*Title Page

Physics-based Reinforcement Learning Optimisation of Nuclear Assembly Design

Majdi I. Radaideh[a], Isaac Wolverton[b], Jousha Joseph[b], Benoit Forget[a], Nicholas Roy[b], Koroush Shirvan[a]

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

[b] MIT Quest for Intelligence, Massachusetts Institute of Technology, Cambridge, MA 02139, United States

Physics-based Reinforcement Learning Optimisation of Nuclear Assembly Design

Majdi I. Radaideh[a,*], Isaac Wolverton[b], Jousha Joseph[b], Benoit Forget[a], Nicholas Roy[b], Koroush Shirvan[a]

[a] *Department of Nuclear Science and Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, United States*

[b] *MIT Quest for Intelligence, Massachusetts Institute of Technology, Cambridge, MA 02139, United States*

Abstract

Optimisation of nuclear fuel assemblies if performed effectively, will lead to fuel efficiency improvement, cost reduction, and safety assurance. However, assembly optimisation involves solving high-dimensional and computationally expensive combinatorial problems. As such, fuel designers' expert judgement has commonly prevailed over the use of stochastic optimization (SO) algorithms such as genetic algorithms and simulated annealing. To improve the state-of-art, we explore a class of artificial intelligence (AI) algorithms, namely, reinforcement learning (RL) in this work. We propose a physics-based AI optimisation methodology by establishing a connection through reward shaping between RL and the tactics fuel designers follow in practice by moving fuel rods in the assembly to meet specific constraints and objectives. The methodology utilizes RL algorithms, deep Q learning and proximal policy optimisation, and compares their performance to SO algorithms. The methodology is applied on two boiling water reactor assemblies of low-dimensional ($\sim 2\times10^6$ combinations) and high-dimensional ($\sim 10^{31}$ combinations) natures. The results demonstrate that RL is more effective than SO in solving high dimensional problems, i.e., 10x10 assembly, through embedding expert knowledge in form of game rules and effectively exploring the search space. For a given computational resources and timeframe relevant to fuel designers, RL algorithms outperformed SO through finding more feasible patterns, 4-5 times more than SO, and through increasing search speed, as indicated by the RL outstanding computational efficiency. The results of this work clearly demonstrate RL effectiveness as another decision support tool for nuclear fuel assembly optimisation.

Keywords: Combinatorial Optimisation, Deep Reinforcement Learning, BWR Assembly Optimisation, Genetic Algorithms, Deep Q Learning, Proximal Policy Optimisation

1. Introduction

The sustainability of the existing light water reactor fleet is one of the main missions of U.S. nuclear industry and Department of Energy. The existing fleet provides roughly half of all carbon-free electricity in the United States. However, the number of reactors online has declined in recent years, mainly driven by cost. Reducing the nuclear fuel cost is one way to improve fleet efficiency. The nuclear fuel designers dictate the number and attributes of an assembly in terms of its enrichment and burnable poison loading. In effect, fuel designers attempt to solve a "combinatorial optimization" problem by utilizing expert judgement, nuclear design principles, and physics-based tools. Combinatorial optimisation [1] in nuclear reactor design and operation is a known problem that aims to find an optimal pattern from a finite set of patterns [2]. Indeed, the search space for combinatorial optimization is finite by definition, and thus an optimal solution always exists.

Nuclear fuel design involves two common problems: (1) core optimisation and (2) assembly optimisation. *Core optimisation* aims at finding the best loading pattern of all assemblies in the core such that the reactor operation is economic and meets safety constraints [2]. *Assembly optimisation* (the focus of this work) aims on finding the optimal material composition and location of all fuel rods in the assembly such that when the assembly

*Corresponding Author: Majdi I. Radaideh (radaideh@mit.edu)

Nomenclature

α, β_0	DQN prioritized experience replay parameters	N_{GAD}	Number of gadolinium (GAD) fuel rods
χ	Attribute swap/perturbation probability	N_{gen}	GA number of generations
ϵ	DQN exploration probability	N_{pop}	GA number of population per generation
η	Algorithm efficiency	N_{steps}	SA number of annealing steps
γ	Reward discount factor	N_{UO_2}	Number of UO_2 fuel rods
λ	PPO Bias-variance tradeoff parameter	N_{warmup}	Samples to initialize DQN replay memory
a	Action to take by the agent	opt_{epochs}	PPO number of supervised training epochs
B	Mini-batch size	PPF	Power Peaking Factor
C	DQN target model update frequency	r	Reward
CL	Fuel cycle length (days)	s	Agent current state
$CLIP$	PPO clipping parameter	T_{max}, T_{min}	SA max/min annealing temperatures
CX	GA crossover probability	VF_{coef}	PPO value function loss coefficient
E	Assembly average UO_2 enrichment	BWR	Boiling Water Reactor
ENT_{coef}	PPO entropy coefficient	DQN	Deep Q Learning
F_{train}	Model training frequency	GA	Genetic Algorithm
G	Assembly average GAD enrichment	KBS	Knowledge-Based Systems
k_∞	Infinite neutron multiplication factor	PPO	Proximal Policy Optimisation
lr	Learning rate	RL	Deep Reinforcement Learning
MUT	GA mutation probability	SA	Simulated Annealing
N_{anneal}	DQN exploration fraction to anneal ϵ	SO	Stochastic Optimisation

is introduced in the core, economic and safety constraints are satisfied [3]. A review of related literature on optimization techniques is included in section 3.1.

For assembly optimisation, unlike pressurized water reactors (PWR), boiling water reactor (BWR) designs feature more heterogeneous fuel enrichment distribution radially [4], which will also be seen in this work. For PWRs, some utilities adopt optimisation tools to find the most economic core design more rapidly, for example ROSA [5] (Reload Optimisation with Simulated Annealing). However, such stochastic optimisation (SO) based frameworks while fast for individual pattern evaluation, are often computationally expensive for finding high performing solutions, thus their commercial application has not found much adoption for more complex problems as in the case of BWRs. It is also worth mentioning that SO code packages such has ROSA leverage surrogate models to reduce computational burden and thus do not rely on licensed methodologies. Therefore, when the best design option is found by SO, manual tuning still needs to be performed by the licensed codes. Aside from classical SO, to the authors' knowledge, there have been very limited attempts so far to investigate the performance of modern RL algorithms (e.g., DQN, PPO) to support nuclear engineering decisions regarding fuel assembly optimisation, either for PWR or BWR. RL algorithms would prove effective if they demonstrate promising performance in embedding domain or expert knowledge through reward shaping, in exploring the search space effectively, and in their ability to more effectively find a global optimum than standard SO in a problem with many local optima. Accordingly, we explore the ability to train an intelligent system by RL that is able to learn from interactions with physics-based environments and prior expert knowledge, such that it can take proper actions in a short amount of time to efficiently arrange and optimise nuclear fuel within the assembly. RL is compared to SO algorithms (i.e., GA, SA),

which act as baselines that have been widely investigated in literature.

In this work, we provide important definitions about the design of nuclear assemblies of interest in section 2. The methodology is described in section 3, which starts with a literature review of related work, followed by the optimisation strategy and the process of building physics-based environments to facilitate RL and SO. Next, the mathematical foundation of RL and SO algorithms and their connection to the physics-based environment are described, followed by the code deployment. The results of this paper are presented in two case studies in section 4 and section 5, respectively. The first case study highlights a small and low-dimensional nuclear assembly (BWR 6x6) with global optima known beforehand using brute-force search, where RL/SO algorithms are assessed and compared to each other. Next, RL is compared to SO in a bigger high-dimensional nuclear assembly (BWR 10x10), that is also limited by expensive simulation costs. Finally, the conclusions of this work are presented in section 6.

2. Nuclear Fuel Assembly Design

The system optimised in this work is the nuclear fuel assembly; a top view of two BWR assembly designs of interest to this work are sketched in Figure 1. Assembly optimisation is seen as a permutation with repetition problem with cost proportional to $O(m^n)$, where m is the number of fuel types (i.e., choices) to pick from, while n is the number of fuel rod locations to optimise (i.e., number of times to choose). To reduce the search space, researchers tend to take advantage of problem symmetry to reduce the value of n. Due to the multiobjective nature of nuclear optimisation, weighted scalarization has been widely used to construct the objective/fitness/cost function [2, 6, 7]

$$\min_{\vec{x}} \; F(\vec{x}) = \sum_{i=1}^{k} w_i f_i(\vec{x}) \tag{1}$$

where k is the number of single objectives included in the optimisation and w is the corresponding weight determined by the analyst based on prior experience and preliminary convergence tests. The first design in Figure 1 has a dimension of 6x6 with 36 fuel rods, while the second design is 10x10, with 92 fuel rods, and 2 large water rods occupying the remaining 8 positions. The fuel material consists of Uranium Oxide (UO_2). The 6x6 assembly in Figure 1 features two types of UO_2 fuel with 1.87% and 2.53% U-235 enrichment. In addition, some fuel rods have UO_2 mixed with Gadolinium Oxide (GAD), which absorbs neutrons and helps in controlling the fission reaction in large assemblies as in Figure 1(b). GAD enrichment is also variable and typically limited to below 10% weight fraction. Optimising the number of GAD rods and GAD enrichment is extremely important for the safety and economy of the nuclear assembly. Both assembly designs satisfy 1/2 symmetry, which makes the total number of possible rod locations to be 21 for the 6x6 and 51 for the 10x10. The numbered rods on and below the diagonal line in Figure 1 are included in the optimisation process. It is worth mentioning that the 6x6 assembly has an interesting feature as all possible permutations can be evaluated by brute-force techniques in a reasonable amount of time. This means that optimum solutions are known beforehand as will be shown later in Case Study 1, section 4. In this work, the two fuel assemblies are modeled and simulated using the CASMO-4 code. CASMO-4 [8] is a multigroup two-dimensional transport theory code for burnup calculations of PWR and BWR fuel assemblies based upon the Method of Characteristics. The term beginning of life (BOL) is used when fuel depletion is not simulated, which can be seen as a steady state simulation of the first time step in the cycle. The term end of life (EOL)

involves depleting the fuel and simulating time-dependent behavior of the cycle. Both BOL and EOL simulations are conducted under fixed fuel temperature of 900 K and moderator temperature of 560 K. For each assembly design in this work, the following assembly attributes are of high importance to be measured and optimised:

1. k_∞: The infinite neutron multiplication factor, which is a measure of the change in the fission neutron population in an infinite multiplying system (e.g., reflected assembly) from one neutron generation to the subsequent generation. k_∞ is a critical safety parameter for nuclear reactor control, thus it should be optimised.
2. PPF: Power peaking factor, which is the ratio of the highest fuel rod power in the assembly to the average assembly power by all rods. PPF is a constrained safety parameter to be minimized and maintained below a threshold value.
3. N_{UO_2}: Number of pure UO_2 rods in the assembly.
4. N_{GAD}: Number of GAD rods in the assembly. Current BWR nuclear designs have N_{UO_2} as 5-7 times of N_{GAD}.
5. E: Average assembly enrichment, which is the mean of U-235 enrichment in all assembly rods (N_{UO_2}+N_{GAD}). Average enrichment is an economic parameter, as higher E values require additional production cost.
6. G: Average GAD enrichment, which is the mean of GAD enrichment in the assembly GAD rods (N_{GAD}). Similar to E, GAD enrichment is an economic parameter, where higher G values require additional costs.
7. CL: Cycle length (or assembly burnup), which is a measure of the nuclear fuel cycle length that the assembly design can supply. It is measured with a unit called effective full power days (for simplicity days in this work), which is the number of days the reactor can operate at full power under the current fuel loading. Obviously, CL is an economic parameter that maximizes the benefit from the fuel assembly. For BOL problems, where fuel depletion is not simulated, $CL = 0$. *In this work, when fuel depletion is simulated, CL is determined to be the time in days after the k_∞ peak at which $k_\infty = 0.95$.*

A sample design for a BWR 6x6 assembly is shown in Figure 1(a). The design is originally obtained from the Gundremmingen-A nuclear power plant, which started commercial operation in 1967 in Germany [9] and used later for spent fuel analysis and nuclear code validation [10]. For this design, $N_{UO_2} = 36$, and since this is a BOL problem with assembly that has no GAD rods, then G, N_{GAD}, and CL are all equal to zero. The average enrichment E can be calculated as $(2.53 \times 29 + 1.87 \times 7)/36 = 2.402\%$. To calculate k_∞ and PPF, the 6x6 assembly is simulated as a BOL problem using the CASMO-4 nuclear code, which resulted in $k_\infty = 1.26072$ and $PPF = 1.455$. The 6x6 design will be used in Case Study 1 later in section 4.

A sample design for a BWR 10x10 assembly is shown in Figure 1(b), which is known as GE14. It is a relatively new design, originally obtained from [4]. For this design, there are $N_{UO_2} = 74$ rods and $N_{GAD} = 18$ rods, forming a total of 92 fuel rods. The values of E and G can be inferred from Figure 1(b), which are respectively 4.305% and 7.72%. The remaining responses are calculated based on CASMO-4 simulation with fuel depletion, which are as follows: $k_\infty^{max} = 1.09814$, $PPF = 1.62$, and $CL = 1450$ days, where CL is determined at $k_\infty = 0.95$. The 10x10 design will be used in Case Study 2 later in section 5.

The previous reference designs are presented here to demonstrate the process of modeling and simulation. The geometrical features for the 6x6 and 10x10 assemblies such as fuel rod and assembly dimensions, number of fuel

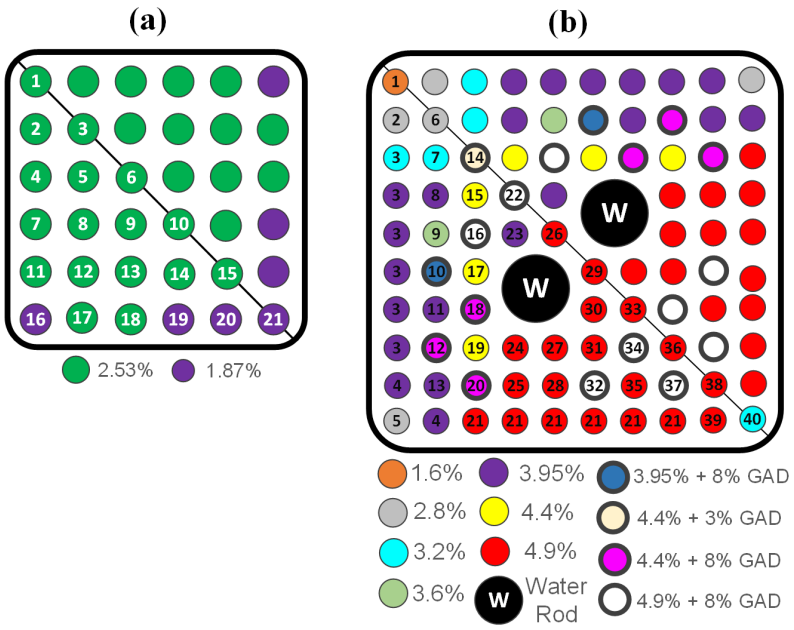

Figure 1: Top view of the nuclear fuel assemblies used in the analysis: (a) BWR 6x6 with 1/2 symmetry and (b) BWR 10x10 with 1/2 symmetry (Numbers on the fuel rods are explained in section 4 and section 5)

rods, and number of water rods will be fixed during optimisation. However, the material aspects such as UO_2 enrichment and location as well as GAD enrichment and location will be the main focus of optimisation. From previous descriptions and prior efforts on assembly optimisation, four challenges are known to hinder an accurate decision making regarding final designs: (1) a very large search space of possible patterns, (2) multiple objectives to optimise and constraints to meet, (3) large number of local optima (very likely to find hundreds of patterns that have exact same fitness), and (4) computational burden associated with computer simulation. These challenges emphasize the need for intelligent systems to reduce time and efforts solving such problems, some of which are described next.

3. Methodology

3.1. Related Work

For most engineering problems, the search space size of combinatorial problems grows drastically with the size of the input space, e.g., $O(n!)$. Popular combinatorial problems include the traveling salesman problem (TSP), the minimum spanning tree problem, and the knapsack problem. Historically, combinatorial optimisation has been approached by a variety of methods, including but not limited to dynamic programming [11], adaptive search [12], simulated annealing (SA) [13], genetic algorithms (GA) [14], tabu search [15], neural networks [16], and many others. A review on metaheuristics for combinatorial optimisation can be found in [17].

Classical reinforcement learning (RL) through REINFORCE and Q-learning (Q originated from quality) was first used to approach combinatorial optimisation [18, 19]. The extension to deep RL has also being explored to solve combinatorial problems [20]. *Since this work mainly focuses on deep RL, for convenience, RL will be used to refer deep RL.* RL algorithms offer the ability to train an agent, expert, or a knowledge-based system (KBS) that learns

how to take proper actions to find an optimised configuration, which sounds attractive to combinatorial optimization [21]. This learning ability highlights a major difference between RL and GA/SA methods. Deep Q learning (DQN) [22] and proximal policy optimisation (PPO) [23] are common and widely used RL algorithms. Earlier efforts featured combining classical Q-learning (RL) with ant colony system (evolutionary) into a technique called Ant-Q to solve the TSP problem. Ant-Q demonstrated competitive performance compared to other approaches based on neural networks and local search [24]. Two approaches based on policy gradients were proposed to perform neural combinatorial optimisation with RL [21], which were inspired originally from the pointer networks [25]. The *pretraining approach* uses a training set to optimize a recurrent neural network which parameterizes a stochastic policy. The *active search* starts from a random policy and iteratively optimizes the recurrent neural network parameters on a single test instance. Both approaches use a problem-dependent objective/cost function for RL reward shaping. Another study [26] proposed a combination of RL and graph embedding to learn a policy that can be used to construct a solution in combinatorial optimisation. Other efforts on applying and enhancing RL in optimisation include using the Metropolis criterion of SA to balance between exploration/exploitation in Q-learning [27], multi-agent (RL team) search with decentralization to handle memory issues [28], and using stochastic policy gradient algorithms to solve the vehicle routing problem [29].

Both core and assembly problems have been approached by almost similar methods that belong to evolutionary algorithms. For core optimisation, successful applications of simulated annealing [2], genetic algorithms [30], particle swarm optimization [31], continuous firefly algorithm [32], and recurrent neural networks [33] have been demonstrated to produce high quality solutions (i.e., core loading patterns) to inform the designers. Aside from evolutionary techniques, the prescribed Ant-Q approach that includes RL was also brought to solve the nuclear core optimisation problem [34]. For assembly optimisation, GA was utilized by [35] to optimise the axial enrichment in a 3D boiling water reactor (BWR) assembly to minimize the overall average enrichment (i.e., reducing costs) needed to obtain the reference cycle length, while meeting the safety constraints. Unlike PWRs, BWR designs feature larger core size with ~ 700 assemblies, compensated by smaller assembly size (92 fuel rods per assembly), and more heterogeneous fuel enrichment distribution radially and axially [4], which will also be seen in this work. Radial optimisation in a 2D BWR assembly was conducted by [3] using tabu search. Assuming 10 different fuel types in the assembly, the authors were able to achieve some reductions on the average enrichment, and a large reduction in computational time to find optimum solutions. Additional efforts on using Hopefield neural networks [36], adaptive simulated annealing [37], and differential evolutionary algorithms [38] for fuel assembly design optimisation have been conducted.

3.2. Environment Structure

Since most of the biggest advances in RL research have been in video game applications [22], most of its terminology are inherited from computer games, meaning that we need to observe our problem as a game to facilitate the analysis. OpenAI Gym [39] is a toolkit for building environments to test RL algorithms in efficient form. Following the OpenAI Gym data structure, we developed a CASMO-4 environment based on the geometries in Figure 1. There are many possibilities for the game scenario and the rules of the game will dictate the efficiency of the RL framework. We describe the particular game strategy for each case study later in its appropriate section.

Compared to OpenAI Gym default functions, we have added two additional functions called *Fit* and *Monitor* to merge SO methods seamlessly into the OpenAI Gym platform, which is RL-oriented.

In particular, our physics-based environment consists of five major functions: constructor, step, fit, monitor, reset, and render. The **constructor** initializes all environment variables, including the action space and state space for RL. The **step** function is exclusive to RL, it receives input as the action to take, and returns output as the new state, reward for the taken action, and whether the episode reaches the end or not. The **fit** function is exclusive to SO algorithms (i.e., GA, SA), it takes enrichment in each rod position as input and returns reward as output. Compared to the **step** function, the **fit** function lacks temporality and acts as a black-box function that takes input and returns output regardless of the problem state. Finally, **monitor**, **reset**, and **render** functions log the data for RL/SO, reset the environment back to initial state for RL, and plot the assembly board showing the enrichment in each rod location, respectively. Checkpointing (if necessary) can be done through saving the trained model at different time steps to be used for other training purposes.

Based on the previous descriptions, it is worth defining the following units that we will use to describe the training progress:

- *Time Step (TS)*: A time step involves single call of the step function, the agent takes an action, observes reward and next state for the next time step. TS may or may not involve a CASMO-4 call, and TS is exclusive to RL methods.
- *Episode*: A collection of time steps that mark several interactions between the RL agent and the environment. The size of episode is directly connected with the game or optimisation strategy adopted. For SO, the episode involves one interaction with the environment.
- *Epoch*: A collection of episodes that help drawing statistics about the agent performance (e.g., reward mean, variance, max, min). The size of epoch is determined by the analyst to balance between computational costs and having sufficient sample size to infer accurate performance. Epoch is used for both RL and SO.
- *CASMO4 Call (C4C)*: this is a universal unit for both RL and SO, which counts the number of CASMO-4 calls during training regardless of the method used. Therefore, to compare between the algorithms in terms of speed and efficiency, the efficiency parameter can be defined based on C4C

$$\eta = \left(1 - \frac{\text{C4C for } 1^{st} \text{ Feasible}}{\text{Total C4Cs}}\right)\%. \tag{2}$$

Before moving forward, it is worth highlighting two major quantities that would come from epoch statistics: mean reward and max reward. Mean reward highlights the ability of RL/SO algorithm to consistently produce high quality solutions. This feature is specifically important to RL, as it implies that the agent or knowledge-based system (KBS) is well-trained. In contrast, max reward highlights the best solution or assembly pattern found during training, which is the end goal of optimisation (i.e., what the designer is looking for). Patterns of max reward may appear occasionally or in intermittent form during training, while an excellent KBS with high mean reward is expected to recover such patterns more frequently.

3.3. Deep Reinforcement Learning (RL)

RL started with the classical Q-learning [40], which relies on tables to store Q-value for every possible state-action pair in the problem, and then using that table to determine the action with maximum Q value to guide agent learning. In simple Q-learning, Q value is updated recursively, as derived from the Bellman equation

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha)\overbrace{Q(s_t, a_t)}^{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \overbrace{\Big[\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{optimum future value}}\Big]}^{\text{learned value}}, \tag{3}$$

where s_t, a_t, and s_{t+1} are the current state, current action, and next state, respectively. However, in most real-world applications, the state-action pairs can be extremely large, causing not only memory issues in storing them, but also large computational costs in interpreting these tables. Alternatively, neural networks are used to predict Q-value for each possible action based on previous observations by the agent, and then Q-learning can decide which action to take based on these predicted Q-values. Training deep neural networks to approximate the Q function is known as deep reinforcement learning [22], or simply reinforcement learning (RL) for our work. However, a simple introduction of a neural network to approximate the Q function will be ineffective since the target network is continuously changing at every TS, causing training instabilities and overfitting. The work by [22] resolved this issue by using a concept of two parallel networks, one called primary Q network used for training, and another copy of it called target Q network used for prediction. The primary network is usually updated every TS (or after F_{train}), while the target network is frozen and updated after a specific number of steps, controlled by the parameter C, where $C >> F_{train}$. The target model update involves simply copying the current weights of the primary network into the target network. The DQN objective function during training is to minimize the losses between the two network predictions as

$$Loss = \underbrace{(r_k + \gamma \cdot \max_{a'} Q(s_{k+1}, a'; \bar{\theta})}_{\text{Target}} - \underbrace{Q(s_k, a_k; \theta))}_{\text{Predicted}}^2, \tag{4}$$

where $\bar{\theta}$ and θ are used to distinguish between the weights of the target and primary networks, respectively, k is the sample index in the replay memory, γ is the discount factor (~0.8-0.99), and r_k, s_k, a_k, and s_{k+1} are respectively the reward, current state, action to take, and next state associated with sample k in the memory. Minimizing the losses can be done by training on gradient descent to find the best θ value. Hasselt et al. [41] illustrated that vanilla DQN introduces systematic overestimation as the max operator uses the same Q values from the primary network to select and evaluate the action, causing overoptimistic performance. The solution is to use the primary network for action selection and the target network for action evaluation, where the loss in Eq.(4) can be redefined as

$$Loss = \underbrace{(r_k + \gamma \cdot Q(s_{k+1}, \max_{a'} Q(s_{k+1}, a'; \theta); \bar{\theta})}_{\text{Target}} - \underbrace{Q(s_k, a_k; \theta))}_{\text{Predicted}}^2, \tag{5}$$

where now the target is changed as the action is determined by weights θ, but evaluated by weights $\bar{\theta}$. The algorithm used in this work is given in Algorithm 1. The algorithm starts by initializing a set of hyperparameters.

First, ϵ balances the exploration vs exploitation dilemma in RL, and it represents the fraction of time the agent spent in exploring new knowledge (i.e., taking random actions), where $\epsilon = 1$ refers to a complete exploration, while $\epsilon = 0$ means the agent relies completely on its experience/memory in taking actions. We gradually reduce ϵ during training from ϵ_{init} to ϵ_{end}, where the training period at which ϵ is reduced is controlled by N_{anneal}. After N_{anneal} is passed, ϵ is fixed to ϵ_{end}. The replay memory D is another feature of DQN, where past experiences are stored in a buffer during training. A mini-batch of size B is used to estimate the losses in Eq.(5), by running gradient descent with learning rate lr. The memory is initialized with some random transitions determined by $N_{warmup} > 4B - 5B$, where B is the mini-batch size. The frequency of sampling transitions from the replay memory and updating the primary network is controlled by F_{train}, where $F_{train} = 1$ means that the primary network is updated every TS. Afterwards, three loops are executed, for epochs, episodes, and time steps, respectively. By the end of each epoch, the average reward is determined and used to indicate if the model is improved or not, if yes, the model is saved.

Algorithm 1 Deep Q Learning

•Set hyperparameters: $\epsilon_{init}, \epsilon_{end}, \gamma, C, B, N_{warmup}, N_{anneal}, F_{train}, lr$
•Initialize primary/target Q networks and set $\bar{\theta} = \theta$
•Initialize replay memory D with capacity N_{warmup}
•Initialize $r^{best} = -\infty$
for Epoch $i = 1$ to $EPOCH$ **do**
  for Episode $j = 1$ to $EPISODE$ **do**
    •Initialize first state s_1 (i.e., Reset Environment)
    for Time Step $t = 1$ to $TIME$ **do**
      •With probability ϵ, select random action a_t
      •Otherwise select $a_t = \max_a Q(s_t, a; \theta)$
      •Execute action a_t and observe next state s_{t+1}, and reward r_t
      •Set $s_{t+1} = s_t$
      if Frequency F_{train} is reached **then**
        •Sample random mini-batch B of transitions (s_k, a_k, r_k, s_{k+1}) from memory D
        if Episode ends at $k+1$ **then**
          •$y_k = r_k$
        else
          •$y_k = r_k + \gamma \cdot Q(s_{k+1}, max_{a'} Q(s_{k+1}, a'; \theta); \bar{\theta})$
        •Run gradient descent with learning rate lr on $(y_k - Q(s_k, a_k; \theta))^2$ with respect to θ
      •If target model update frequency C is reached, then set $\bar{\theta} = \theta$
      •If N_{anneal} is reached, then set $\epsilon = \epsilon_{end}$
      •Otherwise anneal ϵ between ϵ_{init} and ϵ_{end}
  •Calculate mean reward $\bar{r}_i$ for the current epoch
  if $\bar{r}_i > r^{best}$ **then**
    •Save model M_i for current epoch
    •Set $r^{best} = \bar{r}_i$

Two additional enhancements to DQN are highlighted briefly here since they are important for DQN performance even in their default settings: (1) dueling and (2) prioritized replay, the reader is referred to the following references for detailed information [42, 43]. The dueling architecture [42] makes the agent able to evaluate a state without caring about the effect of each action from that state. Dueling is useful for cases where certain actions from a state do not affect the environment at all; these actions are not evaluated, giving the algorithm more speed. So far for DQN, experience transitions are uniformly sampled from the memory, replaying them in same frequency regardless of their significance. Prioritized replay aims for prioritizing experience by replaying important transitions (i.e., higher rewards) more frequently, therefore, learning more efficiently [43]. Prioritized replay is controlled by

two common exponents [43]: α for proportional prioritization and β for importance sampling correction, which is usually annealed from $\beta_0 < 1$ toward 1 by end of training. In this work, according to the literature suggestions for prioritized replay, we set $\alpha = 0.6$, $\beta_0 = 0.4$. In summary, the major hyperparameters to tune for DQN are: $\{F_{train}$, γ, lr, C, B, N_{warmup}, $N_{anneal}\}$, and their reasonable values are listed in Table 1.

Table 1: RL/SO hyperparameters and their values or tuning range

Parameter	Method(s)	Range
γ	DQN/PPO	Fixed to 0.99
B	DQN/PPO	$2^n, n = 2, ..., 12$
lr	DQN/PPO	$10^{-5} - 10^{-3}$
F_{train}	DQN	1-16
C	DQN	100-10000
N_{warmup}	DQN	100-8000
N_{anneal}	DQN	[0.1-0.6] × Total TS
$\epsilon_{init}, \epsilon_{end}$	DQN	Fixed to 1.0, 0.02
α, β_0	DQN	Fixed to 0.6, 0.4
F_{train}	PPO	400-140000 (N=20 cores)
λ	PPO	0.9-1.0
$CLIP$	PPO	0.1-0.3
VF_{coef}	PPO	0.5-1.0
ENT_{coef}	PPO	0-0.03
opt_{epochs}	PPO	3-30
χ	GA/SA	0.005-0.2
N_{pop}	GA	20-80
CX	GA	0.3-0.9
MUT	GA	0.025-0.4
T_{max}	SA	1000-150000
T_{min}	SA	Fixed to 1
$Cooling$	SA	Fast, Boltzmann, Cauchy, Eqs.(10)-(12)

Proximal policy optimisation (PPO) belongs to the policy gradient (PG) family. PG family has been developed to preserve simplicity in RL implementation and less hyperparameter sensitivity than the Q-learning family. PG aims to train a policy that directly maps states to appropriate actions without an explicit Q function, by optimising the following loss function

$$L^{PG}(\theta) = E_t[log\ \pi_\theta(a_t|s_t)A_t], \tag{6}$$

where E_t is the expectation over a batch of transitions, π is the policy to be optimised which has weights θ. The policy π predicts action a given state s at time step t. The loss term to be optimised is embedded in A_t, the advantage estimate, which is controlled by γ (the discount factor) and λ (the bias-variance tradeoff parameter), see [44] for more details. For a special case when $\lambda = 1$, A_t can be written as [44]

$$A_t = \underbrace{\sum_{k=0}^{\infty} \gamma^k r_{t+k}}_{\text{Discounted Reward}} - \underbrace{V(s_t)}_{\text{Baseline (or VF) Estimate of Discounted Reward}}, \tag{7}$$

where r is the reward function, γ is the discount factor, and V is the baseline or value function (VF) estimate of the discounted reward. The advantage function estimates how much better the action took based on the baseline expectation of what would normally happen in this state, or in other words, it quantifies whether the action the agent took is better than expected ($A_t > 0$) or worse ($A_t < 0$) than the baseline estimate. Another benefit of

using the advantage estimate is that it helps inferring the change in VF without the need to explicitly calculating the real VF value. After vanilla PG, several enhancements have been conducted to improve PG for continuous monitoring and control, which we include here for brevity: deep deterministic policy gradient [45], Trust-Region Policy Optimisation [46], and finally reaching to PPO [23]. The PPO algorithm performs two major steps. In the first step, transitions (i.e., a sequence of states, rewards, and actions) are gathered based on several interactions of an initial (old) policy with the environment. The length of time during which transitions are collected before update is $F_{train} = NT$, which is the multiplication of number of parallel actors (i.e., N cores) times the time horizon of each actor, T in TS. For fixed N cores, F_{train} or T is tuned for better performance. In the second step, the policy is updated by optimising the neural network model, i.e., also known as the "surrogate". Likewise DQN, the second step of PPO is the deep learning part, which involves running gradient descent over the surrogate objective (L^{PPO}) with learning rate lr, mini-batch of size B, and for opt_{epochs} epochs. Notice that opt_{epochs} here is used to refer to the supervised learning epoch, which marks a complete forward and backward pass through a static dataset, in this case, a batch of transitions collected over a time horizon. This epoch definition is different than our definition of RL epoch in section 3.2. By defining the probability ratio, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, where θ_{old} are the policy parameters before the update, the surrogate loss/objective function for the clipped PPO (i.e., L^{CLIP}) can be written as [23]

$$L^{CLIP}(\theta) = E_t[\min(\underbrace{r_t(\theta)A_t}_{\text{Modified PG Objective}}, \underbrace{\text{clip}(r_t(\theta), 1-CLIP, 1+CLIP)A_t)}_{\text{Clipped Objective}})], \tag{8}$$

where $CLIP$ is the clip range. The first term in the min function is the modified PG objective after including the trust-region [46], 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 - $CLIP$, 1 + $CLIP$]. Two final terms are added to PPO [23] to ensure more stable training. The two terms come from the fact that since we are using a neural network architecture that shares parameters between the policy and the value function (which is embedded in the advantage estimate A_t), a loss function (L_t^{VF}) should be used that combines the policy surrogate error and the value function (VF) error terms. This effect is controlled by VF_{coef}. To augment this effect, an additional entropy term ($S[\pi_\theta]$), acting as a regularizer is incorporated, which is controlled by ENT_{coef}. The entropy term ensures sufficient exploration by preventing premature convergence of a "single" action probability to dominate the policy. By combining the three terms, the final PPO objective can be written as follows [23]

$$L^{PPO}(\theta) = E_t[\underbrace{L_t^{CLIP}(\theta)}_{\text{Clipping Term}} \underbrace{- VF_{coef} \cdot L_t^{VF}(\theta)}_{\text{Value Function Loss}} \underbrace{+ ENT_{coef} \cdot S[\pi_\theta](s_t)}_{\text{Entropy Term}}]. \tag{9}$$

According to the previous descriptions, PPO hyperparameters to be tuned are: {F_{train} or T, γ, λ, $CLIP$, VF_{coef}, ENT_{coef}, lr, B, opt_{epochs}}, and their reasonable values are listed in Table 1. Lastly, it is worth highlighting some major differences between DQN and PPO. During training, DQN updates the parameters/weights of the Q-function, which acts as a middleman to map states to actions, while PPO updates the policy π directly, that maps states to actions without Q-function in between. In addition, DQN is observed as sample efficient due to the

replay memory and the frequent training $F_{train}^{DQN} << F_{train}^{PPO}$. Although PPO is observed as sample inefficient since it discards previous experiences after updating the surrogate every F_{train} (i.e., no replay memory), this sacrifice allows simpler implementation and faster training, facilitates hyperparameter tuning, and eases parallel calculations. The PPO algorithm used in this work is shown in Algorithm 2.

Algorithm 2 Proximal Policy Optimisation

•Set hyperparameters: γ, λ, $CLIP$, VF_{coef}, ENT_{coef}, lr, B, opt_{epochs}
•Initialize policy $\pi_{\theta_{old}}$ with random weights
•Initialize r^{best}=$-\infty$
for Epoch i = 1 to $EPOCH$ **do**
    for Actor j = 1 to N **do**
        •Run policy $\pi_{\theta_{old}}$ in the environment for time horizon T
        •Compute rewards and advantage estimates A_1, ..., A_T
    •Optimise surrogate $L^{PPO}(\theta)$, Eq.(9), with respect to θ, with opt_{epochs}, lr, and mini-batch $B \leq NT$
    •Set $\theta_{old} = \theta$
    •Calculate mean reward $\bar{r}_i$ for the current epoch
    if $\bar{r}_i > r^{best}$ **then**
        •Save surrogate for the current epoch
        •Set $r^{best} = \bar{r}_i$

3.4. Stochastic Optimisation (SO)

SO in this work is used to refer to GA and SA, which act as baselines to PPO and DQN. GA [47] is inspired by the theory of natural evolution, where the fittest individuals are selected to produce offspring of the next generation. GA consists of five major phases: (1) generation of initial population with size N_{pop}, (2) fitness evaluation, (3) crossover, (4) mutation, and (5) selection. The initial population is a set of possible problem solutions, generated randomly from the search space, while the fitness function is the reward function described before for RL. The crossover operation selects two random individuals for mating with probability CX. In two-point crossover, two points are selected randomly from the parent genes. The genes between the two points are swapped between the parents, resulting in two offspring, each carrying genetic information from the parents. After new offspring are formed, some of the individuals may be selected for mutation with a small probability MUT. In combinatorial optimisation, the genes of the selected individual are mutated with probability χ with an integer uniformly drawn between the lower and upper bounds of each gene, where integers in this work are used to encode the enrichment/fuel types. The GA hyperparameters to tune are: $\{N_{pop}, MUT, CX, \chi\}$, and their reasonable values are listed in Table 1.

SA [48] is inspired from the concept of annealing in physics to reduce defects in crystals through heating followed by progressive and controlled cooling. In optimisation, SA is a combination of high climbing and pure random-walk to help us find an optimum solution. SA consists of five main steps: (1) generate a candidate solution, (2) evaluate its fitness, (3) generate a random neighbor solution and calculate its fitness, (4) compare the old and new fitness evaluations (i.e., ΔE increment), if better continue with the new solution, if worse, accept the old solution with probability $\alpha = exp^{-\Delta E/T}$, (5) repeat steps 3-4 until convergence. Here temperature T refers to the cooling temperature which is annealed between T_{max} and T_{min} over the annealing period of length N_{steps}. In this work, three commonly used temperature annealing schedules are considered when tuning SA, namely, fast, Boltzmann,

and Cauchy, described respectively as follows

$$T_{Fast} = T_{max} \cdot exp\left[\frac{-log(T_{max}/T_{min})k}{N_{steps}}\right], \tag{10}$$

$$T_{Boltzmann} = \frac{T_{max}}{log(k+1)}, \tag{11}$$

$$T_{Cauchy} = \frac{T_{max}}{k+1}, \tag{12}$$

where k is the current annealing step, which builds up from 1 to N_{steps}. Two modes of random-walk are used for SA. Dual swap randomly picks two attributes from the input individual and perturbs them between the lower and upper bounds of the corresponding attributes. The full swap is consistent with GA mutation, where all input attributes are subjected to swap with a small probability χ, where the selected attributes are replaced with an integer uniformly drawn between their lower and upper bounds. The SA hyperparameters to tune are: $\{\chi, T_{max}, Cooling\}$, and their reasonable values are listed in Table 1.

3.5. Coding and Implementation

For this work, stable-baselines-2.10.0 [49] is used here for DQN and PPO implementation. DEAP is an evolutionary computation framework providing efficient data structures for testing and constructing variety of SO algorithms. DEAP-1.3.0 [50] is used here for GA implementation. For SA, we used our own implementation. All algorithms, including our algorithmic modifications to match the methodology described in sections 3.2-3.4, are housed under an automated system. The user provides a configuration file that has all algorithms' hyperparameters as well as a physics-based environment following the structure described in section 3.2. An optimiser class is initialized based on the selected algorithms to use for optimisation, solving the problem in parallel mode.

Grid search is adopted in this work to tune the hyperparameters of the four investigated algorithms. Grid search has the advantages of easier implementation and parallelization and it is recommended when the analyst is aware of certain range to use for hyperparameters. Each hyperparameter is discretized into k nodes based on its sensitivity, where finer nodes are used for the sensitive parameters. All possible combinations of hyperparameters are evaluated and the best combination is selected for further analysis. Lastly, it is worth mentioning that we have used a workstation with 2 nodes (useful for CASMO-4), 8 cores each (a total of 32 threads), 128 GB RAM, and NVIDIA Quadro 4000 graphics card (useful for deep learning) to perform most of the calculations in this work.

4. Case Study 1: BWR 6x6 Fuel Assembly

The first case study forms a 6x6 BWR assembly with a total of 36 rods. The environment setup and reward shaping are described in the first subsection, while the results are presented and discussed in the second subsection.

4.1. Problem Setup

The assembly geometry is sketched in Figure 1(a), where the numbered rods on and below the diagonal line are included in the optimisation process. The exact number of possible permutations (i.e., full size of the search

space) can be estimated as $2^{21} = 2,097,152$. All these permutations were evaluated beforehand with the CASMO-4 code. Therefore, beside optimisation, this design serves as a proof-of-concept, as all methods are benchmarked by their ability to find feasible solutions and their closeness to the known global optima. Following the weighted scalarization in Eq.(1), the multi-objective function to be minimized can be expressed as

$$\min_{\vec{x}} \; F(\vec{x}) = \left|\frac{1.25 - k_\infty}{1.25}\right| + w_p \left|\frac{1.35 - PPF}{1.35}\right| + \left|\frac{2.3 - E}{2.3}\right|, \tag{13}$$

where $w_p = 1$ when $PPF > 1.35$ and $w_p = 0$ otherwise. The reason is that PPF is perceived as a threshold safety parameter to be maintained below a specific value (e.g., 1.35). Once this condition is satisfied, optimising PPF is no longer needed. Notice that this objective function is uniform as it normalizes all objectives to their constraints, and gives them equal weights. This is to reduce the bias that would come from well-engineered objective functions that rely on physical heuristics, which may overestimate the performance of RL/SO algorithms. The reward to be maximised for all RL/SO methods is nothing but the reciprocal of $F(x)$, i.e., $r(x) = 1/F(x)$.

The action space has a size of two: 1.87% and 2.53%, which are UO_2 enrichment. The state space is an array of size 21×2. The first column contains the current enrichment of each rod, while the second column is a vector of zeros, except 1 in the entry pointing to the next rod position to be visited. The RL game strategy for this problem can be summarised as follows:

1. Each episode starts with same initial pattern with fixed enrichment (1.5%) in all rods, which is different than the two possible fuel types (1.87% and 2.53%).
2. The agent randomly visits a rod location and takes an action using the step function defined in section 3.2.
3. Step 2 is repeated until all 21 rod locations are visited, if they do, CASMO-4 is activated and the reward is calculated using Eq.(13). Otherwise, the reward is zero.
4. If the new pattern has better reward than the previous (or initial) pattern, the new pattern is used as initial for the next round.
5. Steps 2-4 are repeated until 84 TS (4 rounds) are completed, which mark the end of episode. *The best pattern found from the 4 patterns is reported for that episode.*
6. Steps 1-5 are repeated until all training episodes or TS are completed.

Step 5 is equivalent to giving the player multiple chances to play the game (i.e., game lives), where here we give the agent four trials to find a good pattern before we end the episode. Notice that the neural network will be trained based on all time steps and patterns found, this step is only for logging purposes, as we noticed less uncertainty in RL predictions compared when terminating the episode early after 1 round (i.e., 21 TS). *Therefore, each RL episode in this problem has a length of 84 TS and involves 4 C4C.* The neural network structure for RL has a fixed depth, namely, 2 layers and 64 nodes per layer. For SO, the "fit" function handles the training for GA/SA, where the size of the input vector ($\vec{x}$) for this problem is 21, and each episode involves 1 C4C. The individual or input vector has the fuel type in each of the 21 rod locations. The individual is perturbed in each iteration through random-walk in SA and crossover/mutation in GA. Compared to RL, where the perturbation is local by changing the assembly one-at-a-time with C4C every 21 TS, the SO perturbation is global, where C4C is used directly after.

To finalize the setup, a total of 252000 TS are executed for RL, which correspond to 252000/84=3000 episodes. Every 30 episodes, which involve 120 C4Cs, are grouped into 1 epoch. Therefore, a total of 100 training epochs are analyzed for RL. In contrast, for SO, since there is no TS, every 120 episodes (or C4C) are treated as 1 epoch. As indicated in the last row of Table 2, the current setup ensures all algorithms have equal amount of interactions with CASMO-4 regardless of their type.

4.2. Results and Discussion

The results of brute-force search demonstrate that there are 59 global solutions with rewards equal to **287.5**. The global solutions exactly meet the constraints, k_∞ = 1.25000, PPF < 1.35, and E = 2.292% (this is the closest to 2.3% given the two discrete possibilities of enrichment). The reason for having multiple global solutions originates from the threshold condition imposed on PPF (< 1.35). Now, due to the inherent uncertainty in CASMO-4 (e.g., model-form, nuclear data uncertainty), like any other physical model, we decided to assume a ±0.0002 uncertainty over the k_∞ constraint, and use that margin to filter a set of "feasible" patterns. Consequently, in the subsequent discussions of this section, any pattern that has PPF < 1.35, E = 2.292, and $k_\infty \in [1.24080, 1.25020]$ will be considered a *feasible* pattern. The reward lower bound for these feasible patterns is **274.85**, which is referred to as the *desired reward*.

The hyperparameter tuning procedure is applied four times for the four algorithms in this case study. Since grid search is adopted, fixed grids are used for the hyperparameters without random sampling. A summary of the optimised hyperparameters is listed in Table 2 for all algorithms, as detailed results cannot be presented due to their voluminous size. Clearly, the number of RL hyperparameters is about twice as SO hyperparameters. For this specific problem, we noticed that DQN and PPO are more sensitive to hyperparameters compared to GA and SA. For instance, for GA, we noticed less sensitivity to changes in χ, CX, and MUT. On the other hand, DQN shows large sensitivity to N_{anneal}, C, B, and F_{train}, which all have wide range. Although PPO shows also sensitivity to multiple parameters, namely, F_{train}, λ, $CLIP$, and ENT_{coef}, the last three parameters have narrow range, and so easier to tune. Overall in this case study, we noticed that PPO is easier to tune than DQN, and both are more difficult to tune than GA and SA.

Table 2: Optimum training hyperparameters for all methods for the BWR 6x6 assembly

DQN		**PPO**		**GA**		**SA**	
Item	Value	Item	Value	Item	Value	Item	Value
F_{train} (TS)	4	F_{train} (TS)	2100	N_{gen}	200	N_{steps}	12000
γ	0.99	γ	0.99	N_{pop}	60	Swap Mode	Dual
lr	0.0005	lr	0.00025	χ	0.1	Cooling	Cauchy
ϵ_{init}, ϵ_{end}	1, 0.02	ENT_{coef}	0	CX	0.4	T_{min}	1
N_{anneal} (TS)	88200	VF_{coef}	0.5	MUT	0.1	T_{max}	150000
C (TS)	1000	λ	1	C4C	12000	C4C	12000
N_{warmup} (TS)	4200	B	4				
B	32	opt_{epochs}	15				
α	0.6	$CLIP$	0.3				
β_0	0.4	Total TS	252000				
Total TS	252000	Episodes	3000				
Episodes	3000	Epochs	100				
Epochs	100	C4C	12000				
C4C	12000						

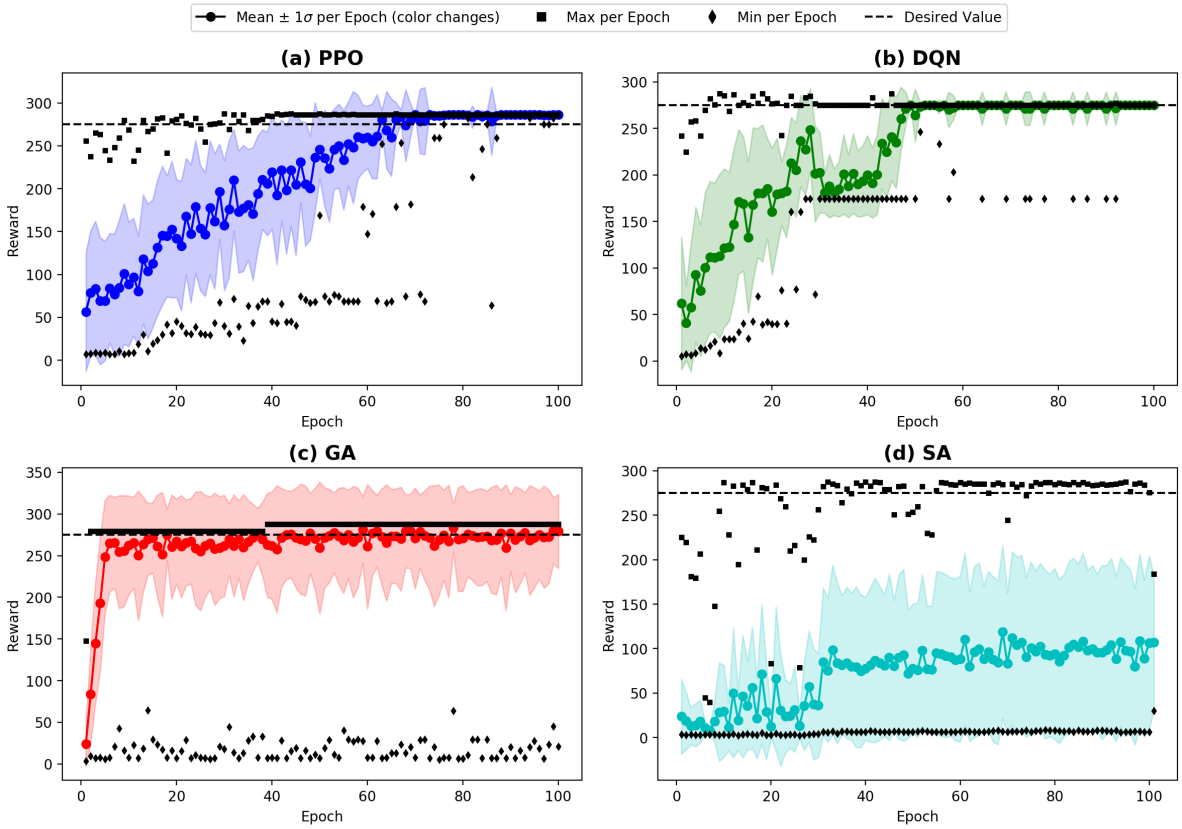

Figure 2: Reward convergence and statistical plots for the BWR 6x6 assembly: (a) PPO (1 epoch=30 episodes=120 C4C), (b) DQN (1 epoch=30 episodes=120 C4C), (c) GA (1 epoch=120 C4C), and (d) SA (1 epoch=120 C4C)

Based on the hyperparameters in Table 2, reward statistics with number of epochs for all algorithms are shown in Figure 2 along with the desired reward as reference. For RL, each epoch consists of 30 episodes and 120 C4C, while for SO, it consists of 1 episode and 120 C4C. At the beginning of training, the PPO agent collects initial samples to improve the knowledge-based system (KBS) performance, so this period is characterized by large uncertainty. After about 60 epochs, the PPO agent collected sufficient samples to leverage the mean reward and make the KBS less stochastic. For DQN, similar behavior is observed, earlier epochs are characterized by high epsilon values (i.e., ϵ_{init}), which involve taking random actions. This results in the large uncertainty in the reward early on. Once ϵ decreases to ϵ_{end} (after $N_{anneal} \sim 35$ epochs), the DQN agent prioritizes its main experience. The reward momentarily experiences a sharp decline after 35 epochs, followed by a consistent increase to converge to the desired reward after about 50 epochs. Both DQN and PPO are able to find many feasible patterns over the training period as can be inferred from the maximum reward per epoch.

GA converges very quickly compared to other methods, but seems to be less exploratory as the mean, max, and standard deviation of the reward remain somehow invariant over a large number of epochs. This implies that GA, with the selected hyperparameters, is more prone to hang in a local optima before escaping it to search into another region. This may be attributed to the relatively small MUT probability that restricts its exploration, however, using too large mutation rates are known to destruct the population and likely to lead to divergence. The opposite behaviour is seen for SA, which is much more exploratory than all other methods with our selected

hyperparameters, as can be told from the significant reward standard deviation and the large number of feasible patterns discovered (i.e., max reward per epoch). On the other hand, SA seems to converge slowly, and also unable to raise the mean reward to a satisfactory limit. Lastly, Figure 3 shows convergence plot of the optimised objectives (k_∞, E, PPF) for PPO. The PPO case is presented here for demonstration. Since other methods do not change the conclusion, they are not presented for brevity. Clearly, all objectives seem to converge in a harmonious manner shortly after 60 epochs, as can be observed from the strong agreement between each objective's max, min, and mean values as well as the negligible standard deviation when approaching the end of the training period.

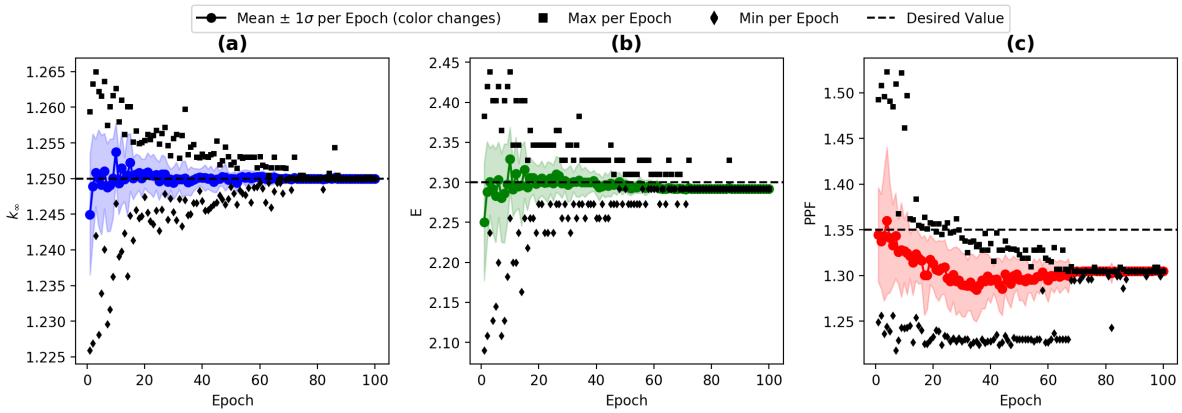

Figure 3: PPO's objective plots of (a) k_∞ (optimised to 1.25000), (b) average enrichment E (optimised to 2.3%), and (c) PPF (maintained below 1.35) with number of training epochs for the BWR 6x6 assembly (1 epoch=30 episodes=120 C4C)

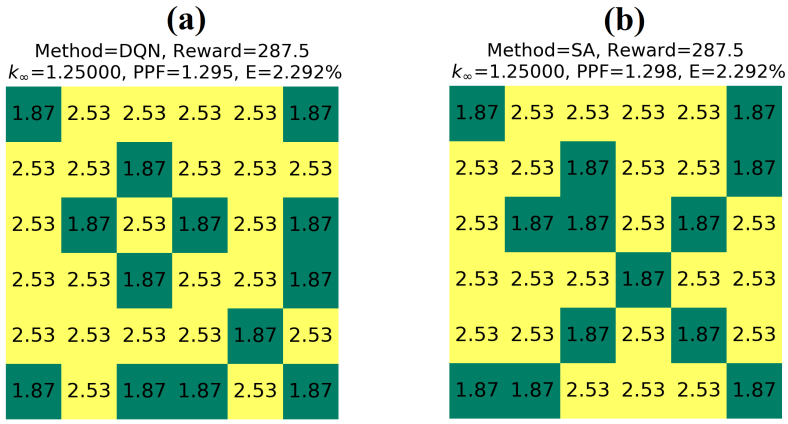

Figure 4: Sample plots of the best BWR 6x6 assembly configurations found by two different methods: (a) DQN (b) SA. Numbers in black represent UO_2 rods and their UO_2 enrichment

The summary of this case study is presented in Figure 4 and Table 3. Figure 4 illustrates two samples of the best patterns found by two selected methods during the optimisation search, and both are global optimum with reward of 287.5. The best patterns shown have low enrichment (1.87%) at the corners, meaning that RL is able to learn our physics heuristics on reducing the assembly PPF by favoring low enrichment at the corners, which helps reducing neutron leakage at the boundaries and improving overall performance. Table 3, which compares between methods in finding feasible patterns, reinforces previous findings. GA is able to find only 7 feasible patterns, but with very high efficiency (η) to find the first one. On the other hand, SA explores the space well by finding 178

feasible patterns even though it seems to be the less efficient among all. DQN and PPO are bounded by GA and SA in terms of the number of feasible patterns found as well as their efficiency needed during training. In addition, many studies prefer to compare RL performance after training by using the trained KBS for predictions (i.e., testing phase), since the training process is needed once. This comparison is given in the fourth column of Table 3. Obviously, if the trained KBS is used for optimisation, DQN and PPO can achieve a superior efficiency compared to SO algorithms in discovering the first feasible pattern. The KBS is able to consistently find a feasible pattern after 1 episode, which is equivalent to 4 C4C, and this should be expected from the converged reward shown before in Figure 2.

Table 3: Number of "distinguished" feasible patterns found for the BWR 6x6 assembly by different methods

Method	Number of Feasible Patterns	Efficiency η (Training)	Efficiency η (Testing)
DQN	27 (2 Global)	93.3%	99.9%
PPO	63 (2 Global)	91.6%	99.9%
GA	7 (1 Global)	98.6%	98.6%
SA	178 (3 Global)	90.8%	90.8%

Based on the observations of this case study and our approach to tune algorithm hyperparameters, key differences can be highlighted between the four algorithms in terms of KBS ability, exploratory nature, hyperparameter sensitivity, computational cost, and convergence. The next case study features a more complex problem that also represents a commercial design, where RL performance will be assessed.

5. Case Study 2: BWR 10x10 Fuel Assembly

In this section, the optimisation strategy is described first, which includes detailed reward/fitness shaping. Next, RL and SO performances in optimising the BWR 10x10 assembly are evaluated and discussed.

5.1. Problem Setup

The assembly geometry is sketched in Figure 1(b), where the numbered rods on and below the diagonal line are included in the optimisation process. The application of same problem setup as Case study 1 did not result in satisfactory results as the RL agent struggled to find feasible patterns, which was also the case for SO. Therefore, design heuristics based on expert input such as using low enrichment on assembly boundaries and allowing GAD rods only in the interior assembly rods, are used to restrict the design search space. Therefore, we utilize some of these heuristics here, which will also help reducing the dimensionality of the problem:

1. Heuristic 1: In Figure 1(b), the corner rods: 1, 5, and 40 have *identical* UO_2 enrichment with the following possibilities: 1.6%, 2.0%, 2.4%, 2.8%, and 3.2%. Corner rods usually have low UO_2 enrichment to reduce neutron leakage at the boundaries.
2. Heuristic 2: In Figure 1(b), the edged rods: 2, 4, and 39 neighboring the corner rods have *identical* UO_2 enrichment with the following possibilities: 2.8%, 3.2%, 3.6%, 4.0%, 4.4%, and 4.95%.
3. Heuristic 3: In Figure 1(b), the remaining edged rods: 3 and 21 have *identical* UO_2 enrichment with the following possibilities: 3.6%, 4.0%, 4.4%, and 4.95%.

4. Heuristic 4: In Figure 1(b), the interior rods between 6-38 except 21 can have both UO_2 and GAD rods and are not necessarily identical. The UO_2 possibilities are: 3.6%, 4.0%, 4.4%, and 4.95%, while the GAD possibilities are: 4.4% + 7% GAD, 4.95% + 7% GAD, 4.4% + 8% GAD, and 4.95% + 8% GAD.

Considering Heuristics 1-3, the exact number of combinations (outer frames) is $5 \times 6 \times 4 = 120$ outer frames. For each frame, there is a total of 8^{32} interior combinations based on Heuristic 4, leaving us with a total of $120 \times 8^{32} \approx 10^{31}$ possible patterns. Even after applying the above heuristics, problem dimensionality is still too high to find a global solution by brute-force. However, in most engineering applications, the designers are looking for a set of candidate solutions that meet certain constraints. In our case, we assume the following:

1. Constraint set 1 (CASMO-4 independent):

$$E \in [4.25\%, 4.36\%], \quad N_{GAD} \in [16, 18]. \tag{14}$$

2. Constraint set 2 (CASMO-4 dependent):

$$k_{\infty}^{max} \leq 1.11000, \quad PPF < c, \tag{15}$$

where c equals to 1.6 or 1.4. There are two primary motivations behind picking two constraints on PPF. First, it can provide more insights on relative performance of different tested algorithms since the optimization performance is problem specific. Second, having two constraints on PPF offers a compromising range between safety and economics. Assembly patterns whose $PPF < 1.6$ are expected to achieve higher CL than the patterns whose $PPF < 1.4$.

3. Objective (CASMO-4 dependent):

$$\max_{\vec{x}} \; F(\vec{x}) = CL. \tag{16}$$

The previous constraints are left up to the analyst to decide, where our choices here are based on various 10x10 designs available in the literature, including the one in Figure 1(b).

Our optimisation strategy is a two-step process: the KBS training step and the optimisation step. In the first step, a KBS trained by RL is used to satisfy constraint set 1, which does not require C4C. In the second step, after meeting constraint set 1, all methods are trained to meet the constraint set 2 and maximize CL, which require C4C. The pre-trained KBS is re-loaded and used directly for optimisation by continuing RL training under a different objective function. Each C4C for the 10x10 assembly takes about 1-1.5 minutes compared to 1-2 seconds for the 6x6 assembly, which is significantly larger.

Similar to Case Study 1, the agent randomly visits one position-at-a-time, picks a valid action, and moves to the next position. However, a major difference should be highlighted. Since the outer frames can be counted (i.e., 120 frames), they can be included explicitly without a need for random-walk to visit them. In other words, the positions 1, 2, 3, 4, 5, 21, 39, and 40 in Figure 1(b) are used as a fixed frame to the assembly based on their possible fuel types (see Heuristics 1-3). The agent starts with the first frame, visits the remaining 32 interior locations according to Heuristic 4, evaluate the pattern according to the rules below, and then moves to the next frame. This

process is repeated for all 120 frames, and this marks the end of one episode. Therefore, each episode consists of $120 \times 32 = 3840$ TS. This setup ensures the agent passes through all possible outer frames in every episode rather than randomly sampling them. Now, lets shape the reward function in form of game rules that the agent is going to learn, which are similar to the rules that nuclear engineers follow in practice:

1. Rule 1 (Excessive GAD actions): This rule penalizes the agent for excessively taking GAD actions during training (e.g., 4.4% + 7% GAD). This rule helps the agent to understand early on that large number of GAD rods is undesirable from physics perspective since they cause large power suppression by overly killing neutrons. If $N_{GAD} > 25$ and the agent attempts to take a GAD action, the agent is penalized with -1 reward, and the GAD action is replaced by a random UO_2 action (e.g., 3.6%, 4.0%, 4.4%, 4.95%)

$$V_1 = \sum_{t=1}^{TS} \mathbf{1}_{\{N_{GAD} > 25 \cap a_t \in G_{set}\}}, \tag{17}$$

   where $\mathbf{1}$ is the indicator function, a_t is the action taken at time step t, and $G_{set} = \{4.4\% + 7\%$ GAD, 4.95% + 7% GAD, 4.4% + 8% GAD, 4.95% + 8% GAD$\}$ is the set of possible GAD actions.

2. Rule 2 (N_{GAD} control): This rule asks the agent to learn how to optimise N_{GAD} between lower/upper bounds according to constraint set 1 in Eq.(14). Violation of Rule 2 is quantified by relative difference

$$V_2 = w \left| \frac{N_{GAD} - Bound}{Bound} \right| \%, \tag{18}$$

   where $Bound$ can be either 16 or 18 depending on N_{GAD} value and w is set to 1 for this rule.

3. Rule 3 (GAD positioning): This rule deals with the position of GAD rods, where two GAD rods are not allowed to sit next to each other either vertically or horizontally, diagonally is allowed. In addition, for the two rod positions 26 and 29, which neighbor the large water rods, GAD rods are not allowed. See Figure 1(b) for a sample of how GAD rods are positioned. From physics perspective, having multiple GAD rods next to each other causes large power suppression, which is reflected in having larger power production from the neighboring UO_2 rods to match the total assembly power, leading to a large and undesirable PPF, see Eq.(15). For each assembly pattern, number of GAD position violations ($N_{violate}$) are counted and multiplied by a factor of 10 to preserve a comparable numerical scale with other rules

$$V_3 = 10 * N_{violate}. \tag{19}$$

4. Rule 4: This rule asks the agent to learn how to optimise E between lower/upper bounds according to constraint set 1 in Eq.(14). Violation of Rule 4 (V_4) is quantified similar to V_2 by relative difference where N_{GAD}, $Bound$, and w in Eq. (18) are replaced by E, 4.25%/4.36%, and 50, respectively. Using larger w for V_4 is done to preserve comparable numerical scale as other rules.

5. Total Reward: Reward is the sum of the 4 rules' violation. A negative sign is used to convert violation to reward

$$Reward_1 = -(V_1 + V_2 + V_3 + V_4 + p), \tag{20}$$

where p is a penalty factor between [0,400] to handle a possible caveat that could result from Rule 2. The p value is determined on-the-fly based on V_2 value, which proved to improve the learning stability.

The term ***candidate pattern*** is used to refer to a pattern that satisfies constraint set 1 in Eq.(14). For the KBS step, a total of 6,912,000 TS are executed for RL, which correspond to $\frac{6912000}{120\times32}$ = 1800 episodes. Every 30 episodes are grouped into 1 epoch, and so a total of 60 training epochs are analyzed for RL. In contrast, for SO, since there is no TS, every 3600 episodes (or C4C) are treated as 1 epoch. The individual for SO has a size of 51, each corresponds to the fuel type in each rod location. The individual is perturbed globally using random-walk in SA and mutation/crossover in GA. As indicated in the last row of Table 4, the current setup ensures all algorithms have equal amount of interactions with CASMO-4 regardless of their type.

After mastering the game by RL, a KBS is generated which is capable of producing candidate patterns frequently. The KBS is used for step 2, the optimisation step, where if a candidate pattern is identified during search, it is evaluated by CASMO-4 according to the following reward (i.e., maximizing CL):

$$Reward_2 = CL - p_{k_\infty} - p_{ppf}, \tag{21}$$

where p_{k_∞} and p_{ppf} are set to zero if constraint set 2 in Eq.(15) is met. Otherwise, the two penalties are set to relative difference as in Eq.(18) where w = 1, and N_{GAD} and $Bound$ are replaced by their k_∞/PPF correspondences. Similarly for the optimisation step, the term ***feasible pattern*** is used to refer to a pattern that satisfies constraint set 1 and set 2 in Eqs.(14)-(15), where the objective is to maximize the number of feasible patterns discovered and as a result CL. The epoch setup is similar to the KBS step, but with less number of environment calls due to computational expense, see the items with asterisk in Table 4. For both steps, KBS and optimisation, the neural network structure has a fixed depth, namely, 2 layers and 64 nodes per layer.

In summary, for RL, the agent is firstly trained to match constraint set 1, leading to a trained model/KBS. Then the model is re-loaded and optimisation is continued toward meeting constraint set 2 and maximizing CL. On the other hand, for SO, since no model is trained, GA and SA have to match constraints set 1, set 2, and maximize CL simultaneously.

Table 4: Optimum training hyperparameters for all methods for the BWR 10x10 assembly

PPO		***GA***		***SA***	
Item	Value	Item	Value	Item	Value
F_{train} (TS)	40000	N_{gen}	4320/720*	N_{steps}	216000/36000*
γ	0.99	N_{pop}	50	Swap Mode	Full
lr	0.00025	χ	0.025	Cooling	Cauchy
ENT_{coef}	0	CX	0.8	χ	0.01
VF_{coef}	1	MUT	0.25	T_{min}	1
λ	0.9	C4C	216000/36000*	T_{max}	10000
B	4			C4C	216000/36000*
opt_{epochs}	15				
$CLIP$	0.2				
Total TS	6,912,000/1,152,000*				
Episodes	1800/300*				
Epochs	60/30*				
C4C	216000/36000*				

*Items used in the optimisation step, section 5.3.

5.2. Step 1: KBS Results

For conciseness of the results, we restricted our RL training in this case study to PPO since DQN demonstrated similar performance. Therefore, the term KBS is directly referring to a PPO agent in this case study. The hyperparameters of PPO, GA, and SA are tuned with grid search and listed in Table 4. As mentioned before, for the KBS step (step 1), matching constraint set 1 does not need CASMO-4, so PPO training does not take significant time. The training results are plotted in Figure 5. The agent is able to learn Rule 1 very quickly as V_1 takes few epochs to converge to zero despite attempting about 100 invalid GAD actions in the first epoch. For Rule 2, we can notice that V_2 increases in earlier epochs and the total reward as well. Here, where the penalty p interrupts and corrects the behavior, causing V_2 to drop significantly after the third epoch, and to converge after about 30 epochs. V_3 and V_4 follow similar trends with sharp decrease early on, followed by slow exponential decay, where Rule 4 seems to be the most difficult rule to learn as about 40 epochs are needed to converge. This plot shows how the KBS improved itself over time. The first 10 epochs are characterized by large amount of mistakes as can be inferred from the small reward in Figure 5(a). Once the agent masters the game well, a surge in candidate patterns appears, reaching as many as 90 different candidate patterns per episode, which is set as the desired limit. Consequently, the KBS can generate on average a total of $90 \times 30 = 2700$ candidate patterns over 1 epoch, assuming each epoch has 30 episodes.

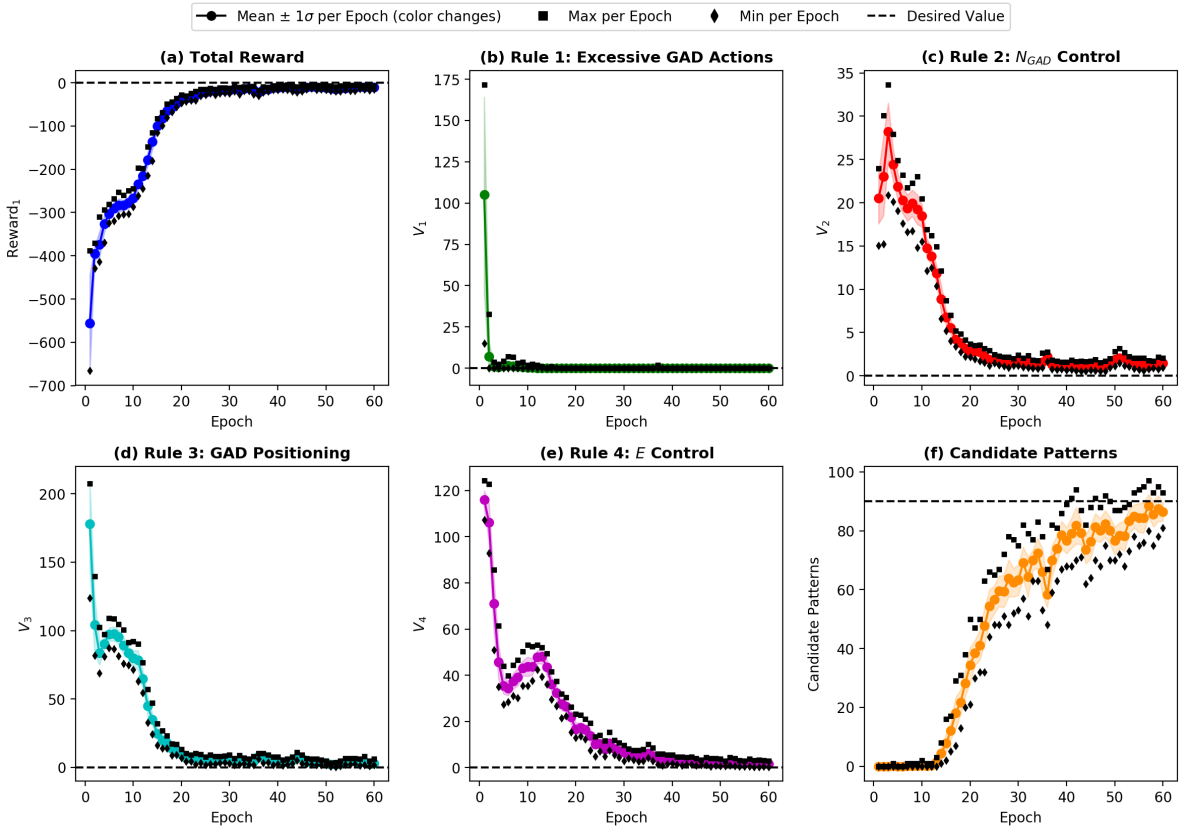

Figure 5: KBS/PPO's convergence plots for (a) total reward, (b)-(e) reward components (game rules), and (f) number of candidate patterns. Each epoch consists of 30 episodes

Comparison of PPO and GA/SA in terms of their ability to find candidate patterns is shown in Table 5. To

ensure one-to-one comparison between the three methods, PPO, GA, and SA are subjected to same amount of interactions with the environment as indicated in the last row of Table 4. In addition, the constraints, rules, and reward shaping described before are preserved across the methods. Compared to Case Study 1, larger CX and MUT probabilities for GA seem to improve the exploratory behavior of GA, while retaining stable performance. For SA, we adopt the full swap approach with small probability of $\chi = 0.01$ for each rod position. The results in Table 5 clearly show that RL performance is superior to GA/SA, generating more than twice of candidate patterns as GA/SA. This clearly implies that the gradient-based nature of RL and its ability to retain expert knowledge in form of game rules (V1-V4) significantly helped in exploring the search space better than SO. The agent was able to build an efficient model to escape most of the infeasible regions, leading to more comprehensive and efficient search for candidate patterns. It is worth highlighting again that these candidate patterns in Table 5 are not necessarily feasible (i.e., may not meet constraint set 2). This fact will be confirmed in the next subsection.

Table 5: Number of "distinguished" candidate patterns found for the BWR 10x10 assembly by different methods after 60 epochs of training. Based on these results, PPO-based KBS is used for the optimisation step, section 5.3

Method	Number of Candidate Patterns*
RL (PPO)	80578
GA	35625
SA	31788

*Candidate patterns refer to the patterns that satisfy constraint set 1 in Eq.(14).

5.3. Step 2: Optimisation Results

The pre-trained KBS model from step 1 is re-loaded and used for optimisation in continual form via PPO based on the reward of Eq.(21). Hyperparameters for all methods are similar to the ones listed in Table 4, where the items marked with asterisk are exclusive for this optimisation section. All algorithms have been exposed to equal 36,000 C4C.

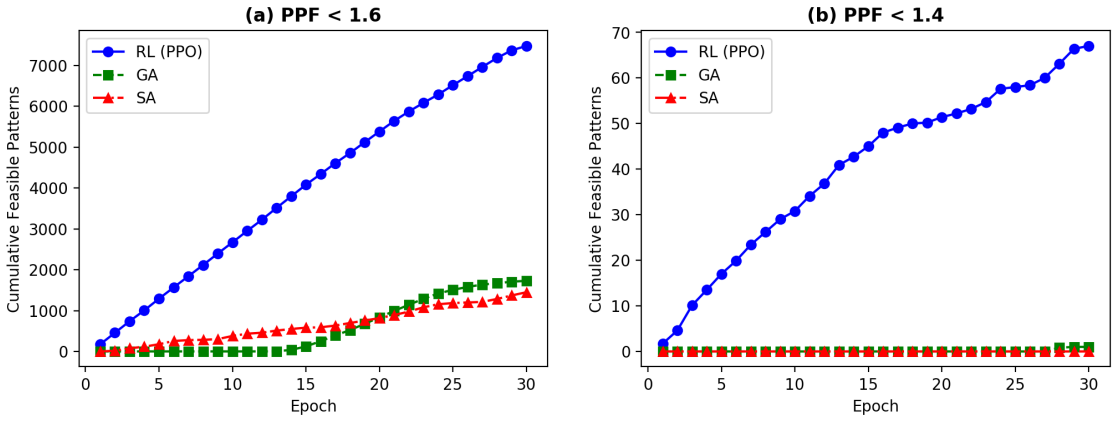

Figure 6: Total number of feasible patterns found by different methods for the 10x10 optimisation case as a function of epochs (1 epoch = 1200 C4C). Feasible patterns refer to the patterns that satisfy constraint set 1 and set 2 in Eqs.(14)-(15)

As shown in Eq.(15), we present results for two PPF cases, PPF < 1.6 (less-constrained) and PPF < 1.4. The

results of the optimisation search for these two cases are shown in Figure 6 and Tables 6-7. Similar to Case Study 1, two evaluation metrics are used to compare the algorithms in Tables 6-7: (1) number of feasible patterns discovered and (2) algorithm efficiency (η). Unlike Table 5 for Case Study 1, the testing efficiency is not presented in Tables 6-7 for RL, since we started the optimisation step with a well-trained KBS that can already generate candidate patterns. Since any candidate pattern could become feasible after executing C4C, it is likely to find a feasible pattern early on in training, which justifies the RL incredible training efficiency in Tables 6-7.

Clearly in Figure 6(a) and Table 6, RL seems to outperform SO by big margin in terms of number of feasible patterns discovered for the PPF < 1.6 case, finding more than 7000 feasible patterns. RL found 4 times more feasible patterns than GA, and 5 times more than SA. Also, RL provides outstanding efficiency in reaching the feasible region with outstanding 99.9% compared to 96% and 90% for SA and GA, respectively. Here, it is worth highlighting that this efficiency improvement, even though it sounds small numerically, is significant computationally. RL saved about 1380 C4Cs compared to SA, which correspond to about 29 hours of serial computational time, assuming 1.25 minutes on average per C4C. And this efficiency enhancement is even larger when compared to GA.

Table 6: Number of "distinguished" feasible patterns with $PPF < 1.6$ found for the BWR 10x10 assembly by different methods

Method	Number of Feasible Patterns*	Efficiency η (Training)
RL (PPO)	7525	99.9%
GA	1752	90.2%
SA	1477	96.0%

*Feasible patterns refer to the patterns that satisfy constraint set 1 and set 2 in Eqs.(14)-(15) with PPF < 1.6.

Table 7: Number of "distinguished" feasible patterns with $PPF < 1.4$ found for the BWR 10x10 assembly by different methods

Method	Number of Feasible Patterns*	Efficiency η (Training)
RL (PPO)	67	99.3%
GA	1	9.4%
SA	0	0%

*Feasible patterns refer to the patterns that satisfy constraint set 1 and set 2 in Eqs.(14)-(15) with PPF < 1.4.

Moving to a more constrained optimisation with PPF < 1.4 in Figure 6(b) and Table 6, we can notice that SO algorithms really struggle to resolve the search space, with only one feasible pattern found by GA, and nothing by SA. RL excels for this confined case, finding larger number of feasible patterns with very good efficiency. Therefore, we can notice the explicit sensitivity of GA/SA performance in finding candidate/feasible patterns, when moving from Table 5 (KBS step) to Table 7 (a constrained optimisation step), which indeed demonstrated the promise of RL in these types of problems. After all, the practical application of the proposed algorithms is limited by the computational resources at the user's disposal, especially the computational time needed by the code to evaluate each pattern.

For the practical results, we plot two patterns featuring the best (highest CL) in Figure 7 for PPF < 1.6 and < 1.4, based on RL search. The final patterns respect the heuristics, game rules, and constraints described before,

yielding a best CL of 1478 days with PPF < 1.6 and 1462 days with PPF < 1.4. In summary, the implication of this case study to non-nuclear applications features using RL as a tool to embed expert knowledge by learning how to match certain rules and constraints of the problem, which in turn leads to a better optimisation search than random-walk or stochastic optimisation.

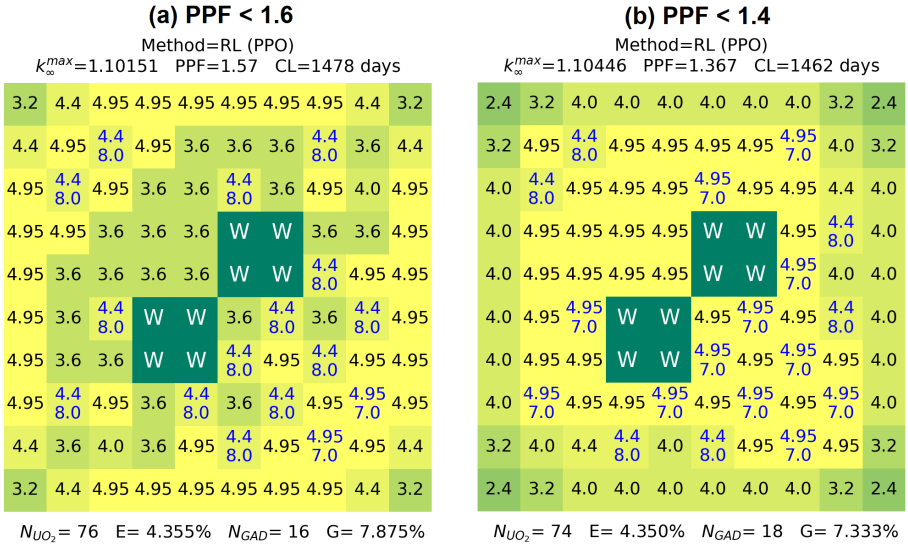

Figure 7: Sample plots of the best BWR 10x10 assembly configurations found for cases with (a) $PPF < 1.6$ and (b) $PPF < 1.4$. Numbers in black represent UO_2 rods and their UO_2 enrichment. Numbers in blue represent GAD rods, where the upper value represents the UO_2 enrichment and lower value represents the GAD enrichment

6. Closing Remarks

The potential efficiency gains in nuclear fuel cost encourage fuel designers to solve high dimensional, and expensive combinatorial optimisation problems. Fuel optimization is still mainly tackled by expert judgement and classical stochastic optimisation (SO) algorithms. In this work, we propose a reinforcement learning (RL) physics-based optimisation methodology based on deep RL to improve upon SO performance under a robust and licensed nuclear code. The methodology utilizes deep Q learning (DQN) and proximal policy optimisation (PPO), where genetic algorithm (GA) and simulated annealing (SA) serve as baselines for comparison.

The algorithms are applied first to a small/low-dimensional BWR 6x6 assembly ($\sim 2 \times 10^6$ combinations). All algorithms are validated by their ability to find a set of pre-evaluated global solutions from brute-force search. All algorithms were able to find at least 1 global solution under similar reward shaping and computational cost constraints. RL algorithms (PPO/DQN) have large number of hyperparameters and experience hyperparameter sensitivity. Although hyperparameter sensitivity is a common issue of deep RL, most of the observations found are still subjective to the problem analyzed and the approach used to tune hyperparameters. On the other hand, RL algorithms offer knowledge-based system (KBS) and good exploratory capabilities to efficiently explore the search space. For SO algorithms (GA/SA), GA seems to converge faster than RL, but prone to fall into local optima and hence does not explore the space well. SA has good exploratory capability, but its inherent random-walk nature could slowdown the search. Overall, the results reveal that DQN and PPO performances are bounded by GA and

SA, leading to a conclusion that SO seems to be adequate to solve this low-dimensional problem. Nevertheless, when using the pre-trained KBS from PPO and DQN, an outstanding testing efficiency can be achieved by RL compared to SO.

In the second case study, a larger BWR 10x10 assembly, which is more complex, high-dimensional ($\sim 10^{31}$ combinations), and expensive, is optimised using RL and SO. As the problem complexity grows, it became clear that expert input based on nuclear engineering principles in setting the rules of the game are critical for a successful RL strategy. RL using PPO demonstrated superior performance in training a KBS to match certain rules and constraints such as controlling assembly enrichment and learning how to place GAD rods in appropriate positions. RL demonstrated superior performance to SO by exploring the search space better and increasing the algorithm speed and efficiency. Overall, RL successfully found large number of competitive assembly patterns for the BWR 10x10 assembly.

While this work clearly demonstrated the value of RL for a simplified BWR 2D lattice optimization, its application to more realistic 2D assembly as well as full core optimization for both PWRs and BWRs is currently being explored. Particularly, the move from case study 1 to 2 required a different problem setup to help RL efficiency. This means as different fuel design optimisation problems are tackled, nuclear engineering input and hyperparameter tuning are needed to obtain optimal results. Also, the performance of traditional SO algorithms will change with new problems. In other words, there are almost never a free-lunch with any optimization technique [51], and problem-dependent reward shaping is always needed. Furthermore, studies on combining RL and SO techniques to better inform each other are worthy of investigation. Lastly, the readers are cautioned on use of black-box surrogates in place of licensed methodology to accelerate the optimization process without fully capturing and understanding the resulting uncertainties on the performance.

Acknowledgment

This work is sponsored by Exelon Corporation, a nuclear electric power generation company, under the award (40008739).

Conflict of Interests

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

References

[1] B. Korte, J. Vygen, B. Korte, J. Vygen, Combinatorial optimization, Vol. 2, Springer, 2012.

[2] 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.

[3] J. Francois, C. Martın-del Campo, R. François, L. Morales, A practical optimization procedure for radial bwr fuel lattice design using tabu search with a multiobjective function, Annals of Nuclear Energy 30 (12) (2003) 1213–1229.

[4] 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).

[5] F. Verhagen, M. Van der Schaar, W. De Kruijf, T. Van de Wetering, R. Jones, Rosa, a utility tool for loading pattern optimization, in: Proc. of the ANS Topical Meeting–Advances in Nuclear Fuel Management II, Vol. 1, 1997, pp. 8–31.

[6] M. A. Nasr, M. Zangian, M. Abbasi, A. Zolfaghari, Neutronic and thermal-hydraulic aspects of loading pattern optimization during the first cycle of vver-1000 reactor using polar bear optimization method, Annals of Nuclear Energy 133 (2019) 538–548.

[7] A. Zameer, M. Muneeb, S. M. Mirza, M. A. Z. Raja, Fractional-order particle swarm based multi-objective pwr core loading pattern optimization, Annals of Nuclear Energy 135 (2020) 106982.

[8] 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.

[9] P. Barbero, G. Bidoglio, M. Bresesti, Post-irradiation analysis of the gundremmingen bwr spent fuel, Tech. rep., EUR–6301, Commission of the European Communities (1979).

[10] F. Michel-Sendis, I. Gauld, J. Martinez, C. Alejano, M. Bossant, D. Boulanger, O. Cabellos, V. Chrapciak, J. Conde, I. Fast, et al., Sfcompo-2.0: An oecd nea database of spent nuclear fuel isotopic assays, reactor design specifications, and operating data, Annals of Nuclear Energy 110 (2017) 779–788.

[11] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, Dynamic programming and optimal control, Vol. 1, Athena scientific Belmont, MA, 1995.

[12] T. A. Feo, M. G. Resende, Greedy randomized adaptive search procedures, Journal of global optimization 6 (2) (1995) 109–133.

[13] S. Nahar, S. Sahni, E. Shragowitz, Simulated annealing and combinatorial optimization, in: 23rd ACM/IEEE Design Automation Conference, IEEE, 1986, pp. 293–299.

[14] H. Mühlenbein, M. Gorges-Schleuter, O. Krämer, Evolution algorithms in combinatorial optimization, Parallel computing 7 (1) (1988) 65–85.

[15] J.-F. Cordeau, G. Laporte, A. Mercier, A unified tabu search heuristic for vehicle routing problems with time windows, Journal of the Operational research society 52 (8) (2001) 928–936.

[16] K. A. Smith, Neural networks for combinatorial optimization: a review of more than a decade of research, INFORMS Journal on Computing 11 (1) (1999) 15–34.

[17] M. Gendreau, J.-Y. Potvin, Metaheuristics in combinatorial optimization, Annals of Operations Research 140 (1) (2005) 189–213.

[18] R. J. Williams, J. Peng, Reinforcement learning algorithms as function optimizers, in: Proceedings of the International Joint Conference on Neural Networks, Washington DC, Vol. 2, 1989, pp. 89–95.

[19] R. J. Williams, J. Peng, Function optimization using connectionist reinforcement learning algorithms, Connection Science 3 (3) (1991) 241–268.

[20] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: a methodological tour d'horizon, arXiv preprint arXiv:1811.06128.

[21] I. Bello, H. Pham, Q. V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, arXiv preprint arXiv:1611.09940.

[22] 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.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347.

[24] L. M. Gambardella, M. Dorigo, Ant-q: A reinforcement learning approach to the traveling salesman problem, in: Machine Learning Proceedings 1995, Elsevier, 1995, pp. 252–260.

[25] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, in: Advances in Neural Information Processing Systems, 2015, pp. 2692–2700.

[26] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, in: Advances in Neural Information Processing Systems, 2017, pp. 6348–6358.

[27] M. Guo, Y. Liu, J. Malec, A new q-learning algorithm based on the metropolis criterion, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 34 (5) (2004) 2140–2143.

[28] A. Berny, Selection and reinforcement learning for combinatorial optimization, in: International Conference on Parallel Problem Solving from Nature, Springer, 2000, pp. 601–610.

[29] M. Nazari, A. Oroojlooy, L. Snyder, M. Takác, Reinforcement learning for solving the vehicle routing problem, in: Advances in Neural Information Processing Systems, 2018, pp. 9839–9849.

[30] G. T. Parks, Multiobjective pressurized water reactor reload core design by nondominated genetic algorithm search, Nuclear Science and Engineering 124 (1) (1996) 178–187.

[31] D. Babazadeh, M. Boroushaki, C. Lucas, Optimization of fuel core loading pattern design in a vver nuclear power reactors using particle swarm optimization (pso), Annals of nuclear energy 36 (7) (2009) 923–930.

[32] N. Poursalehi, A. Zolfaghari, A. Minuchehr, H. Moghaddam, Continuous firefly algorithm applied to pwr core pattern enhancement, Nuclear Engineering and Design 258 (2013) 107–115.

[33] J. J. Ortiz, I. Requena, Using a multi-state recurrent neural network to optimize loading patterns in bwrs, Annals of Nuclear Energy 31 (7) (2004) 789–803.

[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] 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.

[36] S. Tayefi, A. Pazirandeh, Using hopfield neural network to optimize fuel rod loading patterns in vver/1000 reactor by applying axial variation of enrichment distribution, Applied Soft Computing 21 (2014) 501–508.

[37] 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.

[38] A. Charles, G. Parks, Application of differential evolution algorithms to multi-objective optimization problems in mixed-oxide fuel assembly design, Annals of Nuclear Energy 127 (2019) 165–177.

[39] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym, arXiv preprint arXiv:1606.01540.

[40] C. J. Watkins, P. Dayan, Q-learning, Machine learning 8 (3-4) (1992) 279–292.

[41] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Thirtieth AAAI conference on artificial intelligence, 2016.

[42] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, N. De Freitas, Dueling network architectures for deep reinforcement learning, arXiv preprint arXiv:1511.06581.

[43] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, arXiv preprint arXiv:1511.05952.

[44] J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel, High-dimensional continuous control using generalized advantage estimation, arXiv preprint arXiv:1506.02438.

[45] 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.

[46] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International conference on machine learning, 2015, pp. 1889–1897.

[47] D. Whitley, A genetic algorithm tutorial, Statistics and computing 4 (2) (1994) 65–85.

[48] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by simulated annealing, science 220 (4598) (1983) 671–680.

[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.

[50] 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.

[51] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, IEEE transactions on evolutionary computation 1 (1) (1997) 67–82.

*Bibliography
Click here to view linked References

References

[1] B. Korte, J. Vygen, B. Korte, J. Vygen, Combinatorial optimization, Vol. 2, Springer, 2012.

[2] 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.

[3] J. Francois, C. Martın-del Campo, R. Fran¸cois, L. Morales, A practical optimization procedure for radial bwr fuel lattice design using tabu search with a multiobjective function, Annals of Nuclear Energy 30 (12) (2003) 1213–1229.

[4] 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).

[5] F. Verhagen, M. Van der Schaar, W. De Kruijf, T. Van de Wetering, R. Jones, Rosa, a utility tool for loading pattern optimization, in: Proc. of the ANS Topical Meeting–Advances in Nuclear Fuel Management II, Vol. 1, 1997, pp. 8–31.

[6] M. A. Nasr, M. Zangian, M. Abbasi, A. Zolfaghari, Neutronic and thermal-hydraulic aspects of loading pattern optimization during the first cycle of vver-1000 reactor using polar bear optimization method, Annals of Nuclear Energy 133 (2019) 538–548.

[7] A. Zameer, M. Muneeb, S. M. Mirza, M. A. Z. Raja, Fractional-order particle swarm based multi-objective pwr core loading pattern optimization, Annals of Nuclear Energy 135 (2020) 106982.

[8] M. Edenius, K. Ekberg, B. H. Forss´en, D. Knott, Casmo-4, a fuel assembly burnup program, user's manual, Studsvik0SOA-9501, Studsvik of America, Inc.

[9] P. Barbero, G. Bidoglio, M. Bresesti, Post-irradiation analysis of the gundremmingen bwr spent fuel, Tech. rep., EUR–6301, Commission of the European Communities (1979).

[10] F. Michel-Sendis, I. Gauld, J. Martinez, C. Alejano, M. Bossant, D. Boulanger, O. Cabellos, V. Chrapciak, J. Conde, I. Fast, et al., Sfcompo-2.0: An oecd nea database of spent nuclear fuel isotopic assays, reactor design specifications, and operating data, Annals of Nuclear Energy 110 (2017) 779–788.

[11] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, Dynamic programming and optimal control, Vol. 1, Athena scientific Belmont, MA, 1995.

[12] T. A. Feo, M. G. Resende, Greedy randomized adaptive search procedures, Journal of global optimization 6 (2) (1995) 109–133.

[13] S. Nahar, S. Sahni, E. Shragowitz, Simulated annealing and combinatorial optimization, in: 23rd ACM/IEEE Design Automation Conference, IEEE, 1986, pp. 293–299.

[14] H. M¨uhlenbein, M. Gorges-Schleuter, O. Kr¨amer, Evolution algorithms in combinatorial optimization, Parallel computing 7 (1) (1988) 65–85.

[15] J.-F. Cordeau, G. Laporte, A. Mercier, A unified tabu search heuristic for vehicle routing problems with time windows, Journal of the Operational research society 52 (8) (2001) 928–936.

[16] K. A. Smith, Neural networks for combinatorial optimization: a review of more than a decade of research, INFORMS Journal on Computing 11 (1) (1999) 15–34.

[17] M. Gendreau, J.-Y. Potvin, Metaheuristics in combinatorial optimization, Annals of Operations Research 140 (1) (2005) 189–213.

[18] R. J. Williams, J. Peng, Reinforcement learning algorithms as function optimizers, in: Proceedings of the International Joint Conference on Neural Networks, Washington DC, Vol. 2, 1989, pp. 89–95.

[19] R. J. Williams, J. Peng, Function optimization using connectionist reinforcement learning algorithms, Connection Science 3 (3) (1991) 241–268.

[20] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: a methodological tour d'horizon, arXiv preprint arXiv:1811.06128.

[21] I. Bello, H. Pham, Q. V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, arXiv preprint arXiv:1611.09940.

[22] 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.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347.

[24] L. M. Gambardella, M. Dorigo, Ant-q: A reinforcement learning approach to the traveling salesman problem, in: Machine Learning Proceedings 1995, Elsevier, 1995, pp. 252–260.

[25] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, in: Advances in Neural Information Processing Systems, 2015, pp. 2692– 2700.

[26] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, in: Advances in Neural Information Processing Systems, 2017, pp. 6348–6358.

[27] M. Guo, Y. Liu, J. Malec, A new q-learning algorithm based on the metropolis criterion, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 34 (5) (2004) 2140–2143.

[28] A. Berny, Selection and reinforcement learning for combinatorial optimization, in: International Conference on Parallel Problem Solving from Nature, Springer, 2000, pp. 601–610.

[29] M. Nazari, A. Oroojlooy, L. Snyder, M. Tak´ac, Reinforcement learning for solving the vehicle routing problem, in: Advances in Neural Information Processing Systems, 2018, pp. 9839–9849.

[30] G. T. Parks, Multiobjective pressurized water reactor reload core design by nondominated genetic algorithm search, Nuclear Science and Engineering 124 (1) (1996) 178–187.

[31] D. Babazadeh, M. Boroushaki, C. Lucas, Optimization of fuel core loading pattern design in a vver nuclear power reactors using particle swarm optimization (pso), Annals of nuclear energy 36 (7) (2009) 923–930.

[32] N. Poursalehi, A. Zolfaghari, A. Minuchehr, H. Moghaddam, Continuous firefly algorithm applied to pwr core pattern enhancement, Nuclear Engineering and Design 258 (2013) 107–115.

[33] J. J. Ortiz, I. Requena, Using a multi-state recurrent neural network to optimize loading patterns in bwrs, Annals of Nuclear Energy 31 (7) (2004) 789–803.

[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] 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.

[36] S. Tayefi, A. Pazirandeh, Using hopfield neural network to optimize fuel rod loading patterns in vver/1000 reactor by applying axial variation of enrichment distribution, Applied Soft Computing 21 (2014) 501–508.

[37] 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.

[38] A. Charles, G. Parks, Application of differential evolution algorithms to multi-objective optimization problems in mixed-oxide fuel assembly design, Annals of Nuclear Energy 127 (2019) 165–177.

[39] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym, arXiv preprint arXiv:1606.01540.

[40] C. J. Watkins, P. Dayan, Q-learning, Machine learning 8 (3-4) (1992) 279–292.

[41] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Thirtieth AAAI conference on artificial intelligence, 2016.

[42] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, N. De Freitas, Dueling network architectures for deep reinforcement learning, arXiv preprint arXiv:1511.06581.

[43] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, arXiv preprint arXiv:1511.05952.

[44] J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel, High-dimensional continuous control using generalized advantage estimation, arXiv preprint arXiv:1506.02438.

[45] 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.

[46] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International conference on machine learning, 2015, pp. 1889–1897.

[47] D. Whitley, A genetic algorithm tutorial, Statistics and computing 4 (2) (1994) 65–85.

[48] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by simulated annealing, science 220 (4598) (1983) 671–680.

[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.

[50] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, C. Gagn´e, DEAP: Evolutionary algorithms made easy, Journal of Machine Learning Research 13 (2012) 2171–2175.

[51] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, IEEE transactions on evolutionary computation 1 (1)(1997) 67–82.

*Declaration of Interest Statement

Conflict of Interest

The authors declare that there are no conflicts of interest regarding the work done in this paper.