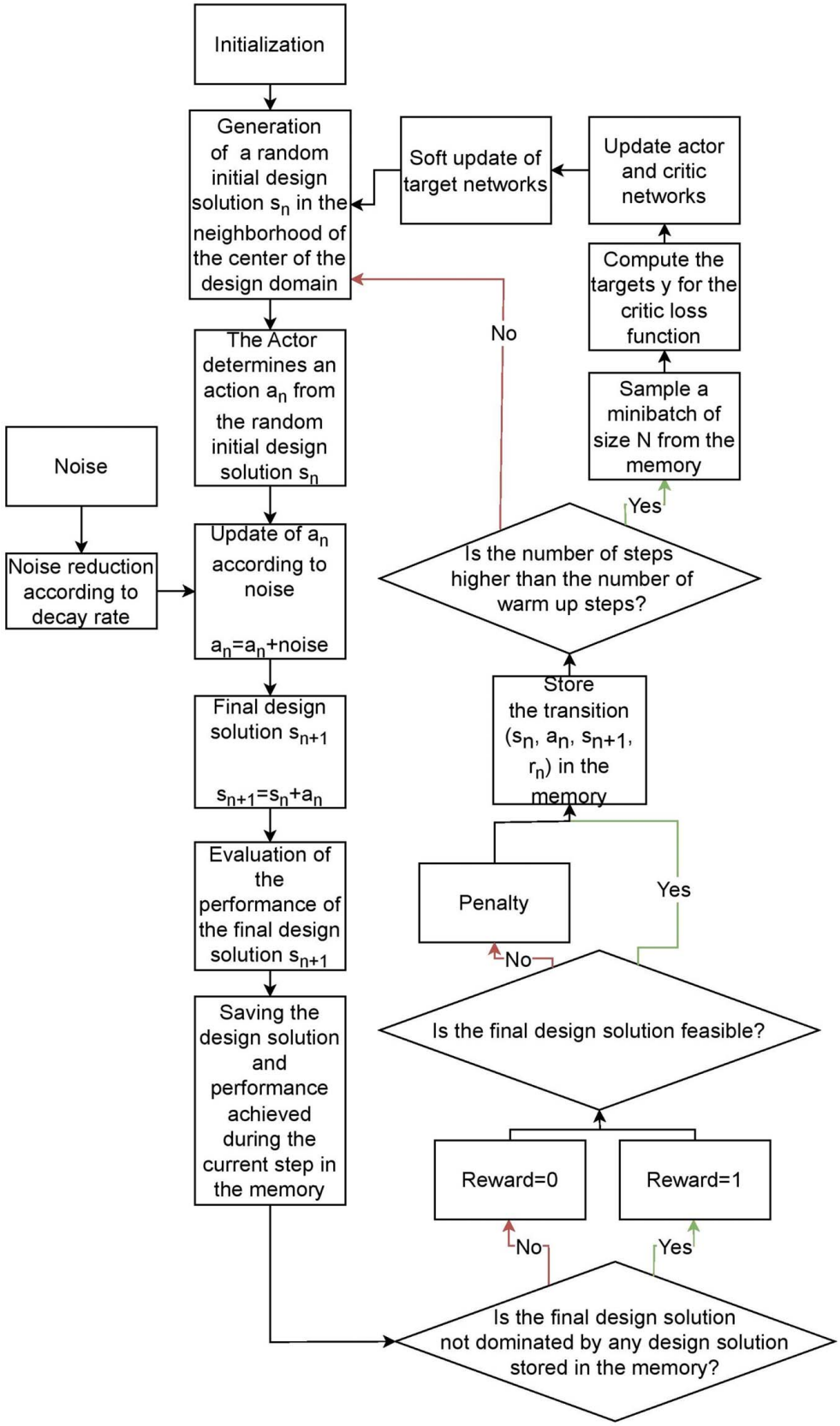

Fig. 5 Flowchart of the whole optimization process. At the beginning of each episode, a random initial design solution is generated in the neighborhood of the center of the design domain. Then, the action determined by the actor and the noise is summed up to find the final design. In the following step, the performance of the final design is evaluated and, according to its Pareto optimality and its feasibility, a reward is assigned. After that, depending on whether the number of episodes is higher or lower than the warm-up step, the actor, the critic, and their respective target networks are updated.

Table 1 Selected parameters used in the case studies

Number of bits for each design variable	8
Mutation probability	3.22%
Fitness function	Dominance depth
Stopping condition	The entire population is nondominated

$$\mathrm{SP} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\hat{d} - d_i)^2} \tag{14}$$

where f is an objective function, K is the number of objective functions, and n is the number of Pareto-optimal solutions [13].

4 Case Studies

This section introduces two case studies that are used to compare the methods previously described.

4.1 Viennet's Benchmark Problem. MORL-DB is tested with Viennet's benchmark problem [14], which is given by

$$\min\begin{cases} 0.5(x_1^2 + x_2^2) + sin(x_1^2 + x_2^2) \\ \frac{(3x_1 - 2x_2 + 4)^2}{8} + \frac{(x_1 - x_2 + 1)^2}{27} + 15 \\ \frac{1}{x_1^2 + x_2^2 + 1} - 1.1e^{(-x_1^2 - x_2^2)} \end{cases} \tag{15}$$

$$\text{s.t.} \quad \begin{cases} -30 \le x_1 \le 30 \\ -30 \le x_2 \le 30 \end{cases}$$

It is a tri-objective optimization problem with two design variables, x_1 and x_2.

The problem is very complex since there are many local optimal solutions in the space of the objective functions, and the Pareto front is discontinuous [14]. A minimum number of iterations of 20,000 was imposed for the genetic algorithm to avoid an early stopping on local suboptima.

To exploit the full potential of the MORL-DB method, its hyperparameters [16] must be tuned for each specific case study. After a series of tests, it has been decided to use the hyperparameters collected in Table 2. All hyperparameters not listed in Table 2 remained the same as Matlab Reinforcement Learning Toolbox's default value [22].

In this case study, the effect of the variation of each hyperparameter is not shown. To quantify it, a sensitivity analysis has been done with the third case study. Although the three case studies are different, the changes in performance metrics associated with a change in hyperparameters are very similar.

As stated previously, at the end of each step, the initial state of the next step is always extracted from the neighborhood of the center of the design domain. For this reason, the method needs to find the best design solution with a single action. A low discount factor promotes actions that lead to a high reward (i.e., an optimal design solution) immediately after the initial state. It is therefore applied in MORL-DB.

Table 2 List of modified hyperparameters and their values for the Viennet problem

Number of neurons per layer	128
Decay rate of the standard deviation	0.075
Minibatch size	256
τ_{Actor}	0.00025
τ_{Critic}	0.00025
Number of episodes	4000
γ	10^{-10}
Maximum experience buffer length	10^4

Note: The other hyperparameters assume the default values in the MATLAB RL Toolbox.

Table 3 List of modified hyperparameters and their values for the Osyczka and Kundu problem

Number of neurons per layer	128
Decay rate of the standard deviation	0.000075
Minibatch size	512
τ_{Actor}	0.000125
τ_{Critic}	0.000125
Number of episodes	25000
γ	10^{-10}
Maximum experience buffer length	2048

Note: The other hyperparameters assume the default values in the Matlab RL Toolbox.

4.2 Osyczka and Kundu Benchmark Problem. Another test used to evaluate the performance of MORL-DB is the Osyczka and Kundu problem [14]. It can be written as

$$\min\begin{cases} -(25(x_1 - 2)^2 + (x_2 - 2)^2 + (x_3 - 1)^2 + (x_4 - 4)^2 + (x_5 - 1)^2) \\ x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 \end{cases}$$

$$\text{s.t.} \quad \begin{cases} x_1 + x_2 - 2 \ge 0 \\ 6 - x_1 - x_2 \ge 0 \\ 2 + x_1 - x_2 \ge 0 \\ 2 - x_1 + 3x_2 \ge 0 \\ 4 - (x_3 - 3)^2 - x_4 \ge 0 \\ (x_5 - 3)^2 + x_6 - 4 \ge 0 \\ 0 \le x_1, \quad x_2, \quad x_6 \le 10 \\ 1 \le x_3, \quad x_5 \le 5 \\ 0 \le x_4 \le 6 \end{cases} \tag{16}$$

Because of the large number of constraints and design variables, a higher number of episodes is necessary. Given the high number of episodes, a lower noise decay rate and lower learning rates were used to lengthen the exploration period and ensure stable training. The minibatch was also increased and, unlike the previous case, the length of the experience buffer was limited so that the agent only learns from the last episodes. This is necessary because, the number of episodes being quite high, the designs that initially obtained a unitary reward may no longer receive it. For this reason, they would then make the agent focus on suboptimal areas of the design domain making the algorithm inefficient.

The values of the hyperparameters used for this case study are listed in Table 3. The hyperparameters not present in the table are left as the default values of the MATLAB Reinforcement Learning Toolbox [22].

4.3 Quarter Vehicle Model. MORL-DB is also tested on the design problem of a road vehicle suspension [19,23]. The adopted model of suspension is a QVM for passively suspended road vehicles, which is shown in Fig. 6. In this model, the mass m_1 is given by the sum of the mass of the wheel and a part of the mass of the suspension arms, m_2 is roughly a quarter of the car's body mass, k_1 is the tire radial stiffness, r_2 and k_2 are, respectively, the damping and the linear stiffness of the suspension [12,19,23].

In this case study, k_2 and r_2 are the design variables while the other quantities are constants. The lower and upper bounds of the design variables and the values of the constants are collected in Table 4. These values and ranges are arbitrary and have been chosen to represent an average small vehicle.

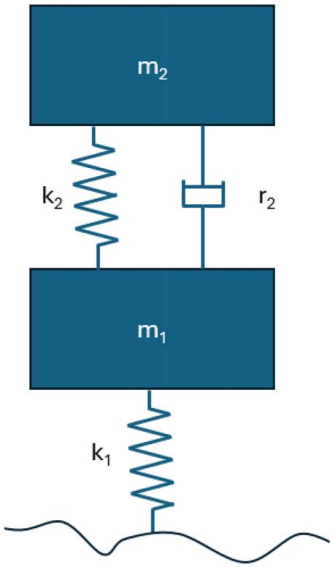

Fig. 6 Simplified suspension model

Table 4 Upper and lower bounds of the design variables and constant values

Quantity	Value
k_1	120,000 N/m
m_1	30 kg
k_2	0–50,000 N/m
r_2	0–5000 N s/m
m_2	230 kg

The three objective functions to be minimized are the road holding σ_{F_z} (standard deviation of the vertical contact patch load), working space $\sigma_{x_2-x_1}$ (the standard deviation of the relative distance between m_1 and m_2), and the discomfort $\sigma_{\ddot{x}_2}$ (standard deviation of the vertical acceleration of the car body m_2).

These objective functions can be computed directly by the following equations:

$$q = \frac{m_1}{m_2} \tag{17}$$

$$K_x = k_2 \frac{(1+q)^2}{k_1 q} \tag{18}$$

$$R_x = r_2 \sqrt{\frac{(1+q)^3}{k_1 m_2 q}} \tag{19}$$

$$f_1^2 = \sqrt{\frac{m_2(1+q)^5}{k_1 q}} \left(\frac{1}{R_x}\right) \tag{20}$$

$$\sigma_{x_2-x_1} = \sqrt{1/2A_b v} f_1 \tag{21}$$

$$f_2^2 = \sqrt{\frac{k_1^3 q^3}{k_1 q}} \left(\frac{K_x}{R_x} + \frac{R_x}{q}\right) \tag{22}$$

$$\sigma_{\ddot{x}_2} = \sqrt{1/2A_b v} f_2 \tag{23}$$

Table 5 List of modified hyperparameters and their values for the quarter vehicle model

Number of neurons per layer	144
Decay rate of the standard deviation	0.00075
Minibatch size	128
τ_{Actor}	0.00025
τ_{Critic}	0.00025
Number of episodes	1000
γ	10^{-10}
Maximum experience buffer length	10^4

Note: The other hyperparameters assume the default values in the MATLAB RL Toolbox.

Table 6 Values of the hyperparameters tested during the sensitivity analysis

Hyperparameter	Low	Reference	High
Number of neurons per layer	80	144	208
Decay rate of the SD	0.000375	0.00075	0.001125
Minibatch size	64	128	192
τ_{Actor}	0.000025	0.00025	0.0025
τ_{Critic}	0.000025	0.00025	0.0025
Number of episodes	500	1000	1500
γ	0.2	10^{-10}	10^{-20}

Note: The other hyperparameters assume the default values in the MATLAB RL Toolbox.

$$f_3^2 = \sqrt{k_1^3 m_2 q^3 (1+q)} \left(\frac{(K_x-1)^2}{R_x} + \frac{1}{q}\left(R_x + \frac{1}{R_x}\right)\right) \tag{24}$$

$$\sigma_{F_z} = \sqrt{1/2A_b v} f_3 \tag{25}$$

where A_b is an index of the road irregularity and v is the vehicle speed. In this case study, the vehicle speed is 30 m/s while A_b is set to $1.4 \cdot 10^{-5}$.

Even in this case, the majority of the hyperparameters of MORL-DB are left as the default value of the Matlab Reinforcement Learning Toolbox [22], while the modified ones and their values are listed in Table 5. In this case, a low decay rate is employed to make the algorithm explore more. Moreover, due to the low complexity of the problem, a low minibatch size and a low number of episodes are used.

In this case study, a sensitivity analysis of the hyperparameters with a one-factor-at-a-time approach [19] is carried out. The values tested for each hyperparameter can be found in Table 6.

Since this method has variable performance due to noise and random initialization, five tests are done for each combination of hyperparameters. The average of the results is used for the sensitivity analysis.

Two metrics used in the sensitivity analysis are the number of Pareto-optimal points and the time required to complete the optimization. Since the performance of the method must be also robust, the standard deviation of the number of Pareto-optimal points is used as an index of robustness.

5 Results

5.1 Viennet. The results are shown and compared to the ideal ones in Figs. 7 and 8.

As can be seen from Figs. 7 and 8, MORL-DB has a good level of accuracy despite the fact that few simulations were carried out. Moreover, results closer to the theoretically correct one can be achieved with more simulations.

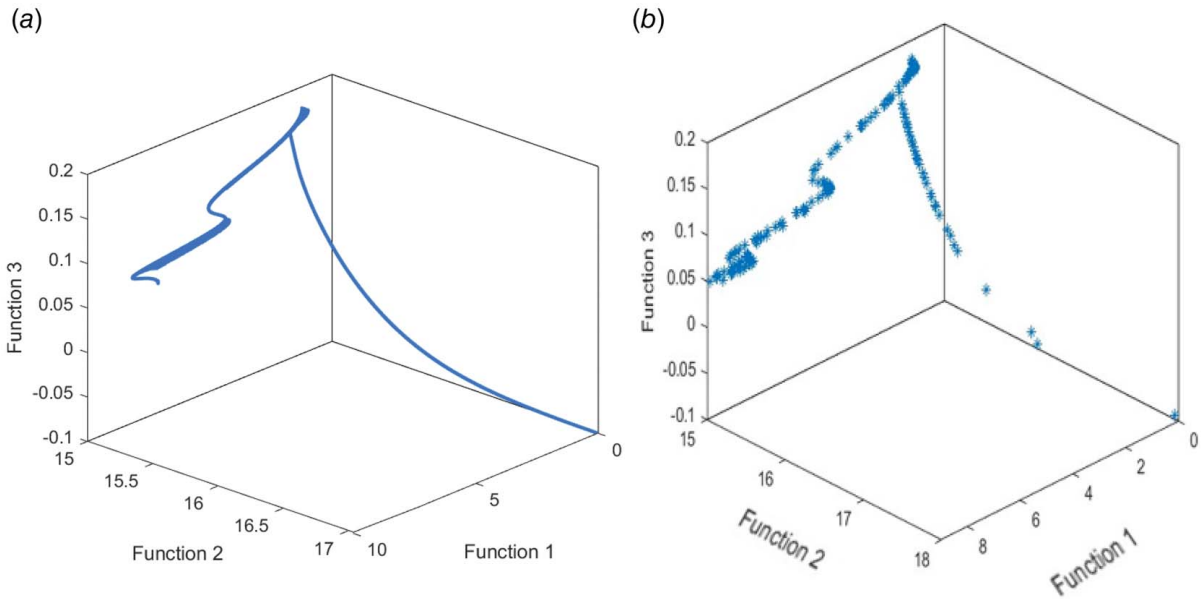

Fig. 7 Theoretically correct Pareto front for Viennet's problem [14] (*a*) compared to the one obtained with the RL-based optimization (*b*)

As shown in Table 8, the main problem with MORL-DB is that, although a very low number of episodes is required to obtain a good approximation of the Pareto front, the time required to train the neural networks makes the optimization process slower with respect to the other methods. However, the situation could be the opposite when optimizing objective functions that require more time to be evaluated. This is, for instance, the case of the computational fluid dynamics (CFD)-based optimization of aerodynamic wing profiles. According to Han et al. [24], an optimization of this kind can take more than 100 h, if solely high-fidelity CFD simulations are employed. In such a context, even slightly reducing the number of simulations to be carried out can lead to a significant reduction in the number of optimization hours.

At the end of all five iterations performed for this case study, comparable results were obtained in terms of the distribution of Pareto-optimal solutions. Moreover, as shown in Table 7, the stochasticity does not influence the number of Pareto-optimal solutions obtained at the end of the optimization.

Another fact worth noting is that the SP values (Eq. (14)) shown in Table 8 are quite close. This means that, even if MORL-DB does not contain any explicit penalty for close Pareto points, MORL-DB is able to create a quite uniform approximation of the Pareto front.

The main limitation of MORL-DB is that these performances are achieved with a good combination of hyperparameters, which may require several tuning iterations to be obtained. It should be considered, however, that the genetic algorithm also required changes to its settings to avoid early stopping and that, in the future, good combinations of the hyperparameters for common problems might be found in the technical literature.

5.2 Osyczka and Kundu. The results of the Osyczka and Kundu problem obtained by means of MORL-DB, NSGA-II, and PSI are shown and compared to the ideal ones in Fig. 9.

As can be seen, fewer points were obtained than in the previous case. Moreover, while PSI manages to obtain a fairly uniform distribution of dots, NSGA-II and MORL-DB generate several zones of high solution concentration. None of the methods perfectly approximates the Pareto front everywhere, but comes closest in certain areas. As can be seen, for example, in the central zone, the solutions obtained by MORL-DB dominate those obtained by

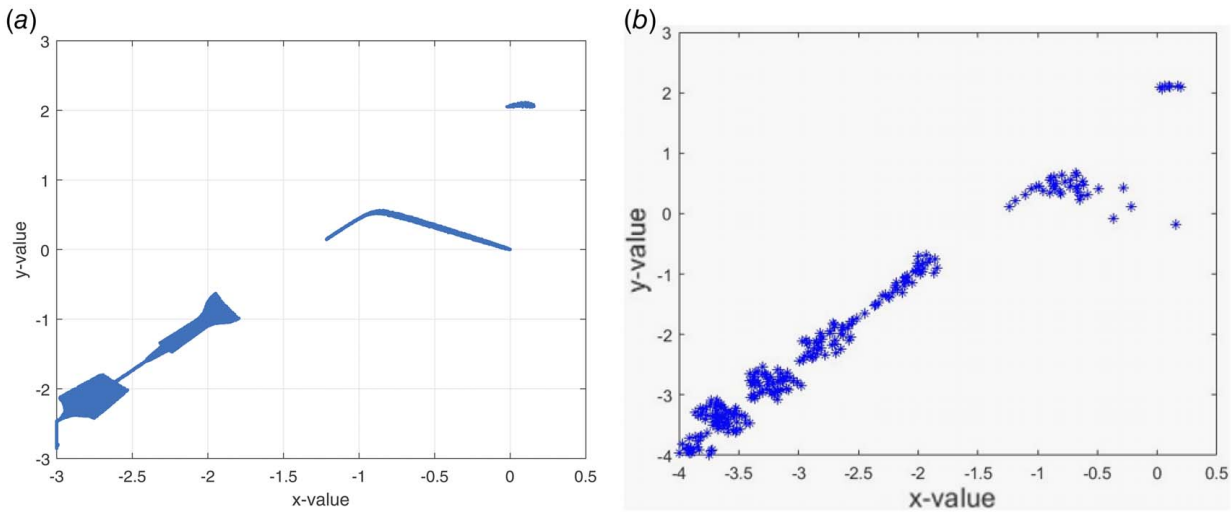

Fig. 8 Theoretically correct nondominated design variables for Viennet's problem [14] (*a*) compared to the ones obtained with MORL-DB (*b*)

Table 7 Performance metrics of three methods employed to solve the quarter of the vehicle model

	MORL-DB	PSI	NSGA-II
# of objective function evaluations	1000	2300	6000
# of Pareto-optimal solutions	638	598	600
Standard deviation of the # of Pareto-optimal solutions	30.9	0	0
Solving time (s)	179.4	0.0175	0.1174
SP (Eq. (14))	0.352	0.374	0.018

Table 8 Performance metrics of three methods employed to solve the Viennet problem

	MORL-DB	PSI	NSGA-II
# of objective function evaluations	4000	600,000	25,000,000
# of Pareto-optimal solutions	533	520	520
Standard deviation of the # of Pareto-optimal solutions	25.3	0	0
Solving time (min)	30.5	23.8	12.59
SP (Eq. (14))	0.197	0.118	0.283

NSGA-II and PSI while NSGA-II performs in the DE segment. As shown in Table 9, NSGA-II proves to be the best method for this test case as it is able to generate a large number of Pareto-optimal solutions in a short time and with only a few iterations. Furthermore, by using a larger population, it is possible to obtain a better approximation of the ideal Pareto front in exchange for a higher computational burden. On the other hand, MORL-DB proves that it is not at the same level as the NSGA-II but still manages to solve even constrained multi-objective optimization problems with more than two design variables with much fewer objective function evaluations than PSI. Moreover, even in this case, the impact of training stochasticity on the distribution and the number of Pareto-optimal solutions is not relevant. However, even in this case, these results were obtained after many hyperparameter tuning iterations.

Due to the high number of episodes, the evolution of the Pareto front generated by MORL-DB during optimization can be appreciated in more detail in this case study. As shown in Fig. 10, the Pareto front obtained with the proposed method continuously improves. Moreover, the higher level of discontinuity of the Pareto front after 25,000 episodes is the consequence of the increase in the number of optimal solutions determined. Finally, as can be seen in the figure, many solutions are discarded despite being very close to being optimal. These designs are generally obtained during the last episodes of the training. Therefore, the level of quality of the suboptimal solutions found using MORL-DB can still be considered quite high.

5.3 Quarter Vehicle Model. The results of the quarter vehicle model case study are represented in Figs. 11–13.

As can be seen in Table 7, even in this case, MORL-DB is capable of generating, even with a limited number of episodes, a large number of Pareto-optimal solutions well distributed within the space of objective functions. In particular, about 60% of the solutions evaluated during the training are part of the Pareto front obtained at the end. This again demonstrates the ability of MORL-DB to find good approximations of the Pareto front with a limited number of evaluations of the objective functions.

Obviously, less accurate approximations of the Pareto front can be obtained with a lower number of episodes. The number of episodes can be modified to define the desired trade-off between accuracy and computational effort.

Again, the optimization process takes longer with respect to the other tested methods because the training of neural networks considerably slows down the approximation of the Pareto front. This is mainly due to the fact that the evaluation of the objective functions requires negligible time. In actual design cases where the evaluation of objective functions could be more time-consuming, the situation could be reversed.

Finally, even in this case study, the stochasticity of the training did not lead to a high variability of the results obtained at each iteration.

The results of the sensitivity analysis presented in Sec. 5.3 are shown in Fig. 14.

As can be seen, the number of Pareto-optimal solutions remains fairly constant. The only parameters that have a significant effect are the learning rate, which in some ranges leads to significant reductions in the number of Pareto-optimal solutions because of the instability of the training process or because the training becomes too slow, and the decay rate, whose rise increases the number of Pareto-optimal solutions but makes their distribution more concentrated in certain regions due to the lower exploration. As far as the elapsed time is concerned, it is mainly influenced by the number of neurons, which increases the training time because of the higher

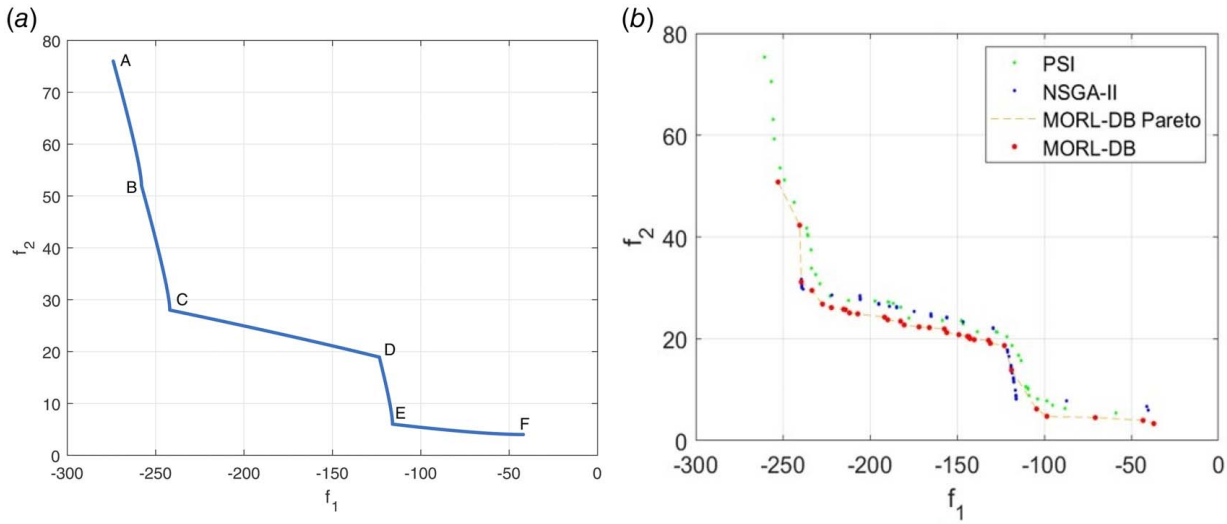

Fig. 9 Theoretically correct nondominated design variables for Osyczka and Kundu's problem [14] (*a*) compared to the ones obtained with MORL-DB, NSGA-II, and PSI (*b*)

Table 9 Performance metrics of three methods employed to solve the Osyczka and Kundu problem

	MORL-DB	PSI	NSGA-II
# of objective function evaluations	25,000	10^8	22,100
# of Pareto-optimal solutions	32	44	100
Standard deviation of the # of Pareto-optimal solutions	2.3	0	0
Solving time (min)	72	79	0.15
SP (Eq. (14))	701.22	788.82	280.48

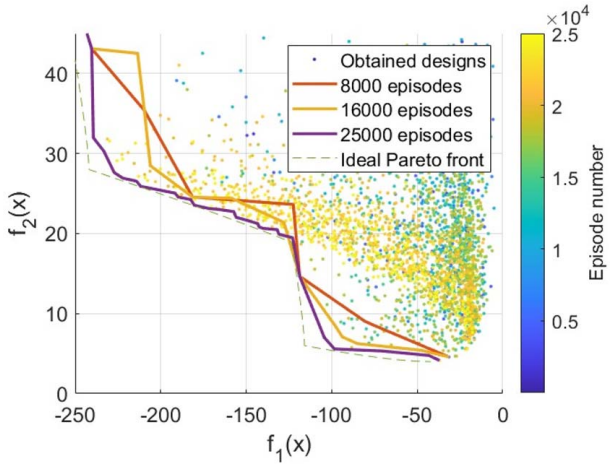

Fig. 10 Pareto front of the Osyczka and Kundu's benchmark problem obtained with MORL-DB after 8000, 16,000, and 25,000 episodes. The points in the plot represent the design solutions tested during the optimization process and their color depends on the number of episode in which they are generated.

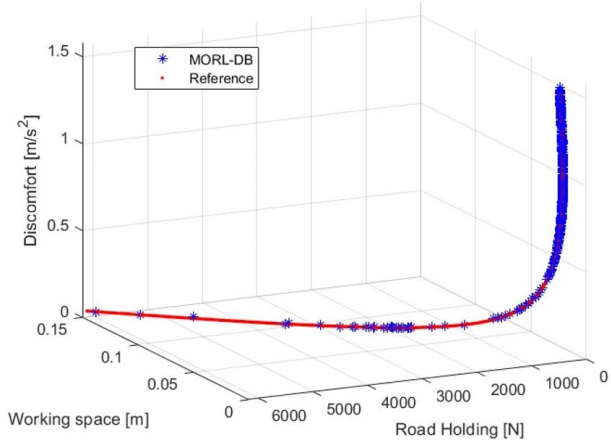

Fig. 11 Pareto front (three objective functions) obtained with MORL-DB for the QVM compared to the theoretically correct one [19]

number of parameters to determine, and by the minibatch size, which, since the default settings are used, also defines the number of warm-up steps. For this reason, a higher minibatch size means a higher number of warm-up steps and thus less training time because of the reduced number of training steps. It is also worth noting the effect of the number of episodes. The number of Pareto-optimal solutions grows almost linearly with this parameter, while the optimization time grows more than linearly. Moreover, it also influences the variability of the results. In the end, as shown by the trend of the standard deviation with the learning rate of the critic,

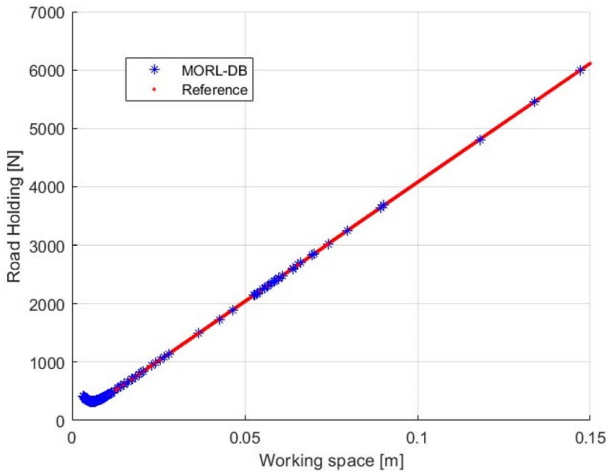

Fig. 12 Projection of the Pareto front on the ($\sigma_{Fz} - \sigma_{x_2-x_1}$) plane. The results obtained with MORL-DB for QVM are compared to the theoretically correct ones [19].

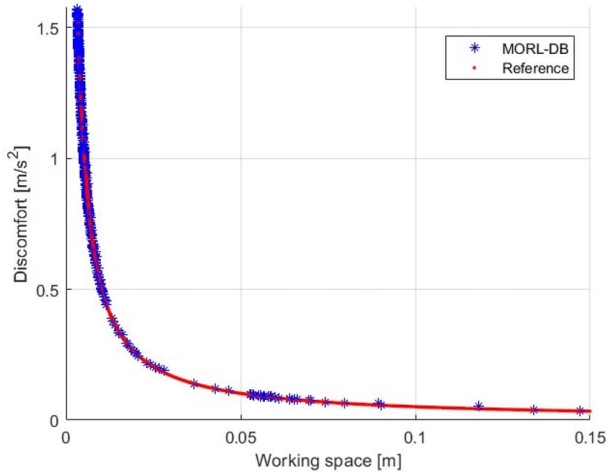

Fig. 13 Projection of the Pareto front on the ($\sigma_{\ddot{x}_2} - \sigma_{x_2-x_1}$) plane. The results obtained with MORL-DB for QVM are compared to the theoretically correct ones [19].

there can be considerable reductions in robustness even with the change of a single parameter. The same thing happens, but to a more limited extent, with the actor's learning rate, the number of neurons, and the decay rate.

In all tests performed during the sensitivity analysis, the algorithm still managed to produce a Pareto front approximation. In contrast, the combinations of hyperparameters used for the other two case studies are less robust, with even a single hyperparameter change potentially causing convergence failure. In these cases, the only way to achieve convergence is through an appropriate combination of all the hyperparameters.

This analysis demonstrates the complexity of the hyperparameter tuning, which requires a lot of time.

6 Conclusion

In this article, a new reinforcement learning-based multi-objective optimization method is presented. The new method, called MORL-DB, is applied to two test cases well known in the technical literature (Viennet and Osyczka and Kundu [14]) and to optimize a road vehicle suspension system. The results are compared with the ones obtained with the nondominated sorting

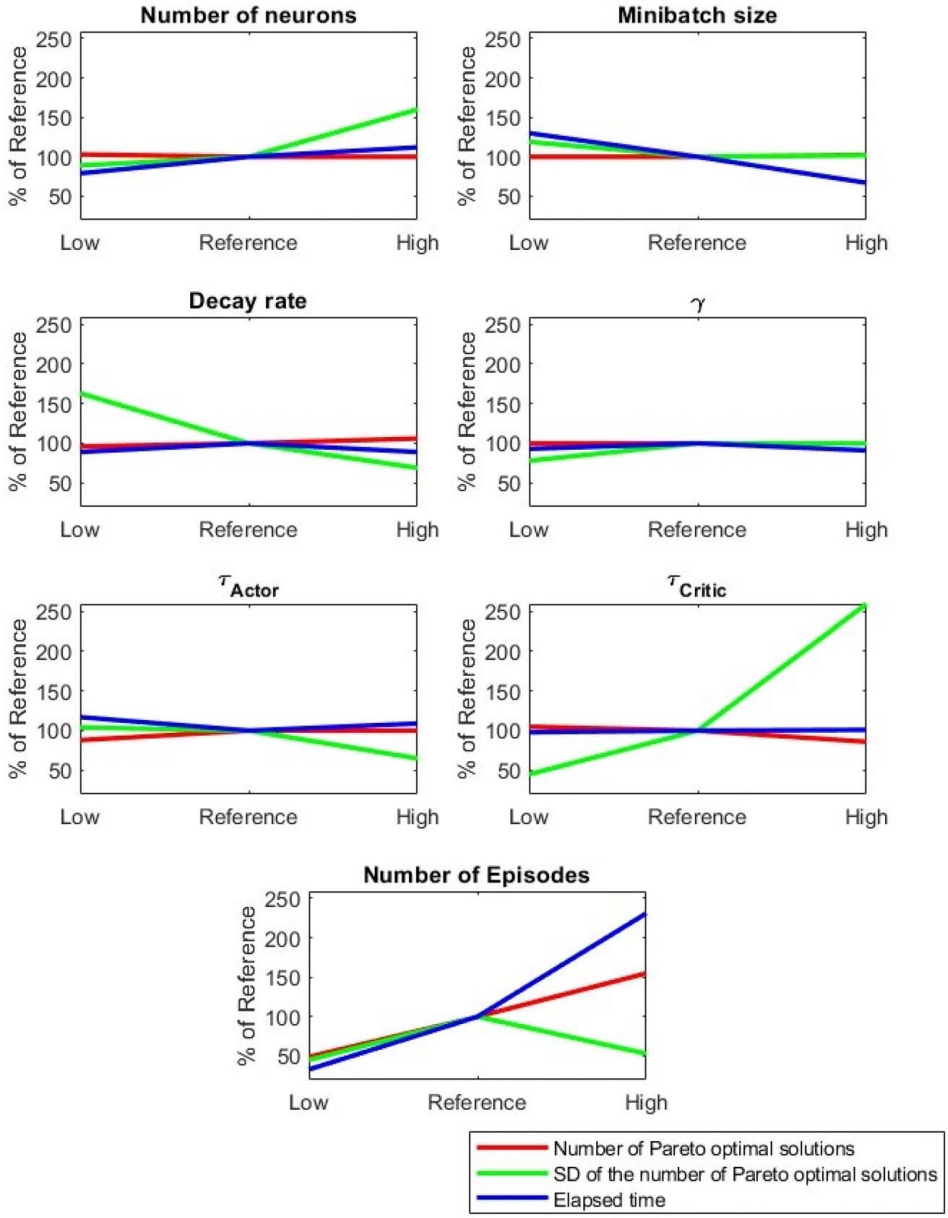

Fig. 14 Sensitivity analysis of the hyperparameters

genetic algorithm (NSGA-II) and PSI. In all test cases, the method demonstrates the ability to generate many Pareto-optimal solutions with few evaluations of objective functions. Moreover, at the end of the training process, the output is not only the Pareto front but also a trained agent that allows one to understand quickly if another design solution will be or not Pareto-optimal. Additionally, MORL-DB managed to obtain a distribution of the Pareto-optimal solutions with an uniformity comparable to PSI. At the same time, it should be considered that the training process slows down the optimization process significantly, and this causes a speed gap with respect to the other tested methods. This gap is very high for the Osyczka and Kundu test problem and for the quarter car vehicle model case study, while it is much less for Viennet's problem. In fact, in Viennet's case study, the training time is partly compensated by a significantly smaller number of objective function evaluations. The main limitation of the MORL-DB algorithm is the need to find a good hyperparameter set to exploit the full potential of reinforcement learning by means of a series of tests. However, in the future, this problem may be less relevant because good combinations of hyperparameters for common problems will be found in the technical literature.

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

The authors attest that all data for this study are included in the paper.

References

[1] Zhang, C., and Lu, Y., 2021, "Study on Artificial Intelligence: The State of the Art and Future Prospects," J. Ind. Inf. Integr., **23**, p. 100224.
[2] Fenjiro, Y., and Benbrahim, H., 2018, "Deep Reinforcement Learning Overview of the State of the Art," J. Autom. Mobile Rob. Intell. Syst., **12**(3), pp. 20–39.

[3] Sharpe, C., Wiest, T., Wang, P., and Seepersad, C. C., 2019, "A Comparative Evaluation of Supervised Machine Learning Classification Techniques for Engineering Design Applications," ASME J. Mech. Des., **141**(12), p. 121404.
[4] Ororbia, M. E., and Warn, G. P., 2023, "Design Synthesis of Structural Systems as a Markov Decision Process Solved With Deep Reinforcement Learning," ASME J. Mech. Des., **145**(6), p. 061701.
[5] Lee, X. Y., Balu, A., Stoecklein, D., Ganapathysubramanian, B., and Sarkar, S., 2019, "A Case Study of Deep Reinforcement Learning for Engineering Design: Application to Microfluidic Devices for Flow Sculpting," ASME J. Mech. Des., **141**(11), p. 111401.
[6] Wang, Z., Zeng, T., Chu, X., and Xue, D., 2023, "Multi-objective Deep Reinforcement Learning for Optimal Design of Wind Turbine Blade," Renew. Energy, **203**, pp. 854–869.
[7] Van Moffaert, K., and Nowé, A., 2014, "Multi-objective Reinforcement Learning Using Sets of Pareto Dominating Policies," J. Mach. Learn. Res., **15**(1), pp. 3483–3512.
[8] Raina, A., Puentes, L., Cagan, J., and McComb, C., 2021, "Goal-Directed Design Agents: Integrating Visual Imitation With One-Step Lookahead Optimization for Generative Design," ASME J. Mech. Des., **143**(12), p. 124501.
[9] Raina, A., Cagan, J., and McComb, C., 2022, "Design Strategy Network: A Deep Hierarchical Framework to Represent Generative Design Strategies in Complex Action Spaces," ASME J. Mech. Des., **144**(2), p. 021404.
[10] He, Z., Tran, K. P., Thomassey, S., Zeng, X., Xu, J., and Yi, C., 2022, "Multi-Objective Optimization of the Textile Manufacturing Process Using Deep-q-Network Based Multi-agent Reinforcement Learning," J. Manuf. Syst., **62**, pp. 939–949.
[11] Barri, D., Soresini, F., Gobbi, M., and Mastinu, G., 2022, "Comparison of Multi-Objective Optimisation Methods for the Design of Electric Motors," 24th International Conference on Advanced Vehicle Technologies (AVT), St. Louis, MO, Aug. 14–17.
[12] Gobbi, M., 2013, "A k, k-ε Optimality Selection Based Multi Objective Genetic Algorithm with Applications to Vehicle Engineering," Optim. Eng., **14**, pp. 345–360.
[13] Durillo, J. J., García-Nieto, J., Nebro, A. J., Coello, C. A. C., Luna, F., and Alba, E., 2009, "Multi-objective Particle Swarm Optimizers: An Experimental Comparison," Proceedings of the 5th International Conference on Evolutionary Multi-criterion Optimization (EMO) 2009, Nantes, France, Apr. 7–10, Springer, pp. 495–509.
[14] Coello, C., Lamont, G., and Veldhuizen, D., 1970, "MOEA Test Suites," *Evolutionary Algorithms for Solving Multi-Objective Problems*, Springer , New York, pp. 175–232.
[15] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. A. M. T., 2002, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," IEEE Trans. Evol. Comput., **6**(2), pp. 182–197.
[16] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., 2015, "Continuous Control With Deep Reinforcement Learning," CoRR.
[17] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M., 2013, "Playing Atari With Deep Reinforcement Learning," preprint arXiv:1312.5602.
[18] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M., 2014, "Deterministic Policy Gradient Algorithms," International Conference on Machine Learning, Bejing, China, June 22–24, pp. 387–395.
[19] Mastinu, G., Gobbi, M., and Miano, C., 2007, *Optimal Design of Complex Mechanical Systems: With Applications to Vehicle Engineering*, Springer Science & Business Media, Berlin, Heidelberg.
[20] Statnikov, R., and Statnikov, A., 2011, *The Parameter Space Investigation Method Toolkit*, Artech House.
[21] Monsef, H., Naghashzadegan, M., Jamali, A., and Farmani, R., 2019, "Comparison of Evolutionary Multi-objective Optimization Algorithms in Optimum Design of Water Distribution Network," Ain Shams Eng. J., **10**(1), pp. 103–111.
[22] The MathWorks, Inc., 2024, "MATLAB and Reinforcement Learning Toolbox Release 2024a," The MathWorks, Inc., Natick, MA, Release Notes.
[23] Gobbi, M., Levi, F., and Mastinu, G., 2006, "Multi-objective Stochastic Optimisation of the Suspension System of Road Vehicles," J. Sound Vib., **298**(4–5), pp. 1055–1072.
[24] Han, Z. H., Xu, C., Zhang, L., Zhang, Y., Zhang, K., and Song, W., 2020, "Efficient Aerodynamic Shape Optimization Using Variable-Fidelity Surrogate Models and Multilevel Computational Grids," Chin. J. Aeronaut., **33**(1), pp. 31–47.