Rule-based reinforcement learning methodology to inform evolutionary algorithms for constrained optimization of engineering applications

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Rule-based Reinforcement Learning Methodology to Inform Evolutionary Algorithms for Constrained Optimization of Engineering Applications*

Majdi I. Radaideh*, Koroush Shirvan

Department of Nuclear Science and Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, United States

ARTICLE INFO

Article history:
Received 17 August, 2020
Received in revised form 6 December, 2020
Accepted 30 January 2021
Available online 5 February 2021

Keywords:
Reinforcement Learning
RL-guided Evolution
Proximal Policy Optimization
Combinatorial Optimization
Nuclear Fuel Assembly
Evolution Strategies

ABSTRACT

For practical engineering optimization problems, the design space is typically narrow, given all the real-world constraints. Reinforcement Learning (RL) has commonly been guided by stochastic algorithms to tune hyperparameters and leverage exploration. Conversely in this work, we propose a rule-based RL methodology to guide evolutionary algorithms (EA) in constrained optimization. First, RL proximal policy optimization agents are trained to master matching some of the problem rules/constraints, then RL is used to inject experiences to guide various evolutionary/stochastic algorithms such as genetic algorithms, simulated annealing, particle swarm optimisation, differential evolution, and natural evolution strategies. Accordingly, we develop RL-guided EAs, which are benchmarked against their standalone counterparts. RL-guided EA in continuous optimisation demonstrates significant improvement over standalone EA for two engineering benchmarks. The main problem analyzed is nuclear fuel assembly combinatorial optimization with high-dimensional and computationally expensive physics. The results demonstrate the ability of RL to efficiently learn the rules that nuclear fuel engineers follow to realize candidate solutions. Without these rules, the design space is large for RL/EA to find many candidates. With imposing the rule-based RL methodology, we found that RL-guided EA outperforms standalone algorithms by a wide margin, with >10 times improvement in exploration capabilities and computational efficiency. These insights imply that when facing a constrained problem with numerous local optima, RL can be useful in focusing the search space in the areas where expert knowledge has demonstrated merit, while evolutionary/stochastic algorithms utilize their exploratory features to improve the number of feasible solutions.

1. Introduction

Gradient-based reinforcement learning (RL) [1] and gradient-free evolutionary algorithms (EA) [2] have been used to complement each other in many different applications by blending learning with evolution and vise versa [3]. The interaction between RL and EA has begun a while ago [4, 5], but started to grow recently after the introduction of deep neural networks in RL [6, 7]. In the literature, most studies explore using EA to guide RL and vise versa.

Integration of EA with RL, which is more common, can be classified in three major categories: (1) a tool to tune RL hyperparameters, (2) an alternative approach to train neural networks instead of gradient descent and backpropagation, and (3) an assisting tool to enhance RL exploration via population-based search. A population-based training approach is proposed by [8] to tune RL neural

*Corresponding author: radaideh@mit.edu (M. I. Radaideh).

networks, including its model architecture, learning rate, loss function, and the internal optimization algorithm. Hierarchical genetic representation scheme was also proposed by [9] to tune neural network architectures in image classification, outperforming random search. Hyperparameters of SARSA and Q-learning algorithms were tuned by the genetic algorithm (GA) [10], leading to a maximized reward and increasing learning speed. The second category is usually referred to as neuroevolution [11]. Earlier efforts featured using NEAT, an evolutionary algorithm that leverages both neural architecture and weights to optimize function approximators corresponding to the value function in Q-learning [12, 13]. A survey of using EA as an alternative to temporal credit assignment and value function in RL is conducted by [5] in the past. After deep learning revolution, the work by [14] demonstrated evolution strategies as an alternative approach to backpropagation and value function in training deep RL agents, which proved to be competitive in Atari games, and highly scalable with number of CPUs. Similar scalability and competitive results of deep neuroevolution were also achieved by [15], which utilized GA and novelty search to train agents of deep Q learning and Asynchronous Advantage Actor-Critic (A3C) [16]. For the last category, EA in form of GA was used to leverage a population of neural networks, to provide diversified data to train RL agents with improved exploration ability [17]. The tests with EA-guided RL seemed to handle the sparse rewards issue and outperform standalone RL in several Atari games [17]. Additional example was demonstrated by [18] on using an EA called goal exploration processes to generate diverse samples for RL first, followed by fine-tuning policy parameters using regular policy gradient techniques. An earlier application of evolutionary RL to perform autonomous vehicle navigation was performed by [4].

The use of RL to guide EA, which is the main focus of this work, is more limited. RL-guided EA have been used for two main applications [19]: (1) online parameter control of EA and (2) design of hyper-heuristics for combinatorial optimization, which are an ensemble of heuristics that automatically adapt to search for the optimal solution [20]. Controlling GA behaviour using Q-learning was done by [21], the RL agent adapts both the crossover and mutation operators as well as individual selection for offspring at every generation. Similar effort was performed by [22], however, utilizing SARSA as the RL algorithm. In application contexts, RL was utilized for combinatorial optimization through adaptive parameter control of GA to solve the travel salesman problem [23]. In addition, Q-learning was used to update the crossover and mutation operators of GA to facilitate optimising multidimensional data discretization [24]. For hyper-heuristics research, the survey by [25] covers advances on hyper-heuristics, which include variety of examples of RL usage, a few can be found here [26, 27, 28]. The idea is using RL to reward or punish each heuristic during EA search, based on its individual performance or the reward gain from its usage to ensure an optimal ensemble during search. RL has always been perceived as an effective tool for action preferences to learn heuristics due to its dynamic nature [29]. The work by [30] presented a study on using RL to learn certain heuristics, and then RL was used to pick an initial guess for a simulated annealing chain. Additional examples of how RL can be effective in decision support can be seen in several efforts such as shaping game strategies [31], multi-objective planning of actions in autonomous control tasks [32], supporting large-scale service composition in service-oriented systems [33], and many more. Lastly, an interesting and comprehensive study of RL, EA, and the history of their hybridization can be found in this recent review [19].

Unlike the previous applications of hybrid RL and EA, this work explores the possibility to use modern deep RL, namely, proximal policy optimization (PPO) to embed domain knowledge in form of rules to solve optimization problems with excessive constraints, which are likely in large-scale engineering optimization. Therefore, the novelty of this work originates from exploring the benefits of hybridizing RL and EA to solve a certain class of optimization problems, namely, constrained optimization of both combinatorial and continuous natures. We demonstrate how standalone EA and RL algorithms start struggling as the search space becomes more confined, while RL-guided EA algorithms under similar settings and hyperparameters can maintain an excellent performance. To support our methodology, a real-world nuclear engineering application with a challenging combinatorial nature is selected to test the proposed algorithms. In addition, two engineering benchmark problems from mechanical and structural engineering are selected to test the performance of the proposed algorithms in continuous optimisation. The RL model is first used to learn some of the problem constraints/rules, and then used to guide EA search through taking proper actions to ensure the search is taking place in feasible regions, leading to more robust search and cost savings.

As our focus in this work will be on the nuclear assembly combinatorial problem due to its complexity along with its importance to our engineering background, most of this paper is devoted to this problem. The other two engineering problems are highlighted in more brief forms, as they are more well-known optimisation engineering benchmarks in the literature. Accordingly, the novelty of the proposed concept is motivated by at least one of the following reasons: (1) effective exploration of heavily-constrained search spaces, (2) computational efficiency of RL-guided EA, (3) handling the problem complexity, and (4) the practical value of optimization. We demonstrate our methodology through developing RL-guided genetic algorithm, RL-guided simulated annealing for combinatorial optimisation, and RL-guided parti-

cle swarm optimisation, RL-guided differential evolution, and RL-guided natural evolution strategy for continuous optimisation. Then, we test all algorithms against their standalone counterparts, under heavily-constrained optimisation problems.

A common optimization problem drawn from nuclear reactor design is described in section 2, which is used to demonstrate the RL-guided concept on discrete optimisation. The problem has several challenges of being multi-objective, combinatorial, constrained, high-dimensional, and expensive (i.e. computer simulation is needed for fitness evaluation). As such, while the standalone versions of RL [34] and EA [35, 36] have been applied to such problems in the past, the nuclear industry still relies on expert input given that a brute force approach is not feasible due to the prescribed challenges. Nevertheless, solving such problems efficiently can lead to a great improvement in nuclear fuel efficiency, cost reduction, and nuclear safety assurance, which is why the largest nuclear power plant operator in the United States (Exelon Corporation) is supporting this work. In Appendices A - B, two common engineering benchmark problems are briefly described to demonstrate the RL guidance concept in continuous optimisation, which are the speed reducer design and the welded beam design. In section 3, the methodology is described on using the RL algorithm PPO to inform genetic algorithm (GA), simulated annealing (SA), particle swarm optimisation (PSO), differential evolution (DE), and natural evolution strategies (NES). We should highlight that SA does not belong to the EA family, but to the bigger family of stochastic optimisation. The five algorithms have been widely used as optimization algorithms, especially for the nuclear optimization problem investigated in this work. Afterward, in section 4, we present our findings on using RL-guided EA algorithms to efficiently solve the prescribed engineering problems, highlighting their computational and search capabilities compared to the standalone (unguided) algorithms, followed by the discussion of these findings in section 5. Finally, the conclusions of this work and outlooks on future work are presented in section 6.

2. Engineering Optimisation Problem Set

In this section, we describe the three engineering problems analyzed in this work. The first problem, the focus of this work, is discrete/combinatorial problem, described in more detail in this section as limited information is available in the literature. The second and third problems are continuous problems, described in brief in Appendices A - B, as these problems are widely investigated in the literature.

The main problem analyzed in this work is nuclear fuel assembly optimisation. A top view of a sample nuclear fuel assembly design is sketched in Figure 1. The assembly is based on commercial boiling water reactor design [37, 38], and it has a dimension of 10x10. For the sample design in Figure 1, there are four types of rods. First, 74 pure Uranium Oxide (UO_2) fuel rods, which are enriched to a specific U-235 enrichment, e.g. 4.4% weight fraction to drive fission reactions. Second, 18 poison rods, each consists of UO_2 fuel mixed with Gadolinium Oxide (neutron poison) to absorb neutrons and help in controlling the fission reaction. Poison rods are represented in pair (X, Y), where X is the UO_2 enrichment and Y is the poison enrichment, e.g. (4.4%, 8%). Third, border rods are regular UO_2 fuel rods, but with smaller enrichment than the interior fuel rods to improve the performance through reducing neutron leakage from the borders. In this work, the border rods are applied as fixed external frames, thus are not included in the optimization process. The last type is two large water rods, occupying the remaining 8 positions at the center to enhance assembly cooling and neutron moderation. The assembly design follows 1/2 symmetry, so only half of the assembly needs to be optimised, and by excluding the border rods, 32 rod locations will be optimized, numbered in Figure 1. The numbered rods on and below the diagonal line in Figure 1 are included in the optimization process.

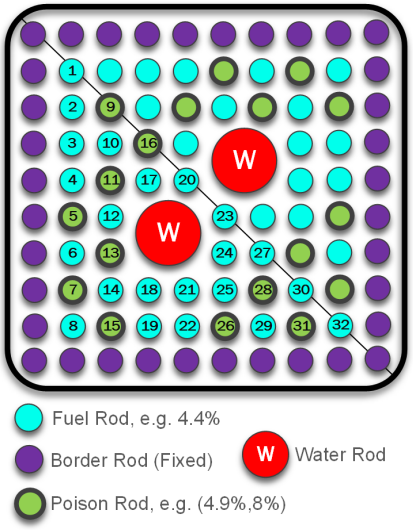

Fig. 1. Top view of the nuclear fuel assembly analyzed in this study with 1/2 symmetry

For the rod items 1-32 in Figure 1, we consider eight combinatorial choices in each location. The first four choices belong to the fuel rod set F_{set} = {3.6%, 4.0%, 4.4%, 4.95%}, while the rest belongs to the poison rod set P_{set} = {(4.4%, 7%), (4.95%, 7%), (4.4%, 8%), (4.95%, 8%)}. Accordingly, the full combinatorial search space is proportional to $O(m^n)$, i.e. $8^{32} \approx 10^{29}$, which is computationally prohibitive to be analyzed by brute force search, given each combination (i.e. assembly pattern) requires computer simulation to be evaluated, as will be described later in this section. What makes the problem even more

challenging is that the design of nuclear fuel assemblies in general follows a heavily-constrained optimization process by adhering to several rules. These rules, which will be used to shape the reward/fitness function for RL and EA, can be summarised as follows:

1. Rule 1 ($r1$): focuses on assembly patterns with small number of poison rods, $N_{poison} < 25$ (green rods in Figure 1), since large N_{poison} overly kills neutrons and suppresses power production. Therefore, if $N_{poison} \geq 25$ and the RL agent attempts to take another poison action, a -1 penalty is applied, and the poison rod is replaced by a random choice from F_{set} (e.g. 4.0%)

$$V_1 = \sum_{t=1}^{T} \mathbf{1}_{\{N_{GAD} \geq 25 \cap a_t \in P_{set}\}}, \tag{1}$$

   where $\mathbf{1}$ is the indicator function and a_t is the action taken at time step t.
2. Rule 2 ($r2$): After passing rule 1, the optimization should focus only on assembly patterns with a tight N_{poison} range, for this work, $N_{poison} \in [16, 18]$. The violation of this rule can be quantified using relative difference $V_2 = \left|\frac{N_{poison}-b}{b}\right|\% + p$, where b can be either 16 or 18 depending on N_{poison} value. p is a penalty factor between [0,400] to handle a caveat in this rule. The p value is determined based on the first term of V_2, which proved to improve the learning stability of this rule. Patterns with N_{poison} outside this range are discarded. The reader may notice at the first sight that Rule 1 and Rule 2 are redundant, however, they are indeed complementary. As the RL agent starts the training by taking many poison actions (> 40), which are undesirable, the penalty of Rule 1 helps accelerating the movement to the low-poison search regions. Then Rule 2 can balance the number of poison rods between 16 and 18. Dropping Rule 1 mathematically has no impact, but computationally slows down the optimisation.
3. Rule 3 ($r3$): focuses on balancing the average UO_2 enrichment (E) of the whole assembly (for all fuel, poison, and border rods) within a tight bound determined by the designer, in this work, $E \in [4.25\%, 4.36\%]$. This rule is an economic rule to minimize E, while maintaining effective and safe performance, since high E requires additional costs. The violation of this rule is quantified by relative difference, $V_3 = 50\left|\frac{E-b}{b}\right|\%$, where b can be either 4.25% or 4.36% depending on E value.
4. Rule 4 ($r4$): this rule handles the position of poison rods, where two poison rods are not allowed to neighbour each other in the assembly either vertically or horizontally, diagonally is allowed, see Figure 1 for typical poison rod positioning. This rule is very important for the nuclear reactor safety, and hence any pattern with at least two poison rods violating this condition is discarded. After counting individual violations ($N_{violate}$), the violation of this rule is expressed by $V_4 = 10 * N_{violate}$.

The previous rules can be perceived as constraints on the input space or assembly layout. Three additional constraints must be met in the output space, which is a function of the input space:

1. Rule 5 ($r5$): the infinite neutron multiplication factor (k_∞) is an important safety parameter, which is a measure of the change in the fission neutron population in the assembly analyzed in this work, and usually expressed in 5 significant digits. k_∞ should be maintained below a threshold value during the simulation, i.e. $k_\infty \leq k_\infty^{max}$. We will define k_∞^{max} later in section 4 as we will use different threshold values to demonstrate the methodology by imposing additional confinement on the space. If k_∞ condition is not met, the pattern is discarded.
2. Rule 6 ($r6$): power peaking factor (PPF), which is the ratio of the highest fuel rod power in the assembly to the average assembly power by all rods. Likewise k_∞, PPF is a stringent safety parameter to be maintained below a threshold value, $PPF \leq PPF^{max}$, and similarly PPF^{max} values will be defined later in section 4.
3. Rule 7 ($r7$): cycle length (CL), which is a measure of the operating time the assembly can provide power inside nuclear power plant before it is completely depleted. In this work, maximizing CL forms the main objective function once all previous constraints, $r1 - r6$, are met. We use the "burnup" unit GWD/MTU to express CL (**G**iga**W**att **D**ays per **M**etric **T**on of **U**ranium), which expresses the power production per day normalized by the initial fuel loading. CL should be maximized and maintained to $CL \geq CL^{min}$ GWD/MTU to ensure economic operation, where CL^{min} values will be defined later in section 4.

The violation of $r5 - r6$ rules (i.e. $V_5 - V_6$) can be quantified similarly through relative difference by determining the deviation from the corresponding threshold of each rule (i.e. k_∞^{max}, PPF^{max}). Our choices of the bounds and thresholds above are not arbitrary, but based on commercial designs in the literature. The summary of the seven rules and the objective is given in Table 1. It is obvious that we have used an approach similar to the common ϵ-constraint approach in multi-objective optimisation [39], where Rules 1-6 are treated as constraints, while Rule 7 (CL) is treated as our single objective to maximize. In addition, according to the previous descriptions, we typically used the penalty factor approach for constraint handling [40] of the rules above, which are of inequality type.

Therefore, the optimiser is penalized when the solution deviates largely from the constraint.

To obtain a realistic physics representation and accurate values for the figure of merits: k_∞, PPF, and CL, an advanced computer simulation is needed. The remaining rules ($r1 - r4$) can be evaluated by averaging and simple mathematical formulas. Therefore, the nuclear assembly geometry is modeled and simulated using the CASMO4 code. CASMO4 [41] is a multigroup two-dimensional neutron transport code for nuclear time-dependent calculations of boiling water reactor fuel assemblies, based upon the Method of Characteristics. More details about CASMO4 methodology can be found in the code manuals and related research utilizing CASMO4 [41, 42, 43]. For this study, each CASMO4 simulation needs about 1.5 minutes to complete, and hence 1.5 minutes will be used as the basis to obtain an insight about the computing time saving by all algorithms. Although using simplified physics analytical or surrogate models can be beneficial to accelerate optimisation, the reader should be careful about using these models as they can give inaccurate values about the objective function and the constraints.

Accordingly, we can formulate the optimisation problem mathematically as follows:

$$\max_{\vec{x}} CL(\vec{x}) = CASMO4(\vec{x}), \tag{2}$$

subject to $r1(\vec{x})$-$r6(\vec{x})$, where $\vec{x}$ is the vector of fuel choices in all 32 locations of Figure 1. The problem statement can be described as follows: *What are the optimal fuel choices $\vec{x}$ (from P_{set} and F_{set}) in all 32 rod locations in Figure 1, such that all six rules can be satisfied simultaneously and r7 is maximized above CL^{min}.*

Due to the large size of the search space $\sim 10^{29}$ and the expensive simulation time to evaluate each individual (1.5 minutes), we cannot know the global optimal solution, which has the maximum CL with all rules/constraints satisfied. Alternatively, we use an implicit constraint ($CL \geq CL^{min}$) in the fitness objective of Eq.(2), where we filter the patterns with satisfactory CL from an engineering perspective. Consequently, *once an assembly pattern is found with all rules met including the implicit rule on CL, this pattern is called "candidate" pattern*. Our objective is then to maximize the number of these candidate patterns to better inform the final assembly design. The reader should notice that all algorithms are programmed to maximize the objective in Eq. (2), and the candidate pattern alternative is nothing but a postprocessing step of the fitness function. Also from an engineering point of view, in the real nuclear reactor design problem, the designer will test out the top candidate assemblies in the full reactor design (consists of over 500 assemblies). Therefore, the larger the candidate patterns to select from, the higher the performance attained by the designer in the next step of the nuclear reactor design process.

Finally, it is worth highlighting that a physics-based environment is developed under the OpenAI Gym [44] toolkit to allow interactions between CASMO4 assembly model and RL/EA algorithms. OpenAI Gym provides data structures to test RL algorithms, where we have added additional functions to the environment to facilitate CASMO4 processing as well as EA fitness evaluation. Using OpenAI gym is advantageous in this work due to its compatibility with the RL algorithm and platform we use in this work, which are both described next.

3. Methodology

In this section, we describe the reinforcement learning methodology first, then we describe the five RL-guided algorithms: RL-guided GA/RL-guided SA for combinatorial optimisation, and RL-guided PSO/RL-guided DE/RL-guided NES for continuous optimisation.

3.1. Reinforcement Learning Proximal Policy Optimization

The recent review by Bengio et al. [45] highlighted different machine learning approaches to tackle combinatorial optimisation, among these, RL is highlighted as an intuitive choice. RL learned by the "experience" setting of [45] can be used in this work. The policy is learned by taking actions in trial/error form accompanied by a reward signal for these actions, where the reward is calculated as described in section 2 for the nuclear assembly, and Appendices A - B for the other engineering problems. For the nuclear assembly, after sufficient training, the policy will learn how to arrange the fuel in Figure 1 such that the reward return (i.e. cycle length) is maximized. Also, the process is Markovian with full observability, as we have access to all problem states, which are the fuel type in each position. These conclusions are also applicable to the speed reducer and welded beam problems.

Proximal policy optimization (PPO) belongs to the policy gradient (PG) family, which is a RL family that trains a policy to map states to actions by optimising the following loss function

$$L^{PG}(\theta) = E_t[log\ \pi_\theta(a_t|s_t)A_t], \tag{3}$$

where E_t is the expectation over a batch of transitions, π is the policy to be optimised which has weights θ, and A_t is the advantage estimate, which is controlled by γ (the discount factor) and λ (the bias-variance tradeoff parameter). For brevity of this work, we refer the reader to this reference [46] for full mathematical details about generalized advantage estimation and its hyperparameters (γ, λ). The policy π predicts action a given state s at time step t. Vanilla PG suffers from poor data efficiency, noisy behaviour, and limited exploration [47]. These limitations inspired several efforts to improve PG performance, referenced here for conciseness: deep deterministic policy gradient [7], Trust-Region Policy Optimization [48], and

Table 1. Summary of the rules/constraints to match for the problem of interest (see Figure 1)

Rule ID	Rule Type	Description
$r1$	Constraint	Maintain poison rods in low levels ($N_{poison} < 25$)
$r2$	Constraint	Only consider patterns with $N_{poison} \in [16-18]$ rods
$r3$	Constraint	Only consider patterns with average enrichment $E \in [4.25\%, 4.36\%]$
$r4$	Constraint	Only consider patterns with proper poison positions
$r5$	Constraint	Only consider patterns with $k_\infty \leq k_\infty^{max}$
$r6$	Constraint	Only consider patterns with $PPF \leq PPF^{max}$
$r7$	Objective	Maximize CL as final objective, only consider patterns with $CL \geq CL^{min}$

then PPO [47]. PPO is observed as sample inefficient compared to its companion, deep Q-learning [6], since it discards previous experiences after updating the surrogate every F timesteps (i.e. no replay memory). Nevertheless, this sacrifice allows PPO to have simpler implementation, faster training, easier hyperparameter tuning, and efficient parallel calculations [47], which are all of high importance to this work.

The PPO algorithm used in this work is shown in Algorithm 1, which can be summarised in two steps. In the first step, transitions (i.e. sequence of states, rewards, and actions) are collected based on old policy ($\pi_{\theta_{old}}$) interactions with the environment for a full time horizon F. In parallel calculations, $F = NT$, where N is number of agents/processors running in parallel, and T is the time horizon of each agent.

In the second step, the old policy is updated by optimising the neural network model, i.e. commonly known as the "surrogate" by running gradient descent over the surrogate objective L^{PPO}, defined below, with learning rate lr, mini-batch of size B, and for number of supervised epochs opt_{epochs}. First, we can write the clipped PPO surrogate loss function as [47]

$$L^{CLIP}(\theta) = E_t[\min(\underbrace{R_t(\theta)A_t}_{\text{Modified PG Objective}}, \underbrace{\text{clip}(R_t(\theta), 1-\omega_{clip}, 1+\omega_{clip})A_t}_{\text{Clipped Objective}})], \tag{4}$$

where $R_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between the new and old policies, and ω_{clip} is the clip range. The first term in the *min* function is the modified PG objective after including the trust-region [48], while the second term modifies the objective by clipping the probability ratio to remove the incentive for moving R_t outside of the interval [1 - ω_{clip}, 1 + ω_{clip}]. Two additional improvements have been added to the clipped PPO objective to improve the performance, see [47] for more details. First, PPO stability is enhanced through adding a value function (VF) loss term (L_t^{VF}), controlled by the coefficient ω_{vf}. Second, PPO exploration is enhanced through an entropy term ($S[\pi_\theta]$), controlled by the coefficient ω_s. The three parameters, ω_{clip}, ω_{vf}, ω_s should be tuned for better performance. By combining the three terms, the PPO final loss function can be written as follows [47]

$$L^{PPO}(\theta) = E_t[\underbrace{L_t^{CLIP}(\theta)}_{\text{Clipping Term}} - \underbrace{\omega_{vf} \cdot L_t^{VF}(\theta)}_{\text{Value Function Loss}} + \underbrace{\omega_s \cdot S[\pi_\theta](s_t)}_{\text{Entropy Term}}]. \tag{5}$$

Algorithm 1 Reinforcement Learning Proximal Policy Optimization [47]

```
•Set hyperparameters: γ, λ, ω_clip, ω_vf, ω_s, lr, B, opt_epochs
•Initialize policy π_θold with random weights
for Epoch i = 1 to EPOCH do
    for Agent j = 1 to N_cores do
        •Run policy π_θold in the environment for time horizon T
        •Compute rewards and advantage estimates A_1, ..., A_T
    •Optimise surrogate L^PPO(θ) in Eq.(5) with respect to θ using:
    •Running gradient descent with learning rate lr, mini-batch B ≤ NT, for opt_epochs
    •Set θ_old ← θ
    •Calculate mean reward for the current epoch and save the agent if performance improved
```

The mapping of RL to the optimization problem involves defining a discrete action space of size 8, which includes all fuel choices in F_{set} and P_{set}. The action space can be defined using the **Discrete** type of OpenAI gym. The state space is a vector of size 32, where each entry highlights the current fuel type in each of the 32 positions in Figure 1. The state space can be defined using the discrete **Box** type of OpenAI gym. The Box type allows defining multiple entries with minimum and maximum limits, where for this study, the 32 entries representing the positions are restricted between 1 to 8 to represent the current fuel type in each position. The agent randomly visits each rod location, picks an action (i.e. fuel choice) from the action space, observes the reward (based on rules $r1-r7$ and Eq.(2)), updates the state space (Box), and then moves to the next position. This process is repeated until all 32 positions are visited, then the episode is terminated. The process is repeated for specific number of time steps, until satisfactory learning or reward is achieved. For the two problems in Appendices A - B, since they are contin-

Table 2. RL/EA hyperparameters and their tuning range

Parameter	Method(s)	Range
γ	PPO	Fixed to 0.99
B	PPO	$2^n, n = 2, 3, ..., 10$
lr	PPO	Fixed to 2.5×10^{-4}
T	PPO	400-5000 (N=20 cores)
λ	PPO	0.9-1.0
ω_{clip}	PPO	0.1-0.3
ω_{vf}	PPO	0.5-1.0
ω_s	PPO	0-0.03
opt_{epochs}	PPO	Fixed to 15
χ	GA/SA/RL-guided GA/SA	0.005-0.2
N_{pop}	GA/NES/DE and RL-guided GA/NES/DE	20-80
CX	GA/RL-guided GA	0.3-0.9
MUT	GA/RL-guided GA	0.025-0.4
F	DE/RL-guided DE	0-2.0
C	DE/RL-guided DE	0.5-1.0
η_μ	NES/RL-guided NES	Fixed to 1.0
η_B	NES/RL-guided NES	0.05-0.3
η_σ	NES/RL-guided NES	0.05-0.3
$N_{particles}$	PSO/RL-guided PSO	50-150
c_1/c_2	PSO/RL-guided PSO	$2.01 - 2.2$
N_{rl}	RL-guided GA/PSO/DE/NES	[0.02-0.3] $\times$ N_{pop} or $N_{particles}$
T_{max}	SA/RL-guided SA	1000-150000
T_{min}	SA/RL-guided SA	Fixed to 1
$Cooling$	SA/RL-guided SA	Fast, Boltzmann, Cauchy, Eqs.(6)-(8)
χ_{rl}	RL-guided SA	0.01-0.2

uous in nature, both the action and state spaces are of type **Box**, where the minimum and maximum limits are set to the ranges of the design variables of each problem.

As stable-baselines [49] provides a modern implementation of PPO based upon the algorithm proposed by [47], stable-baselines-2.10.0 package is used for PPO implementation in our work. According to the previous descriptions, we tune the following PPO hyperparameters: {T, λ, ω_{clip}, ω_{vf}, ω_s, B}, while other parameters {$\gamma = 0.99$, $lr = 2.5 \times 10^{-4}$, $opt_{epochs} = 15$} are fixed to standard values due to their less sensitivity. PPO hyperparameters and their reasonable range are listed in Table 2 (based on some RL packages [49] and our preliminary tests). We did not change the value function settings than what are implemented in stable-baselines-2.10.0, we only tuned the value function loss parameter ω_{vf} for optimal performance. Also, the policy network architecture is set to the default, which is a feedforward neural network with two layers, 64 nodes each. In general, the default policy network and value function settings seem adequate for this work. Lastly and fortunately, as stable-baselines and OpenAI gym platforms are very compatible, the coupling of the nuclear assembly, speed reducer, and welded beam environments with PPO algorithm to perform RL is very straightforward.

3.2. RL-guided Genetic Algorithm

Genetic algorithms (GA) [50] are inspired by the theory of natural evolution, where the fittest individuals are selected to produce offspring of the next generation. GA is one of the widely used algorithms in nuclear power plant optimization; therefore it is selected as an evolutionary candidate for the combinatorial optimisation part of this work. Different versions of GA have been used to solve various optimization problems in pressurized and boiling water reactors [36, 51, 52, 53]. For a specific generation, GA performs the following processes: (1) select individuals for mating/crossover with probability CX, (2) select individuals for mutation with small probability MUT, (3) generate the next offspring (N_{pop}) based on steps 1 and 2, (4) fitness evaluation of the offspring, and (5) select the top individuals based on their fitness value to participate in the next generation. The initial population is generated randomly from the search space, while GA fitness function follows the rules described in section 2. The crossover operation is the classical two-point crossover, where two random individuals are selected for mating with probability CX. Two points are selected randomly from the individual attributes. The attributes in between the two points are swapped between the individuals, resulting in two new individuals. Additionally, some of the population individuals may be selected for mutation with probability MUT. Since the problem to be solved is of combinatorial nature (i.e. nuclear assembly), the attributes of the selected individual are mutated with probability χ with an integer uniformly drawn between the lower and upper bounds of each attribute. The integers in this work are used to encode the fuel types in F_{set} and P_{set}.

In this work, additional hyperparameter called $N_{rl} \in$

$[1, N_{pop}]$ is introduced to improve GA, which represents the number of individuals introduced into the population from external source, in this case, a pre-trained RL/PPO model. RL individuals can be perceived as elite foreign children imported from other generations to enrich the GA generation. N_{rl} is a fraction of N_{pop} introduced into the population before applying crossover and mutation to all individuals, and it can be tuned for desirable performance. Beside blending RL individuals into the population in every generation, the RL model is used to initialize the first population of GA. The RL-guided GA algorithm is described in Algorithm 2. Notice that only two modifications are done on GA to form RL-guided GA, and only one new hyperparameter (N_{rl}) is introduced (i.e. hyperparameter tuning is not an issue for RL-guided GA). In addition, in Algorithm 2, only RL is used to update GA during the RL-guided GA search, while GA does not affect RL search. This would facilitate the extension of the RL-guided concept to other EA algorithms as no changes in RL are needed. In summary, for GA or RL-guided GA, the hyperparameters to tune are: $\{N_{pop}, MUT, CX, \chi, N_{rl}\}$, and their reasonable values are listed in Table 2. Lastly, DEAP-1.3.0 [54], which is an evolutionary computation framework, is used to build GA and our RL-guided GA algorithms. Significant changes have to be performed to couple RL with GA under the DEAP framework.

The mapping of GA and RL-guided GA to the optimization problem involves defining individuals, which are vectors of size 32. Each entry represents a fuel choice from F_{set} or P_{set}. Each individual is evaluated based on the rules $r1 - r7$ to determine the fitness value and the validity of this individual.

As GA was originally developed for combinatorial optimisation, several variants of genetic-based algorithms are developed for continuous optimisation. Since we will explore the RL guidance effect on continuous optimisation in section 4.1, Appendix A, and Appendix B, we briefly highlight two genetic-based continuous algorithms relevant to this work. **Differential evolution (DE)** [55] is a common algorithm that also relies on maintaining a population of size N_{pop} by combining existing individuals with mutation and crossover operations. The main difference is that DE draws three individuals (a, b, c) randomly from the population with recombination probability $C \in [0.5 - 1.0]$. DE then subtracts the individual c from b, multiply the difference with the mutation factor $F \in [0, 2]$, then adds the result to a. This process is repeated until convergence. Similar to GA, DE guidance is provided at the beginning of every generation before mixing, where N_{rl} individuals replace the worst individuals from the previous generation. In summary, for DE or RL-guided DE, the hyperparameters to tune are: $\{N_{pop}, F, C, N_{rl}\}$ and their reasonable values are listed in Table 2.

Natural evolution strategies (NES) [56] are inspired from the genetic-based evolution strategies [57], however, NES iteratively updates a continuous search distribution by following the natural gradient to achieve better fitness (without crossover/mutation operations). We use the implementation of [58] for the exponential NES in this work. Exponential NES has multiple improvements over its NES predecessors by parameterizing the positive definite covariance matrix using the exponential map, which helps in avoiding the computation of the inverse Fisher information matrix. We typically start NES search with a random guess and identity covariance matrix. NES has three main strategy parameters: μ (the update of the center of the search distribution), σ (the update of the step size), B (the update of the transformation matrix), which are adapted to improve a population of size N_{pop}. These strategy parameters are controlled by three learning rates: $\eta_\mu, \eta_\sigma, \eta_B$, where in this work we use the default value for $\eta_\mu = 1$, as suggested by the authors [58], and we tune η_σ and η_B. Similar to GA, NES guidance is provided at the beginning of every generation before adapting the strategy parameters, where N_{rl} individuals enter the population to replace the worst individuals from the previous generation. In summary, for NES or RL-guided NES, the hyperparameters to tune are: $\{N_{pop}, \eta_B, \eta_\sigma, N_{rl}\}$ and their reasonable values are listed in Table 2. As the concept of RL guidance in DE/NES is identical to GA, full RL-guided NES/RL-guided DE algorithms are not presented for brevity. Also, we have used our own implementation for DE, NES, RL-guided DE, and RL-guided NES.

Algorithm 2 RL-guided Genetic Algorithm

```
•Set hyperparameters: N_pop, N_rl, CX, MUT, χ
•Load a pre-trained RL model
for GEN i = 1 to N_gen do
    •Run the RL model for few epochs and store candidate solutions
    if i = 1 (First Generation) then
        •Initialize the population with N_pop using RL candidates
    else
        •Feed the population with N_rl using RL candidates
    •Apply crossover to population with probability CX
    •Apply mutation to population with probability MUT
    •For the selected mutated individuals, mutate attributes with probability χ
    •Evaluate the reward/fitness for all population
    •Select the individuals with highest reward/fitness for next generation
```

3.3. RL-guided Simulated Annealing

Simulated Annealing (SA) [59] is inspired from the concept of annealing in physics to reduce defects in crystals through heating followed by controlled cooling. Likewise GA, SA has been an active algorithm in nuclear reactor optimization research, as can be inferred from several studies utilizing SA for nuclear assembly and core optimization [35, 60, 61]. Therefore, SA is selected as another candidate to test the effect of RL guidance on its performance. SA relies on random walk to generate a candidate solution, followed by fitness evaluation of this can-

didate. Next, a neighboring solution is generated, and its fitness is calculated. The old and new fitness evaluations are compared, if improvement is achieved, continue with the new solution, if no improvement is seen, accept the old solution with probability $\alpha = e^{-\Delta E/T}$, where ΔE is the difference between the two fitness evaluations, and T is the annealing temperature. The previous steps are repeated until convergence. The temperature T is annealed between T_{max} and T_{min} over the annealing period of length N_{steps}. In this work, we include three different temperature annealing schedules when tuning SA hyperparameters, which are commonly used in SA. The fast annealing schedule is defined as

$$T_{Fast} = T_{max} \cdot exp\left[\frac{-log(T_{max}/T_{min})k}{N_{steps}}\right], \quad (6)$$

where k is the current annealing step, which builds up from 1 to N_{steps}. The Boltzmann schedule is expressed by

$$T_{Boltzmann} = \frac{T_{max}}{log(k+1)}, \quad (7)$$

while the Cauchy schedule is given by

$$T_{Cauchy} = \frac{T_{max}}{k+1}. \quad (8)$$

The mode of random-walk for SA is similar to GA mutation. All input attributes are subjected to perturbation with a small probability χ, if $rand \sim U[0,1] < \chi$ is satisfied, the corresponding attribute is replaced with an integer uniformly drawn between the lower and upper bounds. The integers represent the fuel choices in F_{set} and P_{set}.

Similar to RL-guided GA, in this work, an additional hyperparameter called $\chi_{rl} \in [0,1]$ is introduced to SA, which represents a probability to replace a random-walk individual with a RL individual. RL-based individuals help SA to avoid falling in local optima and reduce SA randomness by providing quality solutions periodically during annealing. $\chi_{rl} = 0$ refers to a pure random-walk SA, while for $\chi_{rl} = 1$, the SA chain will completely use RL-based samples. Therefore, χ_{rl} should be tuned for optimal performance to avoid complete bias to RL or insufficient guidance from RL. RL-guided SA is described in Algorithm 3. Similar to RL-guided GA, in Algorithm 3, only RL is used to update SA during the RL-guided SA search, while SA does not affect RL search, as this would facilitate the implementation. The two-way data transfer between RL/EA is kept for future work. In summary, for SA or RL-guided SA, the hyperparameters to tune are: {χ, χ_{rl}, T_{max}, *Cooling*}, and their reasonable values are listed in Table 2. T_{min} is fixed to 1. We have used our own implementation for SA and RL-guided SA as described in this section and Algorithm 3.

The mapping of SA and RL-guided SA to the optimization problem is exactly similar to GA. Each individual in the SA chain is a vector of size 32. Each entry represents a fuel choice from F_{set} or P_{set}. Each individual is evaluated based on the rules $r1 - r7$ in section 2 to determine the fitness value and the validity of this individual.

Algorithm 3 RL-guided Simulated Annealing

```
•Set hyperparameters: N_steps, T_max, T_min, χ, χ_rl, Cooling
•Load a pre-trained RL model
•Initialize the chain with a random candidate and set T ← T_max
•Evaluate fitness E_prev for the random candidate
for Steps i = 1 to N_steps do
    if α_1 ~ U[0,1] > χ_rl then
        •Do a random-walk with probability χ
    else
        •Run the RL model for few epochs and store a candidate solution
        •Use the RL candidate
    •Evaluate fitness E for the new candidate
    •Calculate ΔE = E − E_prev
    if ΔE > 0 and exp(−ΔE/T) < α_2 ~ U[0,1] then
        •Reject the candidate and restore the previous state
    else
        •Accept the candidate and set E_prev ← E
    •Anneal T between T_max and T_min (if any) according to Cooling
```

3.4. RL-guided Particle Swarm Optimisation

Particle swarm optimization (PSO) [62] is a popular evolutionary algorithm for continuous optimisation. Each particle in the swarm experiences position update ($x_i^{t+1} = x_i^t + v_i^{t+1}$), where i is the attribute index and v is the velocity value for that attribute. We implement the constriction approach by Clerc and Kennedy [63] for velocity update in this study, which can be expressed as follows

$$v_i^{t+1} = K[v_i^t + c_1 r_1 (pbest_i^t - x_i^t) + c_2 r_2 (gbest^t - x_i^t)], \quad (9)$$

$$K = \frac{2}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|}, \quad (10)$$

$$\phi = c_1 + c_2, \quad \phi > 4, \quad (11)$$

where c_1, c_2 are the cognitive and social speed constants, respectively, r_1, r_2 are independent uniform random numbers between [0,1], and *pbest*, *gbest* are the local best position for each particle and the global best position of the swarm, respectively. Lastly, K is the constriction coefficient introduced to balance PSO exploration/exploitation and improve stability. Typically, when $c_1 = c_2 = 2.05$, then $K = 0.73$. Another advantage of using constriction is it exempts us from using velocity clamping; therefore there is no need to specify minimum and maximum velocities, which reduces PSO hyperparameters. The number of particles in the swarm is given by $N_{particles}$.

Similar to RL-guided GA/DE/NES, in this work, additional hyperparameter called $N_{rl} \in [1, N_{particles}]$ is introduced to improve PSO, which represents the number of individuals introduced into the swarm from an external source, in this case, a pre-trained RL/PPO model. N_{rl} is a fraction of $N_{particles}$ introduced into the population before applying velocity update to all particles, and it can

be tuned for desirable performance. Unlike RL-guided GA, we noticed that starting the swarm with random particles has better performance for RL-guided PSO than when starting with RL candidates. The RL-guided PSO algorithm is described in Algorithm 4. Notice that only one new hyperparameter (N_{rl}) is introduced (i.e. hyperparameter tuning is not an issue for RL-guided PSO). In addition, in Algorithm 4, only RL is used to update PSO during RL-guided PSO search, while PSO does not affect RL search. In summary, for PSO or RL-guided PSO, the hyperparameters to tune are: $\{N_{particles}, c_1, c_2, N_{rl}\}$, and their reasonable values are listed in Table 2. Lastly, DEAP-1.3.0 [54], which is an evolutionary computation framework, is used to build PSO and our RL-guided PSO algorithms. Significant changes have to be performed to couple RL with PSO under the DEAP framework.

As PSO in this work is restricted to the continuous problems, described in Appendices A - B, the mapping of PSO and RL-guided PSO to the optimization problem depends on the range of the design variables. For the speed reducer, each particle position in the swarm is a vector of size 7 with ranges given in Eq.(24), while for the welded beam, each particle position in the swarm is a vector of size 4 with ranges given in Eq.(33). The particle velocity/speed vector has same size as the position vector, and it is initialized randomly between the prescribed ranges, afterward, the constriction coefficient controls velocity updates.

Algorithm 4 RL-guided Particle Swarm Optimisation

1: •Set hyperparameters: $N_{particles}$, N_{rl}, c_1, c_2
2: •Load a pre-trained RL model
3: **for** GEN $i = 1$ to N_{gen} **do**
4: •Run the RL model for few epochs and store candidate solutions
5: •Feed the swarm with N_{rl} by replacing the worst N_{rl} particles with RL candidates
6: •Update velocity of swarm particles ($N_{particles} + N_{rl}$) with constriction coefficient Eq.(9)-(11).
7: •Generate a new swarm by updating all particle positions.
8: •Evaluate the reward/fitness of the swarm

4. Results

In this section, we describe the results of the three engineering problems analyzed in this work. As the nuclear problem is more complex and of the authors' interest, more results and discussions in sections 4.2 and section 5 are presented for this problem. For the speed reducer and welded beam problems, brief results and discussion are presented to demonstrate the RL-guided concept in PSO/DE/NES, as these problems are well-known optimisation benchmarks.

It is worth mentioning that grid search is used in this work to tune the hyperparameters of all the investigated algorithms (RL-guided and standalone algorithms). Grid search is advantageous when the analyst is aware of a certain range of the hyperparameters based on experience and literature suggestions (see Table 2). In addition to the easier implementation and parallelization of grid search, we noticed that this tuning approach is sufficient to achieve satisfactory performance by all algorithms.

4.1. Benchmarking Results

The objective of this benchmarking section, detailed in Appendices A - B, is to demonstrate the RL-guided methodology in low-dimensional engineering applications with many constraints. The main goal of introducing these benchmarks is to demonstrate the concept on how RL solutions can improve the algorithm search in continuous optimisation, which are for this case the PSO, DE, and NES algorithms. Therefore, this section is by no mean claiming improvement of the state-of-the-art of these benchmark optimal solutions or their state-of-the-art algorithms, instead, is better perceived as a proof-of-concept effort. Also, the tests and conclusions are subjected to the PSO/NES/DE forms described in section 3.

The results in Appendix A are for the speed reducer problem, while Appendix B results are for the welded beam design. Both cases clearly demonstrate that RL-guided algorithms can improve the search capabilities of standalone algorithms under these simplified problems, with results seem to be in good agreement with the literature reported findings. In both problems, standalone PSO with constriction is able to escape all constraints with the predefined tolerance, however, PSO hangs in a local optima early on without converging to a competitive solution. On the other hand, RL-guided PSO with constriction is able to converge to a competitive solution for these problems after introducing RL solutions in the swarm that learn fraction of the rules/constraints. Standalone DE shows competitive performance in both problems, therefore, RL-guided DE shows marginal improvement in the performance. For NES, RL guidance successfully helped NES to achieve better performance in both problems.

For the speed reducer problem, RL-guided PSO best solution is within 0.07% from [64] and better than [65] by about 0.006%, RL-guided DE is better than [65] by 0.02% and within 0.05% from [64], while RL-guided NES is within 0.1% and 0.18% from [65] and [64], respectively. For the welded beam, RL-guided PSO best solution is within 0.1% from [66] and better than [67] by about 28%, RL-guided DE is better than [66] and [67] by 1% and 29%, respectively, while RL-guided NES is better than [66] and [67] by 1.3% and 29.1%, respectively. In summary, in terms of the performance for the welded beam problem, *RL-guided NES with objective value 1.72490 is the best algorithmic performance over all our proposed algorithms*, while for the speed reducer problem, *RL-guided DE achieves the best performance with objective value 2996.33574*.

4.2. Nuclear Assembly Results

The results in this section are presented in two subsections. In section 4.2.1, training results of RL/PPO to learn some of the rules presented in section 2 are described, which yield a pre-trained RL model. In section 4.2.2, we present the RL-guided EA results (mainly GA and SA) in form of two main cases, defined in an appropriate place.

4.2.1. RL Results of Learning Rules 1-4

After applying grid search, the following hyperparameters yielded an optimal performance for PPO when running $N = 20$ cores: $\{T = 2000$ timesteps, $\lambda = 1$, $\omega_{clip} = 0.2$, $\omega_{vf} = 0.5$, $\omega_s = 0$, $B = 4\}$, while other hyperparameters are fixed to $\{\gamma = 0.99$, $lr = 2.5 \times 10^{-4}$, $opt_{epochs} = 15\}$. The results are plotted in Figure 2, where (a) shows the convergence of the total reward based on the sum of four selected rules $r1 - r4$, and (b) shows the learning curve of each individual rule ($V_1 - V_4$), forming the total reward. There is one main reason for selecting $r1 - r4$ for RL, which is problem-specific. The rules $r1 - r4$ are initially displaced since they are considered as constraints on the input space. Therefore, learning them does not require computer simulation via CASMO4, which leads to a significant speedup in RL training. However, the selection is always up to the analyst depending on the problem of interest and computational power availability.

Clearly in Figure 2, the RL agent is able to learn all 4 rules as can be seen from (a) the convergence of total reward to almost zero and (b) the decrease of each rule violation to values close to zero, which both imply very limited mistakes by the agent. Total reward is basically $-\sum_{i=1}^{4} V_i$, where negative sign is used to convert violation to reward, as RL logic is built on maximizing rewards. Rule 1 seems to be the easiest to learn, followed by rule 3, while both rules 2 and 4 require additional time to master them. After about 40 epochs of training, all rules converged, where each epoch involves 115,200 time steps of training. The shaded error bars represent 1-σ standard deviation of the quantity in each epoch, which seem to be very small for our case, implying a stable learning by the RL agent. In Figure 2(b), the rule violation does not reach to zero exactly. Each epoch is average of large number of time steps (115,200), and the policy may make wrong actions occasionally. The wrong action is reflected in violating one or more rules, and these violated patterns are automatically discarded by the fitness function and are not passed to the stochastic algorithms. We will explore the impact of learning these rules using RL on the overall optimization performance in the upcoming section 4.2.2.

4.2.2. RL-guided EA Results

Two cases are explored in this section to investigate the performance of RL-guided algorithms, which can be defined as follows:

- Case 1: this case features meeting all rules $r1 - r7$, however, with relaxed thresholds for $r5 - r7$: $k_\infty \leq 1.12000$, $PPF \leq 1.55$, $CL \geq 42.0$.

  After applying grid search, the following hyperparameters yielded an optimal performance for SA $\{\chi = 0.01$, $T_{max} = 10000$, $Cooling = Cauchy\}$. These hyperparameters are preserved in RL-guided SA to isolate their effect, while $\chi_{rl} = 0.075$ is used for the RL-guided SA special parameter. Similarly for GA, optimal hyperparameters are $\{N_{pop} = 50$, $MUT = 0.25$, $CX = 0.8$, $\chi = 0.025\}$. These hyperparameters are preserved in RL-guided GA to isolate their effect, while $N_{rl} = 6$ is used for the RL-guided GA special parameter, meaning that 6 out of $N_{pop} = 50$ individuals are supplied by the RL agent. The RL hyperparameters are similar to what was described before in the previous section 4.2.1.

- Case 2: this case features meeting all rules $r1 - r7$, however, with more confined and realistic thresholds for $r5 - r7$: $k_\infty \leq 1.11000$, $PPF \leq 1.4$, $CL \geq 43.0$. Although of the small numerical changes on the thresholds, these changes are indeed significant as will be reflected later on the performance of all algorithms. Solving Case 2 is the end-goal for the nuclear assembly optimization as these thresholds represent desirable conditions, as nuclear industry has relied on expert input due to limited ability of standalone RL/EA techniques.

  For hyperparameters, grid search revealed that using $\chi_{rl} = 0.125$ in RL-guided SA and $N_{rl} = 7$ in RL-guided GA can yield better performance compared to $\chi_{rl} = 0.075$ and $N_{rl} = 6$ in Case 1. All other hyperparameters remain as defined for Case 1 above.

For Case 1, all algorithms are executed for 100 epochs, each epoch involves 100 fitness calculations (CASMO4 simulations), leaving us with a total of 10,000 fitness evaluations for all algorithms. Lastly, it is worth highlighting that we gave standalone RL additional boost to allow PPO to shape useful gradients. PPO policy is very stochastic by the beginning of the training process, thus most of the initial samples are of low quality and random nature, making the comparison of standalone RL meaningless. Alternatively, a warmup period of about 20 epochs is given to allow PPO to find some candidate patterns and be prepared, once that occurs, the effective 10,000 iterations and candidate pattern counter start real counting from zero. We will verify later to see if this boosting causes any bias toward RL in the performance.

The results of Case 1 are presented in Figure 3 for RL-guided GA/SA. Notice that we present three different forms of RL-guided GA/SA to capture the effect of gradual guidance. $\text{RL}^{r1 \rightarrow r2}$ represents a RL agent learning the rules $r1 - r2$, $\text{RL}^{r1 \rightarrow r3}$ represents a RL agent learning the

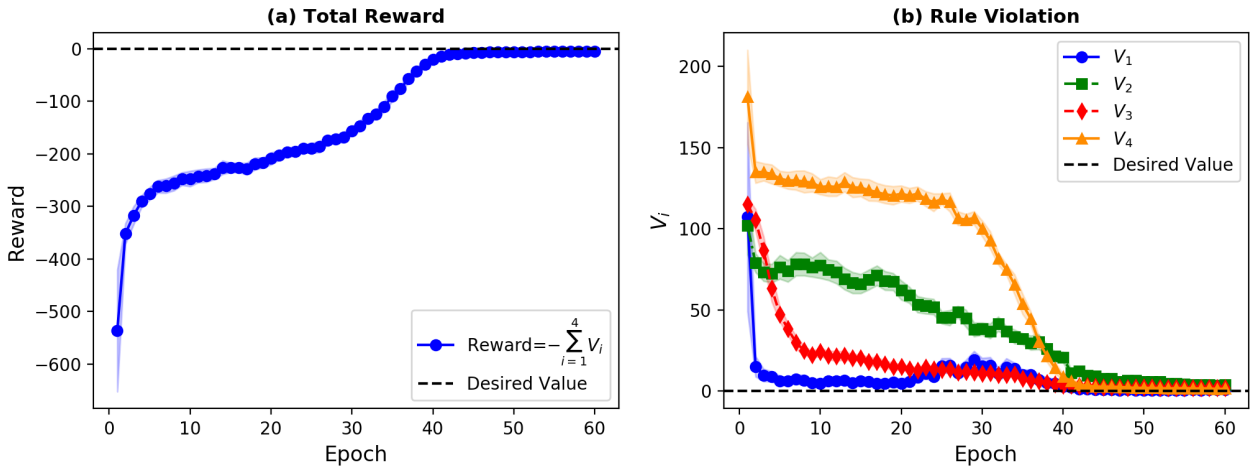

Fig. 2. RL/PPO convergence plots for (a) total reward ($-\sum_{i=1}^{4} V_i$) and (b) rule violation. Each epoch consists of 115,200 time steps of training

rules $r1-r3$, while RL$^{r1\rightarrow r4}$ represents a RL agent learning all rules from $r1-r4$. It is worth mentioning that Rules 1-4 aim to focus the search in areas where expert knowledge has demonstrated its merit from physics perspective, therefore, they do not directly affect the calculation of the remaining Rules 5-7, and vise versa.

The comparison of RL-guided GA in Figure 3(a) against standalone GA and RL shows how RL-guided GA leveraged its performance after obtaining additional guidance from RL. RL$^{r1\rightarrow r4}$-guided GA found 988 solutions, followed by RL$^{r1\rightarrow r3}$-guided GA (585 solutions), and then RL$^{r1\rightarrow r2}$-guided GA (506 solutions). All RL-guided GA forms outperform standalone GA and RL, which found 402 and 36 solutions, respectively, see Figure 3(a). Similar results for RL-guided SA are presented in Figure 3(b). Unlike standalone GA (402 solutions), standalone SA found only 8 candidate patterns, which are indeed less than standalone RL (36 solutions). Again, RL-guided SA excels over standalone algorithms through finding many more candidate patterns as RL$^{r1\rightarrow r4}$-guided SA found 723 solutions, followed by RL$^{r1\rightarrow r3}$-guided SA (435 solutions), and then RL$^{r1\rightarrow r2}$-guided SA (381 solutions), see Figure 3(b). In both cases for RL-guided GA/SA, we can see that the warmup boost given to standalone RL did not change the conclusion, as standalone RL (36 solutions) is still far from competing with RL-guided GA/SA (988 and 723 solutions).

To determine the computational efficiency of RL-guided EA, we present the computational time savings achieved by RL-guided GA against standalone GA and RL in Table 3, while the results for RL-guided SA against standalone SA and RL are presented in Table 4. The computational saving is defined as "*how many fitness evaluations can be saved by a RL-guided EA algorithm to find similar number of candidate patterns as a standalone algorithm*". Evaluations are converted to time by assuming each fitness evaluation requires about 1.5 minutes in average for CASMO4 execution to complete. For example, if standalone SA found 8 candidate patterns over 10,000 fitness evaluations, RL$^{r1\rightarrow r4}$-guided SA needed only 332 fitness evaluations to find 8 candidate patterns, which correspond to a saving of $\frac{(10000-332)*1.5}{60}$ = 242 hours (or 10 days) of serial computing time. Similarly, Table 3 shows that RL$^{r1\rightarrow r4}$-guided GA can achieve about 120 hrs (5 days) of computational savings compared to standalone GA, and up to 230 hours (9.5 days) compared to standalone RL.

Table 3. Case 1 - computational time savings in hours achieved by RL-guided GA compared to standalone GA and RL

Case	Versus GA*	Versus RL**
RL$^{r1\rightarrow r2}$-guided GA	44.9 hrs	204.0 hrs
RL$^{r1\rightarrow r3}$-guided GA	76.7 hrs	205.8 hrs
RL$^{r1\rightarrow r4}$-guided GA	119.8 hrs	230.1 hrs

*Savings are calculated based on the time needed by RL-guided GA
to find 402 candidate patterns (total found by standalone GA)
**Savings are calculated based on the time needed by RL-guided GA
to find 36 candidate patterns (total found by standalone RL)

Given the difficulty of Case 2, we give all algorithms twice computational costs compared to Case 1 (i.e. 20,000 iterations or 200 epochs of search). After moving to Case 2, Figure 4 shows that space confinement causes standalone algorithms to significantly suffer to find a single candidate solution, as standalone SA, GA, and RL could not find any candidate pattern over 200 epochs of search. However, RL-guided GA/SA maintain their outstanding performance, where RL-guided GA found about 1000 candidate patterns, while RL-guided SA found about 30 candidate patterns, after 200 epochs

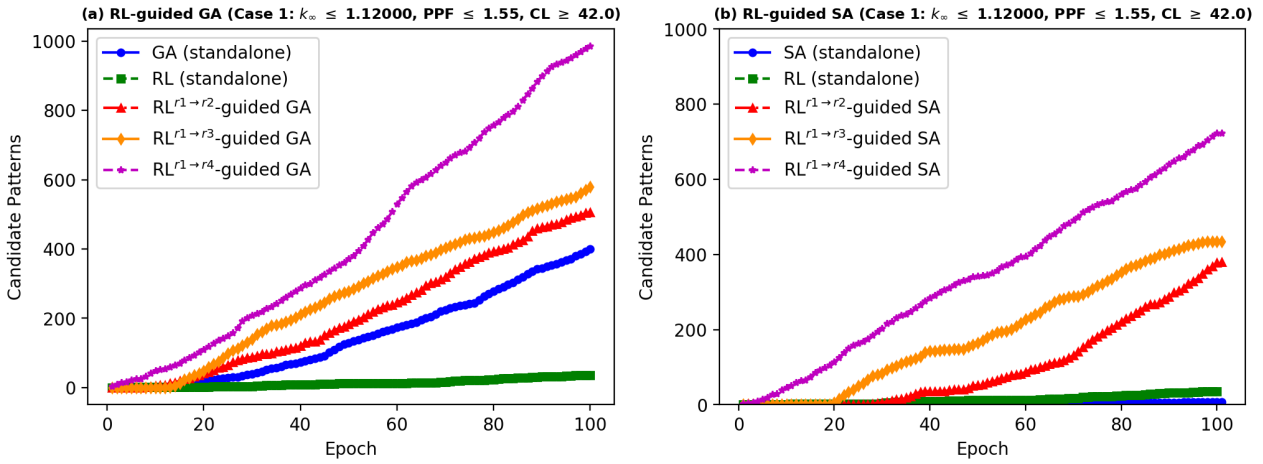

Fig. 3. Case 1 - number of cumulative candidate patterns found by different algorithms for (a) RL-guided GA and (b) RL-guided SA ($k_\infty \leq 1.12000$, $PPF \leq 1.55$, $CL \geq 42.0$). 1 epoch = 100 fitness evaluations for all algorithms

Table 4. Case 1 - computational time savings in hours achieved by RL-guided SA compared to standalone SA and RL

Case	Versus SA*	Versus RL**
RL$^{r1\rightarrow r2}$-guided SA	173.9 hrs	156.4 hrs
RL$^{r1\rightarrow r3}$-guided SA	200.9 hrs	192.8 hrs
RL$^{r1\rightarrow r4}$-guided SA	241.7 hrs	229.8 hrs

*Savings are calculated based on the time needed by RL-guided SA
to find 8 candidate patterns (total found by standalone SA)
**Savings are calculated based on the time needed by RL-guided SA
to find 36 candidate patterns (total found by standalone RL)

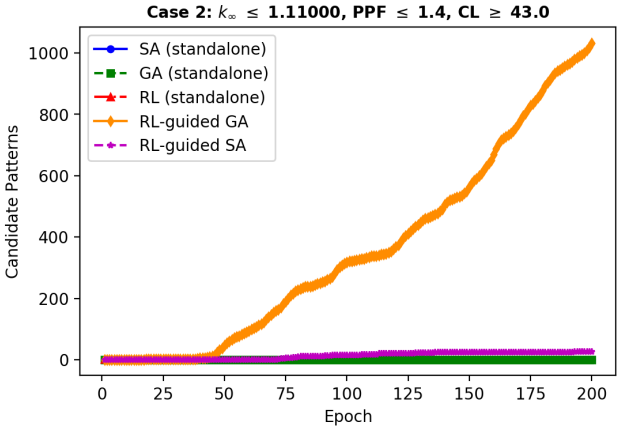

Fig. 4. Case 2 - number of cumulative candidate patterns found by different algorithms ($k_\infty \leq 1.11000$, $PPF \leq 1.4$, $CL \geq 43.0$). RL-guided GA/SA use the RL$^{r1\rightarrow r4}$ agent, 1 epoch = 100 fitness evaluations for all algorithms

of search. Notice that a simplified notation is used for convenience for RL-guided GA/SA in Figure 4, as the RL$^{r1\rightarrow r4}$ agent is used, since it proved to be the best in Figure 3. Due to the inability of standalone algorithms to find a single solution, definition of computational savings does not exist for Case 2 as no reference point can be obtained to determine the speedup as in Case 1 (Tables 3-4). Alternatively, based upon Figure 4 and assuming fuel designers or analysts have access to 20 processors (like we have used in this work), then RL-guided GA can provide 100 candidate solutions in about 7.5 hrs, and RL-guided SA can provide 10 candidate solutions in about 10 hrs. These numbers are feasible times in nuclear design and practice.

5. Discussion

In the previous section along with appendices A - B, we noticed that using RL in form of policy gradient to inform stochastic algorithms can improve their search capabilities by introducing the neural gradient-based solutions into the stochastic population with one-way coupling (i.e. no data transfer back to RL). The RL solutions feature matching fraction of the problem rules/constraints to focus the search into regions close to the feasible regions (e.g. expert knowledge, no constraint violation). We observed significant speedup and improvement in PSO continuous optimisation after implementing RL-guided PSO, while we observed a small improvement in NSE/DE after using RL guidance. Nevertheless, the improvements by RL-guided NES and RL-guided DE are still noticeable and within the range of significance for the presented benchmark engineering problems based on existing publications. In general, more practical value is observed for the nuclear assembly problem, which is much more expensive and high-dimensional, where RL gradient solu-

tions demonstrate more merit after applying RL-guided GA and RL-guided SA.

During the tuning process of RL-guided GA/SA/PSO/NES/DE hyperparameters, we observed that lower values of χ_{rl} in RL-guided SA and N_{rl} in RL-guided GA/PSO/NES/DE are preferred for better performance. Lower values are preferred to avoid restricting the inherent search capabilities of evolutionary/stochastic algorithms, since otherwise the search will be dictated mainly by RL, which is inadequate as standalone as we observed before.

By the end of section 1, we mentioned that the success of our approach on the nuclear assembly optimisation problem is motivated by at least one of the following reasons: (1) effective exploration of heavily-constrained search spaces, (2) computational efficiency of RL-guided EA, (3) handling the problem complexity, and (4) the practical value of optimization.

In Figures 3-4, RL-guided GA/SA clearly show outstanding ability in exploring the search space despite the hard constraints, matching all 7 rules and discovering many more candidate patterns than their standalone counterparts. In particular, for Case 1, RL$^{r1 \rightarrow r4}$-guided GA found more candidate patterns by factors of 2.5 and 28 compared to standalone GA and RL, respectively. Also, RL$^{r1 \rightarrow r4}$-guided SA found more candidate patterns by factors of 90 and 20 compared to standalone SA and RL, respectively. For Case 2 when the constraints become more confined, standalone algorithms did not compete, while RL-guided GA/SA maintained their excellent exploration by finding hundreds of candidate patterns.

In Figure 3, it was shown how RL guidance can help making EA more efficient through ensuring GA/SA search is conducted in feasible regions. For example, in Figure 3, we can see that standalone GA, SA, and RL needed about 20 epochs or more to start discovering patterns, compared to RL-guided algorithms, which can start after 2 or 3 epochs. In terms of computational complexity of the algorithms, obviously, RL-guided EA is more expensive than standalone EA, due to the fact of having neural networks trained in PPO to learn the rules before guiding EA algorithms. However, this increase in RL-guided algorithm cost is compensated by the incredible time savings in the fitness evaluation as presented in Tables 3-4, which can reach as high as 10 days of serial computing time. In addition, for Case 2, we noticed that standalone EA algorithms could not find a candidate solution compared to RL-guided EA, where in this case, the optimisation performance takes priority over algorithm cost and complexity.

As discussed before in section 2, the nuclear optimization problem is described as multi-objective, combinatorial, constrained, high-dimensional, and expensive (i.e. requires computer simulation). All previous challenges, and in particular the last two, result in having nuclear utilities to rely on domain and expert knowledge to optimise the nuclear assembly. The problem is also characterized by many local optima, where it is likely to find hundreds of patterns having exact same fitness value. These local optima solutions could be neighbours (resembling each other), and can be even far from each other, which add additional difficulties. This issue is problematic for standalone GA and SA, as without guidance, the GA/SA searches are likely to hang in those local optima for a while before moving to a new region. On the other hand, the heavily-constrained nature of the problem was mostly problematic to standalone RL. Despite the ability of RL to learn $r1 - r4$ as in Figure 2, RL started to struggle as additional rules are added, and became even worse once the constraints become more confined in Case 2. It is known that RL/PPO directly learns by gradient descent from the collected experiences, which are likely to be of lower quality when exploring more confined spaces. Therefore, due to the hard constraints, the standalone RL agent was not able to leverage competitive gradients even with the additional training boosts we offered. Previous complexities were handled efficiently by RL-guided algorithms through taking advantage of RL gradients for rule-making of $r1 - r4$. Then, the population-based and random-walk nature of GA/SA handled the remaining rules, resulting in more effective optimization.

On the practical aspects of this work, we present the layout of two selected candidate assembly patterns as found by RL-guided GA/SA in Figure 5. The two patterns respect all rules described before where: ($r1$) number of poison rods is low, ($r2$) number of poison rods belongs to the range [16,18], ($r3$) average enrichment E in both patterns belongs to the desired range [4.25%, 4.36%], ($r4$) poison rods do not neighbour each other, ($r5$) $k_\infty = 1.10541/1.09490 \leq 1.11000$, ($r6$) $PPF = 1.38/1.4 \leq 1.4$, and ($r7$) $CL = 43.55/43.32 \geq 43.0$. As mentioned before, the choices of the bounds and thresholds for the constraints were not arbitrary, but guided by the literature and Exelon corporation (the funding agency). Therefore, we know beforehand that these thresholds can be satisfied and candidate patterns exist in the space. This implies that in the cases of high-dimensional combinatorial search of expensive engineering problems, realistic constraints/thresholds should be carefully determined by the analysts, as blind guess may lead to unphysical search, where candidate patterns do not exist.

For the continuous problems (speed reducer and welded beam) in appendices A - B, we should reiterate that our goal of adding these benchmark problems is to demonstrate that the RL guidance may improve the search capabilities of evolutionary/stochastic algorithms in constrained continuous optimisation, rather than proving that RL-guided EA is better than standalone EA or vise versa. As there are large number of varieties of PSO/DE/NES algorithms, that are effective in solving these benchmark problems very accurately, our conclusions on these benchmarks are subjective to the PSO/DE/NES forms and

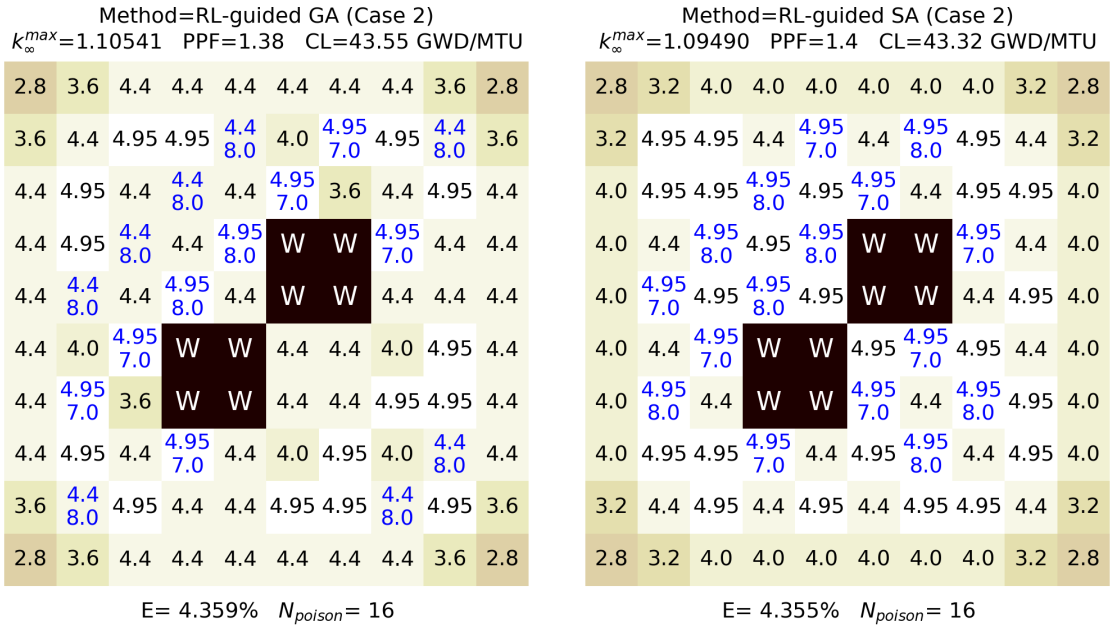

Fig. 5. Case 2 - samples of the best patterns found by RL-guided GA (left) and RL-guided SA (right). Boxes with black numbers refer to fuel rods in F_{set}. Boxes with blue numbers refer to poison rods in P_{set}, where the top value is the UO_2 enrichment and the bottom value is the poison enrichment

the hyperparameter ranges we have used here. Despite the improvement observed for the standalone PSO and NES, the RL-guided concept is justifiable and recommended for high-dimensional and expensive problems like the nuclear assembly case, rather than low-dimensional benchmark problems. Obviously, the computational gain from RL guidance is negligible for the benchmark problems, as their physics is very simple to model and their dimensionality is low.

In this work, RL guidance for GA was in forming the initial population as well as blending N_{rl} individuals in the population every generation before crossover/mutation. The latter RL-guided change was also applicable to DE and NES. For SA, the RL guidance was through providing a single solution periodically with rate controlled by χ_{rl}. For PSO, RL guidance was in replacing the worst N_{rl} particles in the swarm by RL individuals every generation. However, these are not the only approaches that we can try. For example, providing RL solutions after GA crossover/mutation (after offspring), or providing RL solutions before and after the offspring could be reasonable alternatives. These are also applicable to PSO before and after velocity updates. Instead of relying on a single RL solution for SA, multiple chains can be initialized in parallel to test multiple RL solutions before selecting the best for SA. In addition, for RL-guided SA, the RL solution was completely replacing the SA solution once the RL condition is satisfied. Alternatively, the analyst could mix the RL and SA solutions in that stage, which also mimics the crossover effect in RL-guided GA. For PSO, the RL-guidance concept can be tested and improved on more high-dimensional, expensive, and constrained continuous optimisation problems, where different PSO forms (e.g. discrete, velocity clamping, inertia weighting) can be explored [68, 69]. Furthermore, making the values of χ_{rl} and N_{rl} adaptive rather than fixed during search is another interesting idea. When EA optimization is losing track, more RL guidance is used to bring the search back to relevant areas, while when EA optimization is doing well, RL guidance can be minimized. Similar extensions can be performed for DE and NES. Indeed, since we already have observed outstanding improvement in RL-guided EA with our current contributions, we left the previous advanced ideas for the reader to explore, opening multiple doors for future work.

In this work, the selection of evolutionary/stochastic algorithm candidates was dictated mainly by the popularity of GA, SA, PSO, NES, and DE in the optimization area in general, and for the analyzed applications in specific. Nevertheless, the methodology can be extended to investigate the effect of RL guidance on other evolutionary or stochastic algorithms such as tabu search or other evolution strategies, in which RL can provide guidance to these algorithms in continuous optimization with confined search spaces. Similarly, the RL component can be replaced with other RL techniques such as deep Q learning or asynchronous actor-critic strategy (A3C), which may perform better than PPO in other applications. Therefore,

the flexibility of the approach and the rooms for additional investigations are widely open.

Last but not least, the optimised patterns in Figure 5 by our proposed RL-guided algorithms can lead to an improvement in nuclear fuel efficiency through extended cycle operation, cost reduction, and nuclear safety assurance. This is in addition to provide an intelligent decision support system by saving a lot of human and computational efforts to perform manual optimization through expert knowledge. The implication of this study to non-nuclear applications features using RL as a tool to train an agent that learns how to match certain rules and constraints from expert knowledge or design constraints for a high-dimensional and computationally expensive problem. The trained agent is then used to guide EA for final optimization by ensuring the search is taking place in feasible regions.

6. Conclusions

In constrained optimization, analysts are interested in narrow regions of the search space that match all imposed rules/constraints. As constrained optimization is very common in engineering, we demonstrated a rule-based reinforcement learning (RL) methodology to effectively search in heavily-constrained spaces. A real-world engineering problem, featuring nuclear fuel assembly optimization in nuclear power plants, is used to demonstrate the concept on discrete optimisation. The problem is described as multi-objective, combinatorial, constrained, high-dimensional, and expensive (i.e. requires computer simulation), with a total of seven rules to meet. Two engineering benchmark problems, the speed reducer and welded beam, are used to demonstrate the concept on continuous optimisation. RL agents trained by proximal policy optimization are used to learn some of the problem rules/constraints. Next, we developed RL-guided GA, RL-guided SA algorithms for combinatorial optimisation, and RL-guided PSO, RL-guided DE, and RL-guided NES for continuous optimisation. The pre-trained RL agents are used to inject experiences to guide standalone algorithms during optimization. RL experiences were able to maintain GA/SA/PSO/DE/NES search in the feasible regions, allowing them to effectively search in those regions instead of relying on random-walk, which is inefficient.

Despite small guidance from RL and under same hyperparameter and fitness settings, RL-guided GA/SA algorithms demonstrate excellent performance compared to their standalone counterparts (SA, GA, RL), outperforming them by a wide margin for the nuclear assembly problem. RL-guided GA/SA found many more candidate patterns than standalone algorithms with an improvement factor between 2.5 and up to 90, implying much better exploration capabilities. Moreover, RL-guided GA/SA featured higher computational efficiency than standalone algorithms, as RL-guided GA/SA achieved computational time savings up to 230 hours and 242 hours of serial computing time, respectively. Under more confined spaces ($k_\infty \leq 1.11000, PPF \leq 1.4, CL \geq 43.0$), which are more realistic, but difficult to solve, standalone algorithms struggle to find even a single candidate solution, while RL-guided GA and RL-guided SA excel, finding respectively about 1000 and 30 candidate patterns.

The tests on continuous optimisation problems through RL-guided PSO/NES/DE demonstrate excellent performance as RL-guided DE and RL-guided NES found competitive solutions of the engineering benchmark problems. Compared to the literature, RL-guided DE achieved the best result for the speed reducer problem with 11 constraints, while RL-guided NES achieved the best result for the welded beam problem with 7 constraints. Nevertheless, these engineering applications are of low-dimensional and simple physics nature, therefore, the computational gain from RL guidance is not significant. Also, the conclusions are subjected to the PSO/NES/DE forms and the hyperparameter ranges we used in this work.

In summary, the results of this work made another step forward toward thinking of additional ways on hybridizing deep RL with evolutionary and/or stochastic algorithms. As this work focused on RL sending information to EA, but no information is sent back to RL, our future work will focus on developing an efficient and novel algorithm that involves two-way coupling between RL and EA with parallel capabilities, to allow scaling to expensive large-scale optimization problems. The coupled approach will take advantage of the gradient-based and gradient-free properties of RL and EA, respectively, to achieve better optimization. Additionally, the RL policy gradient approach used in this work can be compared in future to other evolutionary and Q-learning algorithms in terms of their performance on guiding and improving the search performance in constrained/unconstrained stochastic optimisation.

Acknowledgment

This work is sponsored by Exelon Corporation, a leading U.S. electric utility, under the award (40008739). The authors would like to thank Dr. Joshua Joseph and Prof. Nicholas Roy from MIT Quest for Intelligence for their comments on the presentation of this work.

Conflict of Interests

The authors have no conflict of interests to declare about this work.

CRediT Author Statement

M. I. Radaideh: Conceptualization, Methodology, Software, Validation, Investigation, Data curation, Visualisa-

tion, Formal analysis, Writing - Original Draft.
K. Shirvan: Conceptualization, Methodology, Investigation, Funding Acquisition, Writing - Original Draft.

Appendix A Speed reducer design benchmark

The speed reducer problem was originally introduced by Golinski [70], which aims on the minimization of the weight of a speed reducer, formulated as

$$\begin{aligned}\min_{\vec{x}} W(\vec{x}) = {} & 0.7854x_1x_2^2(3.3333x_3^2 + 14.9334x_3 - 43.0934) \\ & - 1.508x_1(x_6^2 + x_7^2) + 7.4777(x_6^3 + x_7^3) \\ & + 0.7854(x_4x_6^2 + x_5x_7^2)\end{aligned} \tag{12}$$

where the seven design variables are: the face width (x_1), module of teeth (x_2), number of pinion teeth (x_3), length of first shaft between the bearings (x_4), length of the second shaft between the bearings (x_5), diameter of the first shaft (x_6), and the diameter of the second shaft (x_7). The problem has eleven rules/constraints related to bending stress of the gear teeth, surface stress, transverse deflections of the shafts, and stresses in the shafts, formulated as follows:

$$r1(\vec{x}) = \frac{27}{x_1x_2^2x_3} - 1 \leq 0 \tag{13}$$

$$r2(\vec{x}) = \frac{397.5}{x_1x_2^2x_3^2} - 1 \leq 0 \tag{14}$$

$$r3(\vec{x}) = \frac{1.93x_4^3}{x_2x_3x_6^4} - 1 \leq 0 \tag{15}$$

$$r4(\vec{x}) = \frac{1.93x_5^3}{x_2x_3x_7^4} - 1 \leq 0 \tag{16}$$

$$r5(\vec{x}) = \frac{1}{110x_6^3}\sqrt{\left(\frac{745x_4}{x_2x_3}\right)^2 + 16.9\times10^6} - 1 \leq 0 \tag{17}$$

$$r6(\vec{x}) = \frac{1}{85x_7^3}\sqrt{\left(\frac{745x_5}{x_2x_3}\right)^2 + 157.5\times10^6} - 1 \leq 0 \tag{18}$$

$$r7(\vec{x}) = \frac{x_2x_3}{40} - 1 \leq 0 \tag{19}$$

$$r8(\vec{x}) = \frac{x_1}{12x_2} - 1 \leq 0 \tag{20}$$

$$r9(\vec{x}) = \frac{5x_2}{x_1} - 1 \leq 0 \tag{21}$$

$$r10(\vec{x}) = \frac{1.5x_6 + 1.9}{x_4} - 1 \leq 0 \tag{22}$$

$$r11(\vec{x}) = \frac{1.1x_7 + 1.9}{x_5} - 1 \leq 0 \tag{23}$$

while the ranges of the design variables are:

$$\begin{aligned}&2.6 \leq x_1 \leq 3.6, \quad 0.7 \leq x_2 \leq 0.8, \quad 17 \leq x_3 \leq 28, \\ &7.3 \leq x_4 \leq 8.3, \quad 7.8 \leq x_5 \leq 8.3, \quad 2.9 \leq x_6 \leq 3.9, \\ &5.0 \leq x_7 \leq 5.5.\end{aligned} \tag{24}$$

As this work focuses on demonstrating the RL-guided concept on a continuous optimisation problem, the design variable x_3 is of discrete nature. Therefore, we rounded x_3 to the nearest integer during optimisation. Also, the penalty factor approach is used to handle the constraints above, where the fitness function is penalized once any constraint is violated.

For this problem, the hyperparameters of standalone and RL-guided algorithms are tuned with grid search, where +100 hyperparameter configurations are tested. The optimal hyperparameters of PSO are as follows: $N_{particles} = 62$, $c_1 = 2.05$, $c_2 = 2.05$, while for RL-guided PSO, the optimal hyperparameters are $N_{particles} = 90$, $c_1 = 2.1$, $c_2 = 2.1$, $N_{rl} = 9$. The hyperparameters of DE/RL-guided DE are as follows: $F = 0.4$, $C = 0.6$, $N_{rl} = 3$, while for NES/RL-guided NES: $\eta_\sigma = 0.15$, $\eta_B = 0.07$, $N_{pop} = 50$, $N_{rl} = 6$. The tolerance to escape all constraints is set to 10^{-5}, and we round all results in Table 5 and Figure 6(a) to the nearest fifth significant digit. Figure 6(a) shows the convergence of the speed reducer weight function for all algorithms, while a desired optimal limit is plotted based on what the literature have reported for this problem. RL learns the rules $r1 - r8$ in Eqs.(13)-(20) with PPO, then the samples that satisfy these rules are supplied to PSO/DE/NES at every generation with rate N_{rl}. Therefore, we refer to RL-guided PSO/DE/NES as RL$^{r1\rightarrow r8}$-guided PSO/DE/NES.

Notice that standalone RL/PPO is excluded from the results and the comparison, as it is not competitive to the other evolutionary algorithms due to the sensitive tolerance we apply and the allowed number of function evaluations. Standalone RL is able to find solutions that match some of the rules to inform PSO/NES/DE, but could not match all rules with the predetermined tolerance to optimise the weight function in Eq.(12). PPO tries to learn a policy that can take proper actions, which alone requires a lot of function evaluations due to the excessive constraints of the speed reducer problem. Table 5 presents the optimal design variables and the constraint values obtained using all standalone and RL$^{r1\rightarrow r8}$-guided algorithms along with two reference cases for comparison with the literature. The best weight value of the speed reducer obtained using RL$^{r1\rightarrow r8}$-guided PSO/NES/DE is

Table 5. Optimum results for minimization of the speed reducer weight

Item	PSO	RL-PSO[a]	DE	RL-DE[a]	NES	RL-NES[a]	ABC[b] [65]	SAC[c] [64]
$W(\vec{x})$	3198.96072	2996.85284	2996.33796	2996.33574	3012.24244	3000.15554	2997.05841	2994.74420
x_1	3.50000	3.50000	3.49997	3.49997	3.51290	3.50511	3.50000	3.50001
x_2	0.70000	0.70000	0.70000	0.70000	0.70000	0.70000	0.70000	0.70000
x_3	17	17	17	17	17	17	17	17
x_4	8.06702	7.30000	7.30000	7.30000	8.03800	7.30000	7.30000	7.32760
x_5	8.08681	7.80000	7.80000	7.80002	7.83952	7.82925	7.80000	7.71532
x_6	3.90000	3.35205	3.35022	3.35022	3.35261	3.35044	3.35022	3.35027
x_7	5.32240	5.28674	5.28669	5.28668	5.29114	5.28841	5.28780	5.28665
$r1$	-0.07392	-0.07392	-0.07391	-0.07391	-0.07732	-0.07527	-0.07392	-0.07392
$r2$	-0.19800	-0.19800	-0.19799	-0.19799	-0.20094	-0.19917	-0.19800	-0.19800
$r3$	-0.63196	-0.50027	-0.49918	-0.49917	-0.33331	-0.49931	-0.49917	-0.49350
$r4$	-0.89312	-0.90148	-0.90147	-0.90147	-0.90030	-0.90049	-0.90156	-0.90464
$r5$	-0.36524	-0.00165	-0.00001	0.00000	-0.00085	-0.00020	0.00000	0.00000
$r6$	-0.01994	-0.00003	0.00000	0.00000	-0.00252	-0.00098	-0.00063	0.00000
$r7$	-0.70250	-0.70250	-0.70250	-0.70250	-0.70250	-0.70250	-0.70250	-0.70250
$r8$	0.00000	0.00000	0.00001	0.00001	-0.00367	-0.00146	0.00000	0.00000
$r9$	-0.58333	-0.58333	-0.58334	-0.58334	-0.58180	-0.58273	-0.58333	-0.58333
$r10$	-0.03930	-0.05095	-0.05132	-0.05133	-0.13798	-0.05128	-0.05133	-0.05489
$r11$	-0.04108	-0.01084	-0.01085	-0.01086	-0.01521	-0.01430	-0.01070	0.00000

[a] RL$^{r1 \rightarrow r8}$-guided PSO/DE/NES
[b] Artificial bee colony
[c] Society And Civilization

close to the reference cases, all constraints are met, and with function evaluations range varies between 5000-25000 depending on the algorithm to achieve good accuracy. For example, RL$^{r1 \rightarrow r8}$-guided PSO best solution is within 0.07% from [64] and better than [65] by about 0.006%. Standalone PSO is able to escape the constraint region, however, it is not able to minimize the weight function to satisfactory value as RL$^{r1 \rightarrow r8}$-guided PSO or the reported literature values [64, 65]. Obviously, standalone PSO hangs in a local optima in the sixth epoch of search compared to RL-guided PSO, which obtains help from RL solutions to diversify the search and converge to a satisfactory weight value. The results again show significant improvement in the same PSO algorithm after some guidance from RL.

For NES, the performance of RL$^{r1 \rightarrow r8}$-guided NES is also better than NES as can be told from Table 5 and Figure 6(a). Under similar computational settings, the optimal (minimal) objective value (i.e. speed reducer weight) achieved by RL$^{r1 \rightarrow r8}$-guided NES is 3000.15 compared to 3012.24 by NES, accompanied by better convergence behaviour. The performance of the standalone DE algorithm is already competitive for the speed reducer problem, as can be inferred from the identical performance with RL$^{r1 \rightarrow r8}$-guided DE, see Table 5 and Figure 6(a). Therefore, a marginal improvement is achieved by RL guidance. In summary, in terms of the performance for the speed reducer problem, RL$^{r1 \rightarrow r8}$-guided DE with objective value 2996.33574 is the best algorithmic performance over all our proposed algorithms in terms of agreement with the literature.

Appendix B Welded beam design benchmark

The welded beam was originally introduced by Deb [67], with an objective to find an optimal set of the dimensions h (x_1), l (x_2), t (x_3), and b (x_4) such that the fabrication cost of the beam is minimized. See Figure 1 of [67] for graphical details of the beam dimensions (h, l, t, b). The cost of the welded beam is formulated as

$$\min_{\vec{x}} C(\vec{x}) = 1.10471 x_1^2 x_2 + 0.04811 x_3 x_4 (14 + x_2), \quad (25)$$

subject to 7 rules/constraints, the first on the shear stress (τ)

$$r1(\vec{x}) = \tau(\vec{x}) - \tau_{max} \leq 0, \quad (26)$$

the second on the bending stress (σ)

$$r2(\vec{x}) = \sigma(\vec{x}) - \sigma_{max} \leq 0, \quad (27)$$

three side constraints

$$r3(\vec{x}) = x_1 - x_4 \leq 0, \quad (28)$$

$$r4(\vec{x}) = 0.10471 x_1^2 + 0.04811 x_3 x_4 (14 + x_2) - 5 \leq 0, \quad (29)$$

$$r5(\vec{x}) = 0.125 - x_1 \leq 0, \quad (30)$$

the sixth on the end deflection of the beam (δ)

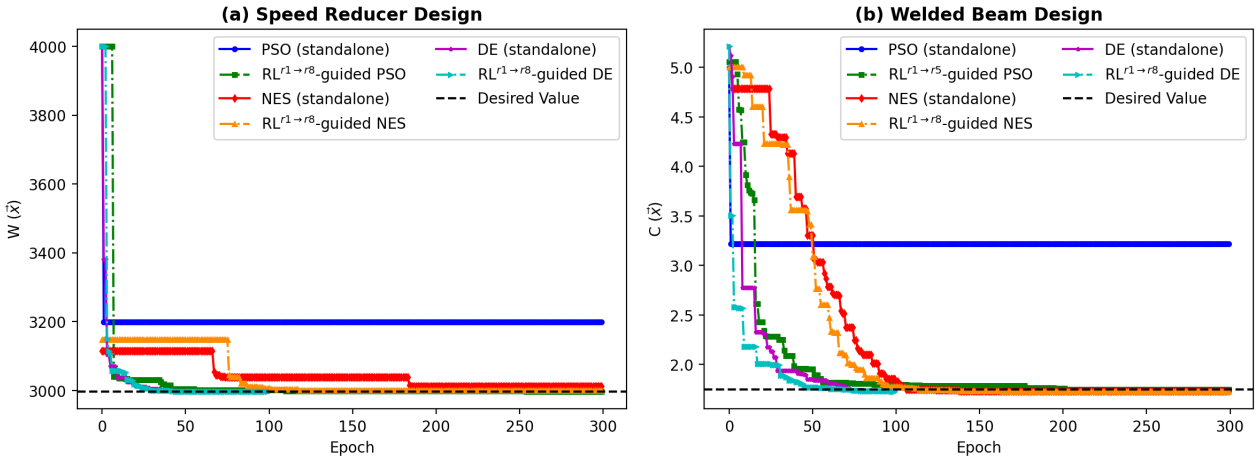

Fig. 6. Convergence of the fitness function with number of epochs for different algorithms for: (a) speed reducer weight and (b) welded beam cost (1 epoch = population size of each algorithm)

$$r6(\vec{x}) = \delta(\vec{x}) - \delta_{max} \leq 0, \tag{31}$$

and the last on the buckling load on the bar (P_c)

$$r7(\vec{x}) = P - P_c(\vec{x}) \leq 0, \tag{32}$$

while the ranges of the design variables are:

$$\begin{aligned} 0.1 \leq x_1 \leq 2, \quad 0.1 \leq x_2 \leq 10, \\ 0.1 \leq x_3 \leq 10, \quad 0.1 \leq x_4 \leq 2. \end{aligned} \tag{33}$$

The derived variables and their related constants are expressed as follows [66]:

$$\tau(\vec{x}) = \sqrt{(\tau')^2 + 2\tau'\tau'' \frac{x_2}{2R} + (\tau'')^2}, \tag{34}$$

$$\tau' = \frac{P}{\sqrt{2}x_1x_2}, \tau'' = \frac{MR}{J}, M = P(L + x_2/2), \tag{35}$$

$$R = \sqrt{\frac{x_2^2}{4} + \frac{(x_1 + x_3)^2}{4}}, \tag{36}$$

$$J = 2\left[\sqrt{2}x_1x_2\left(\frac{x_2^2}{12} + \frac{(x_1 + x_3)^2}{4}\right)\right], \tag{37}$$

$$\sigma(\vec{x}) = \frac{6PL}{x_4x_3^2}, \tag{38}$$

$$\delta(\vec{x}) = \frac{4PL^3}{Ex_3^3x_4}, \tag{39}$$

$$P_c(\vec{x}) = \frac{4.013E\sqrt{\frac{x_3^2x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right), \tag{40}$$

$$\begin{aligned} P = 6000 \text{ lb}, L = 14 \text{ in}, E = 30 \times 10^6 \text{ psi}, \\ G = 12 \times 10^6 \text{ psi}, \\ \tau_{max} = 13,600 \text{ psi}, \sigma_{max} = 30,000 \text{ psi}, \delta_{max} = 0.25 \text{ in} \end{aligned} \tag{41}$$

For this problem, the penalty factor approach is used to handle the constraints above, where the fitness function is penalized once any constraint is violated. The hyperparameters of standalone and RL-guided algorithms are tuned with grid search, where +100 hyperparameter configurations are tested. The optimal hyperparameters of PSO are as follows: $N_{particles} = 69$, $c_1 = 2.05$, $c_2 = 2.05$, while for RL-guided PSO, the optimal hyperparameters are $N_{particles} = 95$, $c_1 = 2.05$, $c_2 = 2.05$, $N_{rl} = 9$. The hyperparameters of DE/RL-guided DE are as follows: $F = 0.4$, $C = 0.7$, $N_{rl} = 4$, while for NES/RL-guided NES: $\eta_\sigma = 0.1$, $\eta_B = 0.04$, $N_{pop} = 40$, $N_{rl} = 7$.

The tolerance to escape all constraints is set to 10^{-5}, and we round all results in Table 6 and Figure 6(b) to the nearest fifth significant digit. Figure 6(b) shows the convergence of the welded beam cost function for all standalone and RL-guided algorithms, while a desired optimal limit is plotted based on what the literature have reported for this problem. RL learns the rules $r1 - r5$ in Eqs.(26)-(30) with PPO, then the samples that satisfy these rules are supplied to PSO/DE/NES at every generation with rate N_{rl}. Therefore, we refer to RL-guided PSO/DE/NES as RL$^{r1\to r5}$-guided PSO/DE/NES.

Notice that standalone RL/PPO is excluded from the results and the comparison for similar reasons as we described in Appendix A for the speed reducer problem. Table 6 presents the optimal design variables and the constraint values obtained using both standalone and RL-guided algorithms along with two reference cases for

Table 6. Optimum results for minimization of the welded beam cost

Item	PSO	RL-PSO[a]	DE	RL-DE[a]	NES	RL-NES[a]	SAP[b] [66]	GA[c] [67]
$C(\vec{x})$	3.21561	1.75005	1.72998	1.72755	1.76032	1.72490	1.74831	2.43312
x_1	0.27517	0.19953	0.20565	0.20452	0.20124	0.20573	0.20880	0.24890
x_2	4.31304	3.64491	3.49023	3.49873	3.59411	3.47053	3.42050	6.17300
x_3	9.25129	9.02337	9.03131	9.04054	9.07596	9.03657	8.99750	8.17390
x_4	0.35025	0.20754	0.20619	0.20574	0.20821	0.20574	0.21000	0.25330
$r1$	-5236.15894	-90.07913	-49.67424	-10.69933	-120.05018	-0.17094	-0.33781	-5758.60378
$r2$	-13187.02019	-174.30163	-31.43584	-27.93511	-613.38600	-0.55127	-353.90260	-255.57690
$r3$	-0.07508	-0.00801	-0.00054	-0.00123	-0.00697	0.00000	-0.00120	-0.00440
$r4$	-2.13725	-3.40609	-3.42866	-3.42973	-3.39624	-3.43294	-3.41187	-2.98287
$r5$	-0.15017	-0.07453	-0.08065	-0.07952	-0.07624	-0.08073	-0.08380	-0.12390
$r6$	-0.24208	-0.23560	-0.23555	-0.23556	-0.23590	-0.23554	-0.23565	-0.23416
$r7$	-24064.46402	-153.88119	-37.86002	-2.86378	-237.14030	-0.50646	-363.23238	-4465.27093

[a] $RL^{r1 \rightarrow r5}$-guided PSO/DE/NES
[b] Self-adaptive penalty
[c] Genetic algorithms

comparison with the literature [67, 66]. The best cost value of the welded beam obtained using $RL^{r1 \rightarrow r5}$-guided PSO/DE/NES is close to or better than the reference cases, where all constraints are met. $RL^{r1 \rightarrow r5}$-guided PSO best solution is within 0.09% from [66] and better than [67] by about 28%. Similar to the speed reducer problem, standalone PSO is able to escape the constraint region to find feasible solutions, however again, the converged cost value in Figure 6(b) is not as low as what is achieved by $RL^{r1 \rightarrow r5}$-guided PSO and the two reported values by the literature [67, 66]. Obviously, standalone PSO hangs in a local optima in the seventh epoch of search compared to RL-guided PSO, which obtains help from RL solutions to diversify the search and converge to a satisfactory cost value. The results again show significant improvement in the same PSO algorithm after some guidance from RL.

For NES, the performance of $RL^{r1 \rightarrow r5}$-guided NES is again better than NES as can be told from Table 6 and Figure 6(b). Under similar computational settings, the minimal beam cost achieved by $RL^{r1 \rightarrow r5}$-guided NES is 1.72490 compared to 1.76032 by NES, which is also better than the two reported literature values. Similar to the speed reducer problem, the performance of the standalone DE algorithm is already competitive for the welded beam in terms of the best solution found, as marginal improvement is achieved by $RL^{r1 \rightarrow r5}$-guided DE. However, $RL^{r1 \rightarrow r5}$-guided DE saved about 200 function evaluations compared to DE in finding its best solution in Table 6. In summary, in terms of the performance for the welded beam problem, $RL^{r1 \rightarrow r5}$-guided NES with objective value 1.72490 is the best algorithmic performance over all our proposed algorithms in terms of the agreement with the literature.

References

[1] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.

[2] K. Deb, Multi-objective optimization using evolutionary algorithms, Vol. 16, John Wiley & Sons, 2001.

[3] D. Ackley, M. Littman, Interactions between learning and evolution, Artificial life II 10 (1991) 487–509.

[4] A. Stafylopatis, K. Blekas, Autonomous vehicle navigation using evolutionary reinforcement learning, European Journal of Operational Research 108 (2) (1998) 306–318.

[5] D. E. Moriarty, A. C. Schultz, J. J. Grefenstette, Evolutionary algorithms for reinforcement learning, Journal of Artificial Intelligence Research 11 (1999) 241–276.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, nature 518 (7540) (2015) 529–533.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971(2015) .

[8] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, et al., Population based training of neural networks, arXiv preprint arXiv:1711.09846(2017) .

[9] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, K. Kavukcuoglu, Hierarchical representations for efficient architecture search, arXiv preprint arXiv:1711.00436(2017) .

[10] F. C. Fernandez, W. Caarls, Parameters tuning and optimization for reinforcement learning algorithms using evolutionary computing, in: 2018 International Conference on Information Systems and Computer Science (INCISCOS), IEEE, 2018, pp. 301–305.

[11] S. Risi, J. Togelius, Neuroevolution in games: State of the art and open challenges, IEEE Transactions on Computational Intelligence and AI in Games 9 (1) (2015) 25–41.

[12] K. O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evolutionary computation 10 (2) (2002) 99–127.

[13] S. Whiteson, P. Stone, Evolutionary function approximation for reinforcement learning, Journal of Machine Learning Research 7 (May) (2006) 877–917.

[14] T. Salimans, J. Ho, X. Chen, S. Sidor, I. Sutskever, Evolution strategies as a scalable alternative to reinforcement learning, arXiv preprint arXiv:1703.03864(2017) .

[15] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, J. Clune, Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning, arXiv preprint arXiv:1712.06567(2017) .

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International conference on machine learn-

ing, 2016, pp. 1928–1937.
[17] S. Khadka, K. Tumer, Evolution-guided policy gradient in reinforcement learning, in: Advances in Neural Information Processing Systems, 2018, pp. 1188–1200.
[18] C. Colas, O. Sigaud, P.-Y. Oudeyer, Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms, arXiv preprint arXiv:1802.05054(2018) .
[19] M. M. Drugan, Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms, Swarm and evolutionary computation 44 (2019) 228–246.
[20] K. Sim, E. Hart, An improved immune inspired hyper-heuristic for combinatorial optimisation problems, in: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, 2014, pp. 121–128.
[21] J. E. Pettinger, R. M. Everson, Controlling genetic algorithms with reinforcement learning, in: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, 2002, pp. 692–692.
[22] G. Karafotias, A. E. Eiben, M. Hoogendoorn, Generic parameter control with reinforcement learning, in: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, 2014, pp. 1319–1326.
[23] Y. Sakurai, K. Takada, T. Kawabe, S. Tsuruta, A method to control parameters of evolutionary algorithms by using reinforcement learning, in: 2010 Sixth International Conference on Signal-Image Technology and Internet Based Systems, IEEE, 2010, pp. 74–79.
[24] Q. Chen, M. Huang, Q. Xu, H. Wang, J. Wang, Reinforcement learning-based genetic algorithm in optimizing multidimensional data discretization scheme, Mathematical Problems in Engineering 2020(2020) .
[25] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: A survey of the state of the art, Journal of the Operational Research Society 64 (12) (2013) 1695–1724.
[26] E. Özcan, M. Misir, G. Ochoa, E. K. Burke, A reinforcement learning: great-deluge hyper-heuristic for examination timetabling, in: Modeling, Analysis, and Applications in Metaheuristic Computing: Advancements and Trends, IGI Global, 2012, pp. 34–55.
[27] K. A. Dowsland, E. Soubeiga, E. Burke, A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation, European Journal of Operational Research 179 (3) (2007) 759–774.
[28] J. Blazewicz, E. K. Burke, G. Kendall, W. Mruczkiewicz, C. Oguz, A. Swiercz, A hyper-heuristic approach to sequencing by hybridization of dna sequences, Annals of Operations Research 207 (1) (2013) 27–41.
[29] J. Broekens, K. Hindriks, P. Wiggers, Reinforcement learning as heuristic for action-rule preferences, in: International Workshop on Programming Multi-Agent Systems, Springer, 2010, pp. 25–40.
[30] Q. Cai, W. Hang, A. Mirhoseini, G. Tucker, J. Wang, W. Wei, Reinforcement learning driven heuristic optimization, arXiv preprint arXiv:1906.06639(2019) .
[31] M. Santos, V. López, G. Botella, et al., Dyna-h: A heuristic planning reinforcement learning algorithm applied to role-playing game strategy decision systems, Knowledge-Based Systems 32 (2012) 28–36.
[32] A. Pan, W. Xu, L. Wang, H. Ren, Additional planning with multiple objectives for reinforcement learning, Knowledge-Based Systems 193 (2020) 105392.
[33] H. Wang, M. Gu, Q. Yu, Y. Tao, J. Li, H. Fei, J. Yan, W. Zhao, T. Hong, Adaptive and large-scale service composition based on deep reinforcement learning, Knowledge-Based Systems 180 (2019) 75–90.
[34] L. Machado, R. Schirru, The ant-q algorithm applied to the nuclear reload problem, Annals of Nuclear Energy 29 (12) (2002) 1455–1470.
[35] D. J. Kropaczek, P. J. Turinsky, In-core nuclear fuel management optimization for pressurized water reactors utilizing simulated annealing, Nuclear Technology 95 (1) (1991) 9–32.
[36] G. T. Parks, Multiobjective pressurized water reactor reload core design by nondominated genetic algorithm search, Nuclear Science and Engineering 124 (1) (1996) 178–187.
[37] M. L. Fensin, Optimum boiling water reactor fuel design strategies to enhance reactor shutdown by the standby liquid control system, Master's thesis, University of Florida (2004).
[38] M. I. Radaideh, D. Price, D. O'Grady, T. Kozlowski, Advanced bwr criticality safety part i: Model development, model benchmarking, and depletion with uncertainty analysis, Progress in Nuclear Energy 113 (2019) 230–246.
[39] R. T. Marler, J. S. Arora, Survey of multi-objective optimization methods for engineering, Structural and multidisciplinary optimization 26 (6) (2004) 369–395.
[40] O. Kramer, A review of constraint-handling techniques for evolution strategies, Applied Computational Intelligence and Soft Computing 2010(2010) .
[41] M. Edenius, K. Ekberg, B. H. Forssén, D. Knott, Casmo-4, a fuel assembly burnup program, user's manual, Studsvik0SOA-9501, Studsvik of America, Inc(1995) .
[42] D. Knott, M. Edenius, Validation of the casmo-4 transport solution, In: Joint international conference on mathematical methods and supercomputing in nuclear applications, Karlsruhe, Germany, April 19-23, 1993(1993) .
[43] M. Pusa, Incorporating sensitivity and uncertainty analysis to a lattice physics code with application to casmo-4, Annals of Nuclear Energy 40 (1) (2012) 153–162.
[44] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym, arXiv preprint arXiv:1606.01540(2016) .
[45] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: a methodological tour d'horizon, European Journal of Operational Research(2020) .
[46] J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel, High-dimensional continuous control using generalized advantage estimation, arXiv preprint arXiv:1506.02438(2015) .
[47] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347(2017) .
[48] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International conference on machine learning, 2015, pp. 1889–1897.
[49] A. Hill, A. Raffin, M. Ernestus, A. Gleave, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, et al., Stable baselines, GitHub repository(2018) .
[50] D. Whitley, A genetic algorithm tutorial, Statistics and computing 4 (2) (1994) 65–85.
[51] B. Q. Do, L. P. Nguyen, Application of a genetic algorithm to the fuel reload optimization for a research reactor, Applied Mathematics and Computation 187 (2) (2007) 977–988.
[52] F. Alim, K. Ivanov, S. H. Levine, New genetic algorithms (ga) to optimize pwr reactors: Part i: Loading pattern and burnable poison placement optimization techniques for pwrs, Annals of Nuclear Energy 35 (1) (2008) 93–112.
[53] C. M. del Campo, J. Francois, H. Lopez, Axial: a system for boiling water reactor fuel assembly axial optimization using genetic algorithms, Annals of Nuclear Energy 28 (16) (2001) 1667–1682.
[54] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, C. Gagné, DEAP: Evolutionary algorithms made easy, Journal of Machine Learning Research 13 (2012) 2171–2175.
[55] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, Journal of global optimization 11 (4) (1997) 341–359.
[56] D. Wierstra, T. Schaul, J. Peters, J. Schmidhuber, Natural evolution strategies, in: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 3381–3387.
[57] H.-G. Beyer, H.-P. Schwefel, Evolution strategies–a comprehensive introduction, Natural computing 1 (1) (2002) 3–52.
[58] T. Glasmachers, T. Schaul, S. Yi, D. Wierstra, J. Schmidhuber, Exponential natural evolution strategies, in: Proceedings of the

12th annual conference on Genetic and evolutionary computation, 2010, pp. 393–400.
[59] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by simulated annealing, science 220 (4598) (1983) 671–680.
[60] T. K. Park, H. G. Joo, C. H. Kim, H. C. Lee, Multiobjective loading pattern optimization by simulated annealing employing discontinuous penalty function and screening technique, Nuclear Science and Engineering 162 (2) (2009) 134–147.
[61] T. Rogers, J. Ragusa, S. Schultz, R. S. Clair, Optimization of pwr fuel assembly radial enrichment and burnable poison location based on adaptive simulated annealing, Nuclear Engineering and Design 239 (6) (2009) 1019–1029.
[62] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95-International Conference on Neural Networks, Vol. 4, IEEE, 1995, pp. 1942–1948.
[63] M. Clerc, J. Kennedy, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, IEEE transactions on Evolutionary Computation 6 (1) (2002) 58–73.
[64] T. Ray, K.-M. Liew, Society and civilization: An optimization algorithm based on the simulation of social behavior, IEEE Transactions on Evolutionary Computation 7 (4) (2003) 386–396.
[65] B. Akay, D. Karaboga, Artificial bee colony algorithm for large-scale problems and engineering design optimization, Journal of intelligent manufacturing 23 (4) (2012) 1001–1014.
[66] C. A. C. Coello, Use of a self-adaptive penalty approach for engineering optimization problems, Computers in Industry 41 (2) (2000) 113–127.
[67] K. Deb, Optimal design of a welded beam via genetic algorithms, AIAA journal 29 (11) (1991) 2013–2015.
[68] A. Banks, J. Vincent, C. Anyakoha, A review of particle swarm optimization. part i: background and development, Natural Computing 6 (4) (2007) 467–484.
[69] A. Banks, J. Vincent, C. Anyakoha, A review of particle swarm optimization. part ii: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications, Natural Computing 7 (1) (2008) 109–124.
[70] J. Golinski, An adaptive optimization system applied to machine synthesis, Mechanism and Machine Theory 8 (4) (1973) 419–436.