Hyperparameter Optimization for Multi-Objective Reinforcement Learning

Daniel Gareev
University of Luxembourg
Email: daniel.gareev.001@student.uni.lu

Elion Hashani
University of Luxembourg
Email: elion.hashani.001@student.uni.lu

Florian Felten
University of Luxembourg
Email: florian.felten@uni.lu

Abstract—**In this paper, we present an automated approach to hyperparameter tuning in multi-objective reinforcement learning algorithms to save researchers time and ensure fair comparisons. Our methodology allows for optimisation and post-analysis of results, identifying the most promising algorithms and parameters. We also discuss potential research opportunities associated with generalising hyperparameters for improved benchmarking and algorithm development.**

1. Introduction

Reinforcement learning is an aspect of machine learning in which agents learn how to make decisions in an environment by interacting with it and receiving feedback through rewards or punishments. In contrast to single-objective reinforcement learning, multi-objective reinforcement learning (MORL) aims to balance multiple, potentially conflicting reward signals.

Hyperparameter optimisation is an essential task in the development of machine learning algorithms, including MORL algorithms, as it plays a critical role in determining the performance of the algorithms. However, manual methods for optimising hyperparameters can be computationally expensive and time consuming, whereas automated approaches can save researchers time and resources.

In this paper, we show an automated approach to hyperparameter tuning that aims to save researchers time and ensure a fair comparison between different algorithms. Our approach systematically tunes parameters using an unbiased methodology, allowing the identification of optimal settings for each algorithm. We also show how the results can be post-analyzed to identify the most promising algorithms and parameter combinations for future research.

In addition, we discuss the potential research opportunities associated with generalising hyperparameters across different algorithms and environments. These studies have the potential to improve benchmarking processes, facilitate algorithm development, and ultimately increase the efficiency and effectiveness of reinforcement learning solutions.

2. Related Work

2.1. Multi-Objective RL

We introduce the multi-objective sequential decision problem and base it on the simpler multi-objective Markov decision process (MOMDP) instead of more complex models. Focusing on single-objective MOMDPs, it is defined as [1]:

- **State space (S)**: The set of all possible states in the environment.
- **Action space (A)**: The set of all possible actions that an agent can take.
- **Transition function (T)**: A probabilistic function, $T : S \times A \times S \to [0, 1]$, which describes the probability of transitioning from one state to another given a specific action.
- **Discount factor (γ)**: A value in the range $[0, 1)$, representing the degree to which future rewards are discounted.
- **Initial state distribution (μ)**: A probability distribution, $(\mu) : S \to [0, 1]$, over initial states.
- **Vector-valued reward function (R)**: A function, $R : S \times A \times S \to R^d$, defining immediate rewards for each of d ≥ 2 objectives.

The main difference from a single-objective MDP is the vector-valued reward function, where the vector length is equal to the number of objectives [1]. Both state and action sets can be discrete, finite or infinite, with real-world problems often having continuous variables or large discrete state spaces [1]. Infinite state and action spaces make the problem more difficult and require function approximators for policy and value estimation [1].

There are two main categories of existing MORL algorithms: single-policy methods and multiple-policy methods. Single-policy methods aim to find the best policy for a given preference among the objectives, using different forms of preference functions such as non-linear ones [2]. However, these methods are not effective when the preferences are unknown. On the other hand, multi-policy approaches learn a set of policies to obtain the approximate Pareto front of optimal solutions [2]. This is usually done by running a single-policy method over different preferences or by simultaneously learning the optimal manifold over a set of preference [2]. However, these methods are difficult to scale up to high-dimensional preference spaces and cannot be easily adapted to new preferences during testing [2].

2.2. Hyperparameter Optimization

Hyperparameter Optimization (HPO) is the process of selecting a set of hyperparameters that yield optimal performance for a given algorithm. The primary objective function in this case is to maximize selected performance indicators, which are discussed in the next section.

Consider A as a multi-objective reinforcement learning (MORL) algorithm with a configuration space consisting of total hyperparameters Φ. Representing A with its hyperparameters ϕ as A^{ϕ}, where $\phi \in \Phi$, note that the hyperparameter space Φ can contain both discrete and continuous dimensions.

In order to evaluate the performance of a MORL algorithm, we introduce the concept of Pareto front. The Pareto front, $P(Y)$, is a non-dominated set of solutions representing optimal trade-offs between conflicting objectives. It can be formally defined as [3]:

$$P(Y) = \{y' \in Y : \{y'' \in Y : y'' \succ y', y' \neq y''\} = \emptyset\}$$

Where Y is the feasible set of criterion vectors in $\mathbb{R}^m$ such that $Y = \{y \in \mathbb{R}^m : y = f(x), x \in X\}$, and y'' strictly dominates y' if $y'' \succ y'$.

For a given MORL algorithm D, the ultimate goal is to identify the most appropriate configuration ϕ^* such that

$$\phi^* = \underset{\phi \in \Phi}{\operatorname{argmax}} \; HV(P(Y(A^{\phi}(D))))$$

Where:

- HV is the hypervolume metric
- D is the MORL problem instance
- ϕ^* denotes the most appropriate hyperparameter configuration
- $P(Y(A^{\phi}(D)))$ is the Pareto front of the MORL algorithm with hyperparameters ϕ on the problem instance D.

By incorporating the Pareto front, this HPO formulation allows for a more comprehensive evaluation of the algorithm's performance, taking into account not only single indicators but also the trade-offs between different objectives. Consequently, it enables a better understanding of the algorithm's behavior in multi-objective optimization scenarios and supports more effective tuning of hyperparameters.

HPO uses different techniques depending on the complexity of the problem. For smaller search spaces, grid search [4] and random search [5] are common. Grid search evaluates all hyperparameter combinations, but faces dimensionality problems and unstable performance as the space grows. In contrast, random search samples configurations at random, which allows for easier parallelization but can lead to suboptimal results.

For non-convex or non-differentiable problems, global optimization algorithms such as Bayesian Optimization (BO) [6] are preferred. BO combines a probabilistic surrogate model that fits samples and learns the relationship between hyperparameters and performance, with a learning function that balances exploration and exploitation. This approach results in efficient optimisation of the hyperparameter space, as BO-based approaches provide guided, efficient exploration compared to grid search and random search. In this paper, we use BO as it is a widely used approach for HPO and a single evaluation of the a set of hyperparameters has a high computational cost.

2.3. Performance Metrics

Performance evaluation is essential in multi-objective optimisation, where the output of the optimisation process consists of sets of non-dominated solutions. Quality indicators can be divided into three main categories: Convergence-Based Indicators, Diversity-Based Indicators and Hybrid Indicators [7]

2.3.1. Convergence-based indicators. Convergence-based indicators measure the closeness of the approximated front to the true Pareto front. Examples include contribution, generational distance (I_{GD}), and the ϵ indicator. These indicators depend on either reference sets or optimal sets and involve different complexities and parameters.

In this paper, we use the I_{GD} metric, which is defined as [7]:

$$I^{t}_{\text{GD}}(A, R) = \frac{\left(\sum_{u \in A} \left(\min_{v \in R} \|F(u) - F(v)\|^2\right)^{1/2}\right.}{|R|}$$

I_{GD} is a metric that calculates the average distance between approximated set A and reference set R [7]. Using Euclidean distance, the pairwise minimum distances are averaged to compute the distance between the sets at each iteration t. If A is a subset of R, I_{GD} is 0.

2.3.2. Diversity-Based Indicators. Diversity-based indicators evaluate the distribution of solutions in the approximated front [7]. Common diversity-based indicators are Spread, Extent and Binary Entropy (B. Entropy) [7]. They generally do not have monotonic properties because they focus on the spread of solutions.

2.3.3. Hybrid Indicators. Hybrid indicators combine convergence and diversity indicators to measure both the proximity to the true Pareto front and the distribution of solutions [7]. Examples of hybrid indicators are hypervolume and R-metrics [7]. These indicators involve reference points or sets, have monotonic properties, and target specific optimisation parameters.

In this paper, we use the hypervolume metric. Hypervolume is a quality indicator measuring the volume of objective space weakly dominated by approximation A [7].

It requires a reference point Z_{ref} and can be binary, using reference set Z_N^* to represent hypervolume dominated by Z_N^* but not A [7]. The closer the binary measure is to 0, the better. Z_N^* can be obtained from the optimal Pareto front or union of all fronts. Unary hypervolume is strictly monotonic but computationally expensive with exponential complexity in the number of objectives.

3. Hyperparameter Optimisation for MORL

In this section we show the proposed method for HPO for MORL algorithms. First, we consider HPO as a sequential decision process. Then, we illustrate the details of the optimisation method, including the choice of mertics for the objective function and the surrogate model. Finally, we discuss the generalisation of the optimal hyperparameters across environments.

3.1. Overview

As a performance measure, we use the hypervolume metric, which is a widely used performance measure in the context of MORL algorithms. We use this metric because it effectively captures both the convergence and diversity properties of the approximated Pareto front. This metric measures the quality of non-dominated solutions in terms of their proximity to the true Pareto front as well as their spread across the objective space. It therefore provides a comprehensive assessment of the trade-offs between different objectives. Moreover, since hypervolume has desirable mathematical properties such as monotonicity and convergence, it provides a robust basis for comparing and optimising the performance of hyperparameter optimisation tasks.

3.2. System Design

We improve the conventional methodology of HPO for MORL algorithms by introducing an automated search algorithm to identify the optimal hyperparameters.

To start the hyperparameter search, it is necessary to configure the ranges of hyperparameters, initiate the training process and obtain an optimal set of policies, known as the Pareto front. Subsequently, a performance indicator, specifically the hypervolume metric, is used to derive a single objective function. This procedure involves the continuous selection of hyperparameters and the repetition of the training process with the aim of maximising the hypervolume metric.

Furthermore, we evaluate the hyperparameters on n seeds to ensure the consistency of the algorithm's performance over multiple runs, as it may be affected by the random initialisation of the algorithm. The mean hypervolume metric, derived from the training results on different seeds, is computed and recorded in Weights & Biases (W&B) [8]. The hypervolume metric is taken at the last time step of the training process for each agent.

For the computation, we use a Univesity of Luxembourg's High-Performance Computing (HPC) nodes [9] with 12 cores and 16 GB of RAM. The node is also equipped with a V-100 GPU.

In summary, our system consists of an HPC server for computation, a W&B server for data storage, and a Bayesian optimisation framework to manage the optimisation process. This is illustrated in Figure 1.

3.3. Implementation

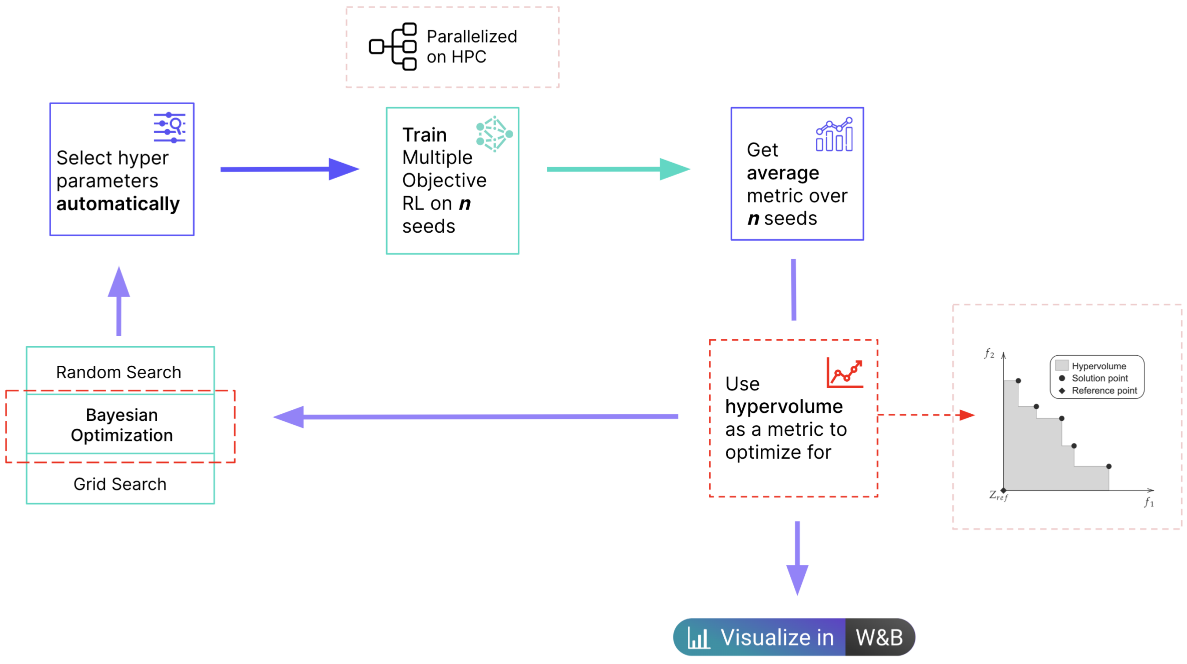
3.3.1. MORL Baselines Library. In this paper we use the MORL Baselines library [10] as the basis for our experiments. The library contains several MORL algorithms, such as Envelope Q-Learning [2] and PGMORL [11], and a set of benchmark environments from MO-Gymnasium [12]. The library also includes a set of metrics for evaluating the performance of MORL algorithms, such as the hypervolume metric, which measures the volume of the Pareto front. The only way MORL baselines differ from the standard Gymnasium API is that it returns a numpy array as a reward. Single and multi-policy algorithms are implemented under both SER and ESR criteria. Performance is automatically reported in W&B dashboards.

3.3.2. Code Description. We introduce a new script that performs an automated hyperparameter search for any algorithm and environment in the library. The script runs a series of experiments, collects performance metrics, and logs the results in W&B. We use the W&B Sweep feature to run multiple experiments in parallel and log the results in a single dashboard [13]. W&B Sweep combines hyperparameter search with interactive, visual experiment tracking. It also supports various methods such as Bayesian optimization for hyperparameter exploration. The script is designed to be easily extensible to new algorithms and environments.

Training is done with multiple seeds in parallel, using the `ProcessPoolExecutor` [14] to run each agent with a different seed at the same time. By running training on a range of seeds, the script accounts for variability in the learning process and provides a more comprehensive evaluation of the algorithms' performance. The average hypervolume metric obtained from the results of training on different seeds is computed and logged in W&B.

3.3.3. Components Description. The main components of the script are as follows

- **Parse Arguments**: Parse command line arguments for the algorithm, environment ID, reference point, W&B entity, project name, number of seeds and training hyperparameters.
- **Worker Classes**: Define classes to handle worker setup and results, including `WorkerInitData` and `WorkerDoneData`.
- **Train Function**: Implement a train function to instantiate the selected algorithm, train the agent and return the hypervolume metric.



Parallelized on HPC
Select hyper parameters **automatically**
Train Multiple Objective RL on ***n*** seeds
Get **average** metric over ***n*** seeds
Random Search
Bayesian Optimization
Grid Search
Use **hypervolume** as a metric to optimize for
Hypervolume
Solution point
Reference point
f_2
f_1
Z_{ref}
Visualize in W&B

Figure 1. System Design

- **Main Function**: Initialise W&B, create a process pool of workers, submit tasks to the workers, collect results, compute the average hypervolume, and log the metrics to W&B.
- **Sweep Setup and Execution**: Load the sweep configuration, set up the sweep with W&B, and run the sweep agent using the main function.

The script allows users to easily perform a sweep of MORL algorithms and environments, explore different hyperparameters and log the results to W&B for further analysis.

The code can be found in the `experiments/hyperparameter_search` directory of the MORL Baselines repository [10].

3.3.4. Usage Example. For an example of a command to run the hyperparameter search script, see the listing 1. This command will run the envelope algorithm on the `minecart-v0` environment, using the reference point $(-1, -1, -200)$, 100 sweeps, 3 seeds, and $100,000$ total timesteps.

```
python launch_sweep.py \
--algo envelope \
--env-id minecart-v0 \
--ref-point -1 -1 -200 \
--sweep-count 100 \
--num-seeds 3 \
--seed 0 \
--train-hyperparams total_timesteps:100000
```

Listing 1. An Example Launcher

3.3.5. Configuration Files. The configs with the ranges of hyperparameters for the sweep should be placed in the `configs` directory with the appropriate algorithm name, (e.g. `envelope.yaml`). The listing 2 shows an example of a configuration file. The configuration file follows the W&B sweep configuration format [15].

```
method: bayes
metric:
  goal: maximize
  name: hypervolume
parameters:
  learning_rate:
    distribution: uniform
    min: 1e-4
    max: 1e-3
  initial_epsilon:
    distribution: uniform
    min: 0.5
    max: 1
  final_epsilon:
    distribution: uniform
    min: 0.02
    max: 0.15
  epsilon_decay_steps:
    distribution: uniform
    min: 80000
    max: 100000
```

Listing 2. An Example Configuration File

3.3.6. Results Visualization. By default, W&B automatically generates a parallel coordinates plot, a parameter importance plot and a scatter plot. The parallel coordinates plot provides an overview of the relationships between numerous hyperparameters and model metrics in a single visualisation. The scatter plot allows comparison of the

W&B runs produced during the sweep. Finally, the parameter importance plot highlights the hyperparameters that were most predictive and highly correlated with favourable metric values. An example of the visualisation is shown in Figure 2.

4. Experiments

In this section we present the results of the experiments. First, we describe the experimental setup. Then we present the results of the experiments and discuss the generalisation of the tuned hyperparameters across environments.

4.1. Evaluation Metrics

We use generational distance (I_{GD}) and hypervolume as performance indicators for the experiments. Note that the calculation of I_{GD} is only possible if the true Pareto front is known. Therefore, it is only possible to compute it for some specific environments.

4.2. Experiment Setup

To perform the experiments, we use the MORL Baselines library [10] and the hyperparameter tuning script described in the previous section.

We compare the performance of the algorithms with the default hyperparameters and the optimal hyperparameters. The default hyperparameters are the hyperparameters used in the MORL Baselines library [10]. The optimal hyperparameters are the hyperparameters obtained from the HPO experiments. We run the benchmarks for the same number of timesteps for both the default and optimized hyperparameters.

4.3. Evaluation

In this section, we present the findings of our HPO experiments. We first discuss the results obtained for the Envelope Q Learning (EQL) algorithm [2], followed by those for the PGMORL algorithm [11].

4.3.1. Envelope Q Learning (EQL). We assess the performance of our proposed method on the Envelope Q Learning (EQL) algorithm [2] using three benchmark environments: `minecart-v0`, `mountainCar-v0` and `reacher-v4`. Note that the known Pareto front is only available for the `minecart-v0` environment, so we can only compute the I_{GD} metric for that environment.

For `minecart-v0` environment, the reference point is set to $(-1, -1, -200)$. For `mountaincar-v0` environment, it is set to $(-200, -200, -200)$. For `reacher-v4` environemnt, the reference point is $(-50, -50, -50, -50)$. These reference points were chosen based on the previous benchmarks experiments for the MORL Baselines library.

The results of the runs with the default hyperparameters are presented in Table 1, Table 2 and Table 3 for each of the environments. Note that *Seed* in the table corresponds to a specific seed selected for each training run for a single agent.

TABLE 1. Default Parameters Results for Envelope Q Learning Algorithm in `minecart-v0` Environment

Seed	HV	I_{GD}
0	199	0.7589
1	199	0.7589
2	199	0.7589

TABLE 2. Default Parameters Results for Envelope Q Learning Algorithm in `mountaincar-v0` Environment

Seed	HV
0	4535918.71
1	4535918.71
2	4535918.71

TABLE 3. Default Parameters Results for Envelope Q Learning Algorithm in `reacher-v4` Environment

Seed	HV
0	21563117.41
1	26511148.65
2	22827728.79

In each environment, we conducted 100 sweeps with 3 seeds per sweep. A sweep corresponds to a single evaluation of selected hyperparameters. Within an evaluation, the metrics are averaged over the 3 seeds. To verify the consistency of the algorithm's performance across multiple runs, we replicated each HPO experiment 3 times.

The results of these experiments are presented in Table 4, Table 5 and Table 6 and Table 7 for each of the environments. Note that *Seed* in the table corresponds to a specific seed selected for each HPO experiment.

TABLE 4. Hyperparameter Search Results for Envelope Q Learning Algorithm in `minecart-v0` Environment

Seed	**Mean** HV	**Std** HV	**Min** HV	**Max** HV
0	216.02	42.81	455.68	394.67
1	224.49	60.63	192.96	455.68
2	207.89	32.28	192.78	388.07

5. Discussion

In general, results of runs with the default hyperparameters are relatively consistent across the different seeds used. On the other hand, the metrics obtained using HPO showed noticeable differences in performance compared to the default setups.

In the '`minecart-v0`' environment, it can be observed that the HPO settings generally yield better results in terms of Mean HV compared to the default parameters. Furthermore, for the metric 'I_{GD}', HPO mostly resulted in better performance compared to the default configuration.

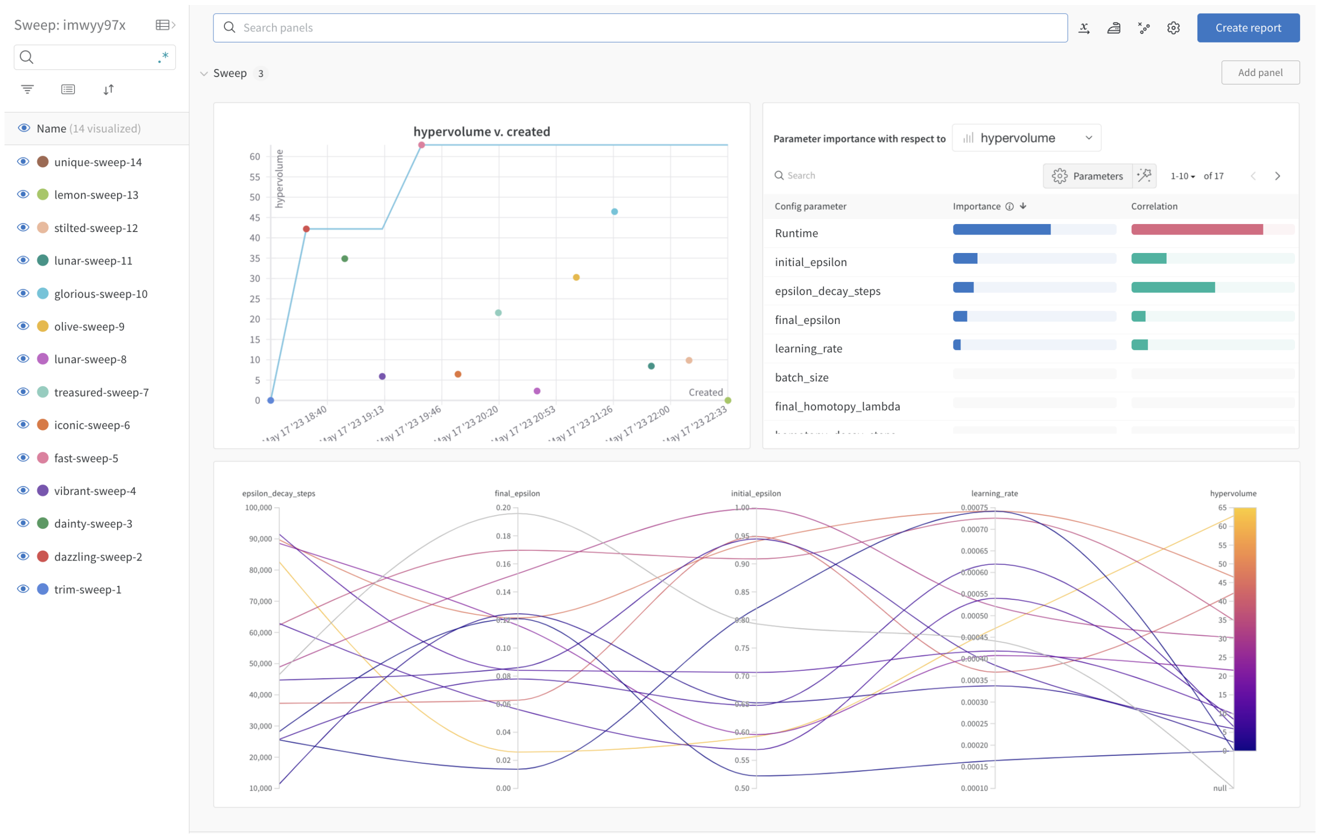


Figure 2. Visualization of the Hyperparameter Search

TABLE 5. Hyperparameter Search Results for Envelope Q Learning Algorithm in `minecart-v0` Environment

Seed	Mean I_{GD}	Min I_{GD}	Max I_{GD}
0	0.795	0.129	10.21
1	0.731	0.197	9.18
2	0.833	0.212	9.62

TABLE 6. Hyperparameter Search Results for Envelope Q Learning Algorithm in `reacher-v4` Environment

Seed	Mean HV	Std HV	Min HV	Max HV
0	22440442.86	3126455.11	13327130.07	27118468.27
1	21044373.41	3665919.58	13286871.35	26108032.32
2	22083021.92	2826130.11	13260016.07	26064285.51

TABLE 7. Hyperparameter Search Results for Envelope Q Learning Algorithm in `mountainar-v0` Environment

Seed	Mean HV	Std HV	Min HV	Max HV
0	6020282.71	114294.41	5047004.30	6059188.03
1	6033215.23	19833.85	5913826.93	6076917.34
2	6018293.05	25785.34	5875690.61	6122383.70

For the ‘`reacher-v4`‘ environment, the results show that HPO managed to achieve higher ‘Mean HV‘ values and lower standard deviations than its default counterpart. This indicates a more stable performance of the algorithm when using tuned hyperparameters.

Finally, in the ‘`mountaincar-v0`‘ environment, HPO also delivered improved results in terms of mean and standard deviation. This suggests that the optimization process was successful in finding better performing hyperparameter combinations, resulting in improved algorithmic competence.

Based on our research and findings, we can conclude that it may be feasible to apply optimal hyperparameters across different environments within the same environment family.

The HPO has several advantages, including

1) Reducing the need to perform a full hyperparameter search for each new environment, saving researchers time and computational resources.
2) Facilitates better comparison of algorithm performance in different environments because the tuning process is automated and consistent.
3) Inform future research on the selection of appropriate hyperparameters for use with a particular family of environments based on the performance metrics.

6. Limitations

We have identified the following limitations:

1) **Dependence on parameter ranges**: The success of the hyperparameter search depends heavily on prior knowledge of the appropriate ranges for each parameter. It is therefore essential to have a thorough understanding of the algorithms and the implications of selecting specific ranges for each parameter.
2) **Limited algorithm coverage**: The current study focused on a limited set of algorithms, which may not necessarily generalise well to all MORL algorithms.
3) **Determining the optimal number of sweeps**: Finding the right balance between computational resources and statistical power is challenging, and the current maximum of 100 sweeps may not capture the entire hyperparameter space.
4) **Sensitivity of metrics**: Our results are sensitive to the choice of metrics used for evaluation; using alternative evaluation metrics may lead to different conclusions about hyperparameter generalisation.
5) **More runs to reach statistical significance**: To obtain more statistically significant results for hyperparameter search, it is necessary to run more HPO experiments.

7. Future Work

As future work, we suggest the following improvements:

1) **Extending hyperparameter configuration files and ranges**: A full understanding of all the algorithms and the implications of selecting specific ranges for each parameter is essential for extending the configuration files and parameter ranges.
2) **Increase the number of sweeps**: To obtain more statistically significant results for hyperparameter tuning, we should increase the number of sweeps beyond the current maximum of 100.
3) **Evaluate hypervolume over multiple time steps**: In order to eliminate transient fluctuations that affect comparisons, it would be useful to compute the hypervolume metric over multiple timesteps rather than only at the last timestep.
4) **Use other metrics**: Explore other relevant performance metrics such as convergence, diversity or hybrid metrics.
5) **Implement early stopping**: The introduction of early stopping based on specific thresholds could further improve the speed of the search process.
6) **Test more environmental classes**: Research how well a set of hyperparameters performs in different environment classes. The performance metrics should be compared for similar environments when a set of optimal hyperparameters is applied.
7) **Validate the optimal hyperparameters**: Run the algorithms for higher number of timesteps and compare the performance directly with the default parameters. The results could be visualized with a line plot.
8) **Analyze hyperparameter values and range selection**: Investigate different hyperparameter value choices and selection procedures strategies.

8. Conclusion

In summary, the presented approach to automating hyperparameter optimization offers several key benefits to researchers. First, it saves valuable time by eliminating manual trial and error in algorithm configuration. Secondly, it enables fair comparison between algorithms by ensuring that each is optimized systematically and consistently.

By using this automated tuning process, we can effectively conduct post-analysis to identify the most promising algorithms and their respective parameter values. This information serves as a solid basis for further research and development of reinforcement learning methods.

Finally, investigating the generalisation of hyperparameters across different algorithms and environments remains a promising area of research.

Acknowledgment

We would like to thank Florian Felten for drafting the research proposals, actively participating in shaping the vision of the project and providing constant feedback on the progress of the project. We would also like to thank the University of Luxembourg for providing access to the High-Performance Computing (HPC) server.

References

[1] C. F. Hayes, R. Rădulescu, E. Bargiacchi, J. Källström, M. Macfarlane, M. Reymond, T. Verstraeten, L. M. Zintgraf, R. Dazeley, F. Heintz, *et al.*, "A practical guide to multi-objective reinforcement learning and planning," *Autonomous Agents and Multi-Agent Systems*, vol. 36, no. 1, p. 26, 2022.

[2] R. Yang, X. Sun, and K. Narasimhan, "A generalized algorithm for multi-objective reinforcement learning and policy adaptation," *Advances in neural information processing systems*, vol. 32, 2019.

[3] Wikipedia contributors, "Pareto front — Wikipedia, the free encyclopedia," 2023.

[4] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[5] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization.," *Journal of machine learning research*, vol. 13, no. 2, 2012.

[6] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pp. 507–523, Springer, 2011.

[7] E.-G. Talbi, *Metaheuristics: from design to implementation.* John Wiley & Sons, 2009.

[8] "Weights and biases." https://wandb.ai/site.

[9] S. Varrette, H. Cartiaux, S. Peter, E. Kieffer, T. Valette, and A. Olloh, "Management of an Academic HPC & Research Computing Facility: The ULHPC Experience 2.0," in *Proc. of the 6th ACM High Performance Computing and Cluster Technologies Conf. (HPCCT 2022)*, (Fuzhou, China), Association for Computing Machinery (ACM), July 2022.

[10] F. Felten and L. N. Alegre, "Morl-baselines: Multi-objective reinforcement learning algorithms implementations." https://github.com/LucasAlegre/morl-baselines, 2022.

[11] J. Xu, Y. Tian, P. Ma, D. Rus, S. Sueda, and W. Matusik, "Prediction-guided multi-objective reinforcement learning for continuous robot control," in *International conference on machine learning*, pp. 10607–10616, PMLR, 2020.

[12] L. N. Alegre, F. Felten, E.-G. Talbi, G. Danoy, A. Nowé, A. L. C. Bazzan, and B. C. da Silva, "MO-Gym: A library of multi-objective reinforcement learning environments," in *Proceedings of the 34th Benelux Conference on Artificial Intelligence BNAIC/Benelearn 2022*, 2022.

[13] "Tune hyperparameters with sweeps." https://docs.wandb.ai/guides/sweeps.

[14] "concurrent.futures — launching parallel tasks." https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.ProcessPoolExecutor.

[15] "Define sweep configuration for hyperparameter tuning." https://docs.wandb.ai/guides/sweeps/define-sweep-configuration.