MULTI-OBJECTIVE EVOLUTION FOR GENERALIZABLE POLICY GRADIENT ALGORITHMS

Juan Jose Garau-Luis, Yingjie Miao, John D. Co-Reyes, Aaron Parisi,
Jie Tan, Esteban Real, Aleksandra Faust
Google Brain
{garau,yingjiemiao,jcoreyes,aarontp,jietan,ereal,faust}@google.com

ABSTRACT

Performance, generalizability, and stability are three Reinforcement Learning (RL) challenges relevant to many practical applications in which they present themselves in combination. Still, state-of-the-art RL algorithms fall short when addressing multiple RL objectives simultaneously and current human-driven design practices might not be well-suited for multi-objective RL. In this paper we present MetaPG, an evolutionary method that discovers new RL algorithms represented as graphs, following a multi-objective search criteria in which different RL objectives are encoded in separate fitness scores. Our findings show that, when using a graph-based implementation of Soft Actor-Critic (SAC) to initialize the population, our method is able to find new algorithms that improve upon SAC's performance and generalizability by 3% and 17%, respectively, and reduce instability up to 65%. In addition, we analyze the graph structure of the best algorithms in the population and offer an interpretation of specific elements that help trading performance for generalizability and vice versa. We validate our findings in three different continuous control tasks: RWRL Cartpole, RWRL Walker, and Gym Pendulum.

1 INTRODUCTION

Many Reinforcement Learning (RL) practitioners working on real-world problems often deal with three challenges that complicate the road to deployment: performance measured by high returns, generalizability, and stability. On the one hand, deployed policies should perform well according to specific standards depending on the environment. On the other hand, the same standards should also be met in scenarios that have not been part of the training process, i.e., policies should generalize zero-shot too. In addition, RL algorithms should be stable, i.e., results should be consistent across independent runs of the algorithm, minimizing the impact of stochastic features. Examples from domain-specific research (e.g., robotics (Ibarz et al., 2021), energy systems (Perera & Kamalaruban, 2021), fluid dynamics (Viquerat et al., 2021)) demonstrate the significance of this triad and the need to meet the three objectives at the same time. Even for state-of-the-art algorithms, these challenges are considerably adverse in the context of real scenarios, especially when they present themselves in combination (Dulac-Arnold et al., 2021). Still, prior work generally prioritizes one objective over the rest, obviating the multi-objective perspective many real-world environments intrinsically require.

Dulac-Arnold et al. (2021) demonstrate that the latest RL algorithms such as D4PG (Barth-Maron et al., 2018) or DMPO (Abdolmaleki et al., 2018) fall short when policies face multiple real-world challenges at the same time, such as generalization to environment configurations not seen during training (e.g., physical parameters of the environment are different). While policy goodness is generally measured by return sequences, domain-specific practitioners also value generalizability and therefore, in some instances, are willing to prioritize it and trade it for training performance (Rahimian & Mehrotra, 2019). In addition, stability is a well-known problem for RL (Henderson et al., 2018); and RL algorithms are seldom ranked according to this property. As current RL algorithm design tends to be different from from this multi-objective setup, there is little knowledge on how to prioritize more than one goal at the same time and balance the tradeoffs.

Previous works propose multiple solutions to address individual goals (Derman et al., 2018; Kirk et al., 2022), the majority of them following human-driven design processes. In the context of

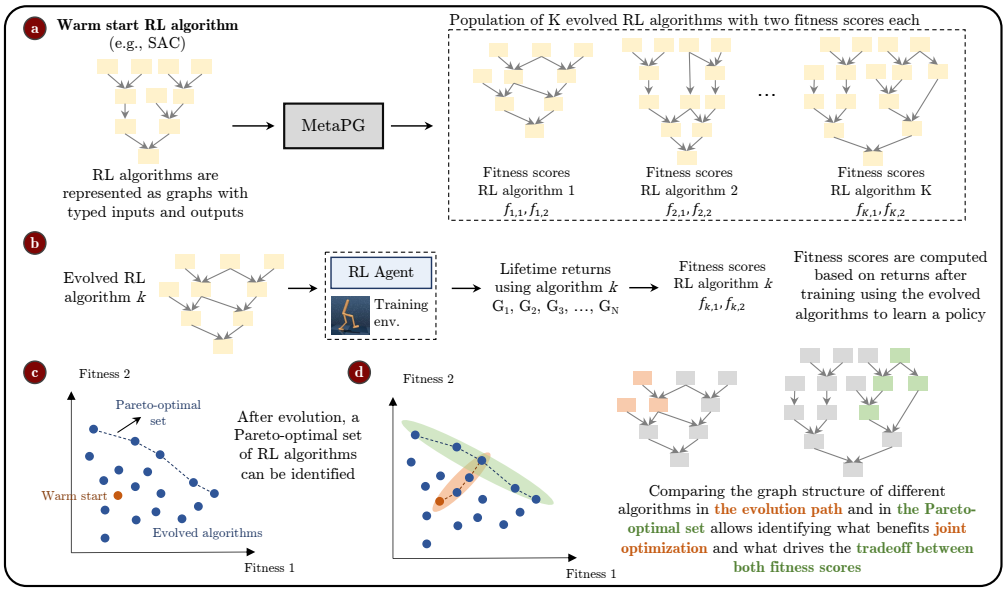


Figure 1: MetaPG overview, example with two fitness scores encoding two RL objectives. (**a**) The method starts by taking a warm-start RL algorithm represented in the form of a directed acyclic graph. MetaPG consists of a meta evolution process that, after initializing algorithms to the warm-start, discovers a population of new algorithms. (**b**) Each evolved graph is evaluated by training an agent following the algorithm encoded by it, and then computing two fitness scores based on the training outcome. (**c**) After evolution, all RL algorithms can be represented in the fitness space and a Pareto-optimal set of algorithms can be identified. (**d**) Identifying which graph substructures change across the algorithms in the Pareto set allows to see which operations favor specific RL objectives. Similarly, comparing the best graphs with the warm-start offers insights on which substructures drive the joint optimization of both objectives. MetaPG can be scaled to more than two RL objectives.

multi-objective RL, this poses two problems: 1) the costs of human-driven design might become prohibitively expensive when trying to optimize more than one RL objective, and 2) it is unclear whether designing an all-purpose RL algorithm that works across domains is possible. We argue multi-objective RL builds the case for automating RL algorithm design and speeding up the process of RL algorithm discovery. Automated Machine Learning or AutoML (Hutter et al., 2019) has proven to be a successful tool for Supervised Learning problems (Vinyals et al., 2016; Zoph et al., 2018; Real et al., 2019; Finn et al., 2017), and it has been already applied in the context of RL (Co-Reyes et al., 2021; Oh et al., 2020; Xu et al., 2020b; Finn et al., 2017). In addition to automated, it is desirable that the design process is interpretable and provides practitioners means to understand and influence the tradeoffs among objectives.

In this paper we propose MetaPG (see Figure 1), an evolutionary method that evolves a population of Policy Gradient RL algorithms (Sutton & Barto, 2018); algorithms are represented in the form of directed acyclic graphs, using multiple fitness scores encoding independent RL objectives that are taken into account by means of NSGA-II algorithm (Deb et al., 2002). Compared to manual design, this strategy allows us to explore the algorithm space more efficiently by automating search operations. Then, given this multi-objective perspective, we are able to obtain a Pareto-optimal set of RL algorithms that lay on the plane that jointly maximizes fitness with respect to each objective, approximating the underlying tradeoff among them.

To test MetaPG, we carry out multi-objective experiments using the RWRL environment suite (Dulac-Arnold et al., 2021), which provides benchmarks for generalization under physical perturbations. Similar to (Feurer et al., 2015), we warm-start the evolution with a graph-based representation of Soft Actor-Critic (SAC) (Haarnoja et al., 2018), and demonstrate that our method is able to evolve a Pareto-optimal set of RL algorithms that improve SAC's performance and generalizability by 3% and 17%, respectively, and reduce instability up to 65%. The best algorithms display a tradeoff of high returns in training configurations (performance) and configurations not seen during training (zero-shot generalizability). In addition, these algorithms are stable across independents runs. Finally,

since algorithms are represented as graphs, by comparing different algorithms in the Pareto-optimal set, we can offer an interpretation of which substructures influence the tradeoff between the different objectives for the environments considered. For instance, we find MetaPG evolves algorithms that remove the entropy term in SAC to trade performance with generalizability on the training environment.

The contributions of this paper are the following:

1. A method that combines multi-objective evolution, a search language to represent Policy Gradient algorithms as graphs, and different scoring functions for different RL objectives, to discover new RL algorithms and provide insights on tradeoffs among objectives.
2. Different sets of Pareto-optimal actor-critic algorithms that can outperform baselines like SAC on multiple objectives (single-task return, zero-shot generalizability, and stability across independent runs) over a set of continuous control tasks.

We believe this paper would be of interest to the broader RL community, as it highlights an important aspect of designing RL algorithms for practical applications: the fulfillment of multiple objectives that together pose important bottlenecks for deployment. At the same time, our findings can benefit domain-specific practitioners that wish to use RL in their domain; it provides a way of encoding their —possibly multiple— problem needs and picking the algorithm whose tradeoffs align better with the practical goals. The latter might not always be possible with current all-purpose RL algorithms.

2 Related Work

RL with multiple reward signals Accounting for and reasoning about multiple rewards encoding different objectives is necessary in some training environments. To that end, different approaches have been explored: combining all objectives into a scalar reward signal (Tan et al., 2019), training individual policies for each independent objective and then combine those policies in the distribution space (Abdolmaleki et al., 2020), training one policy per preference over objectives (Xu et al., 2020a; Yang et al., 2019), or meta learning to automate reward search (Chen et al., 2019; Faust et al., 2019). Our work does not focus on accounting for multiple reward signals in one environment but focuses on finding algorithms that optimize multiple RL objectives.

Optimizing for RL objectives A large body of works focused on identifying various application-specific RL objectives (Dulac-Arnold et al., 2021; Ibarz et al., 2021; Zhu et al., 2020; Garau-Luis et al., 2021). The literature on each of these objectives is rich and algorithms have been already proposed to address individual objectives (e.g., Offline RL (Levine et al., 2020), safe RL (Brunke et al., 2021), generalization (Kirk et al., 2022)). However, joint pursue of these objectives is seldom a goal in the literature, despite the combined presence of different challenges in multiple real-world environments. We frame our method as a multi-objective optimization of RL objectives (performance, generalizability, and stability), in which relevant goals are simultaneously encoded.

Optimizing RL components Automated RL or AutoRL has recently become an important focus in the community (Parker-Holder et al., 2022). Among the different components of the RL pipeline, some authors have explored learning RL algorithms (Kirsch et al., 2019; Oh et al., 2020; Bechtle et al., 2021) and/or their hyperparameters (Zhang et al., 2021; Hertel et al., 2020; Xu et al., 2018). Other studies focus on learning some aspect of the policy/neural network (Gaier & Ha, 2019). Finally, learning some aspects of the environment is another research direction (Ferreira et al., 2021; Florensa et al., 2018; Volz et al., 2018). Our work addresses optimizing RL algorithms by means of evolution and leaves other elements of the RL problem out of the scope. We focus on the RL algorithm given its interaction with all elements in a RL problem: states, actions, rewards, and the policy.

Evolutionary AutoML Evolutionary methods in the context of AutoML have been studied since the introduction of neuro-evolution (Miller et al., 1989; Stanley & Miikkulainen, 2002). More recently, emphasis on evolving the architecture of a neural network (Stanley et al., 2009; Jozefowicz et al., 2015; Real et al., 2019) has led to state of the art results in image classification (Real et al., 2019). More specifically, evolution has also been proposed in the context of RL (Houthooft et al., 2018; Co-Reyes et al., 2021). Our work is also related to the field of genetic programming, in which the goal is to discover computer code (Koza, 1994; Real et al., 2020; Co-Reyes et al., 2021). In this work we use a multi-objective evolutionary method to discover new RL algorithms, specifically

Policy Gradient algorithms (Sutton & Barto, 2018). Alternative approaches to evolution include RL itself (Zoph & Le, 2017; Baker et al., 2017), bayesian optimization Klein et al. (2016), grid search (Zagoruyko & Komodakis, 2017), or random search (Bergstra & Bengio, 2012).

Learning RL algorithms Loss functions play a central role in RL algorithms and are traditionally designed by human experts. Recently, several lines of work propose to view RL loss functions as tunable objects that can be optimized automatically (Parker-Holder et al., 2022). One popular approach is to use neural loss functions whose parameters are optimized via meta-gradient (Kirsch et al., 2019; Bechtle et al., 2021; Oh et al., 2020; Xu et al., 2020b). An alternative is to use symbolic representations of loss functions and formulate the problem as optimizing over a combinatorial space. One example is (Alet et al., 2020), which represents extrinsic rewards as a graph and optimizes it by cleverly pruning a search space. Learning value-based RL loss functions was first proposed in (Co-Reyes et al., 2021), and was applied to solving discrete action problems. In contrast, MetaPG focuses on continuous control problems and searches for symbolic loss functions of policy gradient actor-critic algorithms.

3 METHODS

We represent policy gradient loss functions (policy loss and critic loss) as directed acyclic graphs and use an evolutionary algorithm to evolve a population of graphs ranked based on their fitness scores. The population is seeded or warm-started with known algorithms such as SAC and undergoes mutations over time. Each graph's fitness is measured by training a RL agent with the corresponding algorithm from scratch and encodes three objectives: performance , generalizability, and stability. We use the multi-objective evolutionary algorithm NSGA-II (Deb et al., 2002) to grow a Pareto-optimal set of graphs. Algorithm 1 in Appendix B summarizes the process. The main logic of MetaPG is contained in the evaluation routine, which computes fitness scores (Section 3.1) and employs several techniques to speed up the evaluation. Section 3.2 provides RL algorithm graph representation details. See Appendix B for further implementation details.

3.1 FITNESS SCORES

This work focuses on single-task performance, zero-shot generalizability, and stability across independent runs, as the triad of RL objectives. To compute the fitness scores we rely on a set of environments E, which comprises multiple instances of the same environment class. We choose one or more configuration parameters of such environment class and set it to a different value in each of the instances. Using a specific instance $E_{train} \; 2 \; E$ to train a policy π, the performance score f_{perf} is computed as

$$f_{perf} = \frac{1}{N_{eval}} \sum_{n=1}^{N_{eval}} G(\pi, E_{train}) \tag{1}$$

where G corresponds to the episode return given a policy and an environment instance, and N_{eval} is the number of evaluation episodes. Relying on the same set of environments, the generalizability score f_{gen} is in turn computed as [1]

$$f_{gen} = \frac{1}{jEjN_{eval}} \sum_{E \; 2E} \sum_{n=1}^{N_{eval}} G(\pi, E) \tag{2}$$

Finally, stability entails getting consistent performance and generalizability outcomes across independent runs of the algorithm, mitigating the effect of stochastic elements. In that sense, stability is needed across objectives. To that end, we leverage multiple random seeds; let f be a score (performance or generalizability), we measure f multiple times by running the RL training loop using N seeds. Then we define stability-adjusted score as:

$$\tilde{f} = \mu(ff_n g_{n=1}^N) \quad \kappa \; \sigma(ff_n g_{n=1}^N) \tag{3}$$

where f_n denotes the score for seed n; μ and σ are the mean and standard deviation across the N seeds, respectively; and κ is a penalization coefficient. Then, the fitness of a graph is the tuple ($\tilde{f}_{perf}$, $\tilde{f}_{gen}$).

[1] More precisely, should be $E \in \mathcal{E}$ except E_{train}. In practice, we find this makes no significant difference.

3.2 RL ALGORITHM REPRESENTATION

Using a search language inspired by Co-Reyes et al. (2021), we encode RL algorithms as graphs consisting of typed nodes sufficient to represent a wide class of Policy Gradient algorithms. As a representative example, Appendix D presents the encoding for SAC that we use in this paper. Nodes in the graph encode algorithm inputs, operations, and algorithm outputs. The inputs include elements from experience tuples, constants such as the discount factor γ, a policy network π, and multiple critic networks Q_i. Operation nodes support intermediate algorithm instructions such as basic arithmetic, array, or neural network operations. Then, the outputs of the graphs correspond to the policy and critic losses. The gradient descent minimization process takes these outputs and computes their gradient with respect to the respective network parameters. In Appendix A we provide a full description of the search language and nodes considered. MetaPG's search language supports both on-policy and off-policy algorithms; however, in this paper we focus on off-policy algorithms given its better sample efficiency.

4 RESULTS

In this section we aim to answer the following questions: 1) Is MetaPG capable of evolving algorithms that improve upon the three objectives? 2) Can we derive an explanation on what does bias the scores of specific algorithms by looking at their graph structure?

We divide the experiments into a meta training and a meta validation phase. The same set of environments E is used in both cases, and each algorithm has two sets of fitness scores, one for each of the two phases. The main difference between both phases consists of the set of random seeds used; the role of meta validation is to avoid declaring as best option algorithms that overfit to a specific set of random seeds. First, during meta training the evolution process is carried out and the algorithms are scored using a specific set of seeds S_{train}. Once evolution is over, we meta validate all algorithms using a different set of seeds S_{valid}. Since the evolution process is also non-deterministic per se, we run each experiment 10 different times without configuration changes. Meta validation results are aggregations over the populations from the 10 experiments. Then, in our analyses of the results, we focus on the Pareto-optimal set of meta-validated algorithms.

4.1 TRAINING SETUP

Training environments Cartpole and Walker environments from the RWRL Environment Suite (Dulac-Arnold et al., 2021), and Gym Pendulum, serve as training environments. We define different instances of these environments by varying the pole length in Cartpole, the thigh length in Walker, and the pendulum length and mass in Pendulum. See Appendix C for the specifics on environment configurations.

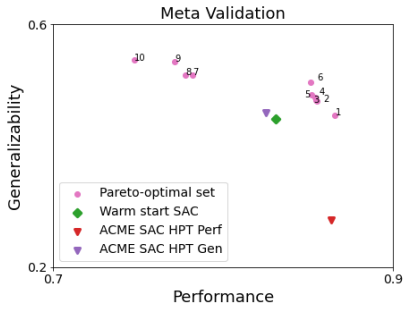
Meta training details The population is 1,000 individuals and the maximum graph size is 60 nodes. All are initialized using SAC as a warm-start (see Appendix D). For the RL algorithm evaluation, we use 10 different seeds S_{train}, fix the number of evaluation episodes N_{eval} to 20, and normalize all fitness scores to the range [0, 1]. We set $\kappa = 1$ in Equation 3. Additional details are in Appendix E.

Meta validation details During meta validation, we use a set of 10 seeds S_{valid}, disjoint with respect to S_{train}. We find that 10 seeds achieve a good balance between preventing overfittng and having affordable evaluation time. N_{eval} is also fixed to 20 when computing meta validation fitnesses.

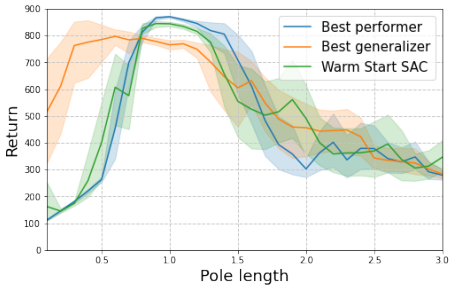
RL Training details The architecture of the policies corresponds to two-layer MLPs with 256 units each. Additional training details are presented in Appendix F.

4.2 OPTIMIZING PERFORMANCE, GENERALIZABILITY, AND STABILITY

Figure 2 shows the Pareto-optimal set of best RL algorithms obtained after applying MetaPG to RWRL Cartpole. In Figure 2a we compare the Pareto-optimal set with the warm-start SAC and ACME SAC (Hoffman et al., 2020). When running ACME SAC on RWRL Cartpole we first do hyperparameter tuning and pick the two configurations that lead to the best performance and the best generalizability (ACME SAC HPT Perf and ACME SAC HPT Gen, respectively). We do not do hyperparameter tuning for the warm-start; see Appendix G.7 for more details.

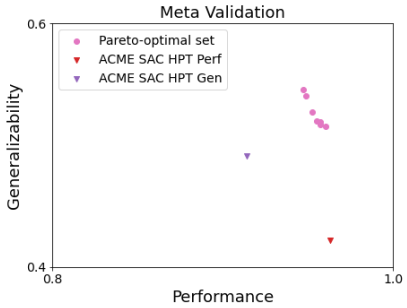


(a) Stability-adjusted fitness scores (computed using Equation 3) for algorithms in the Pareto-optimal set.

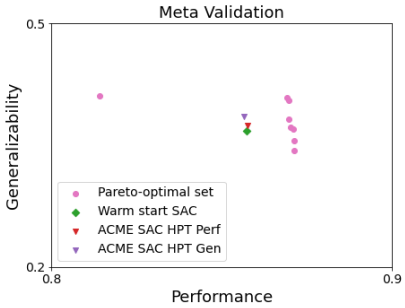


(b) Average return and standard deviation across seeds when evaluating resulting policies in multiple RWRL Cartpole instances with different pole lengths. A length of 1.0 is used as training configuration.

Figure 2: Evolution results (meta validation across 10 different seeds) alongside the warm-start algorithm (SAC), and the hyperparameter-tuned ACME SAC when using the RWRL Cartpole environment for training. We show the Pareto-optimal set of algorithms that results after merging the 10 populations corresponding to the 10 repeats of the experiment. The best performer and best generalizer correspond to the algorithms with the highest stability-adjusted performance and generalizability scores, respectively, according to Equations 1, 2, and 3.



(a) RWRL Walker.



(b) Gym Pendulum.

Figure 3: Stability-adjusted fitness scores (Equation 3) of evolved algorithms (meta validation across 10 different seeds) alongside the warm-start SAC, and the hyperparameter-tuned ACME SAC. We show the Pareto-optimal set of algorithms that results after merging the 10 populations corresponding to the 10 repeats of the experiment. Since the warm-start is not hyperparameter-tuned before evolution, in the case of RWRL Walker its fitness scores are too distant from the Pareto-optimal set.

The results show that, by mutating the graphs, MetaPG discovers RL algorithms that improve upon the warm-start and ACME SAC's performance and generalizability. Table 1 shows the average fitness (standard error of the mean) for the best performer, the best generalizer, and one of the relevant algorithms in the Pareto-optimal set. Compared to the warm-start, the best performer achieves a 3% improvement in the performance score, the best generalizer achieves a 17% increase in the generalizability score, and the selected algorithm in the Pareto-optimal set (Pareto point 6) achieves a 1% and a 9% increase in both **performance** and **generalizability**, respectively. Compared to hyperparameter-tuned SAC, evolved algorithms are able to improve upon both metrics, especially when it comes to generalizability. MetaPG also discovers a Pareto-optimal set of algorithms with the same behaviour with respect to SAC in both RWRL Walker and Gym Pendulum, as seen in Figures 3a and 3b, respectively. Then, in terms of the **stability** objective, evolved algorithms achieve between 33% and 65% reduction in the standard deviation of the results and therefore improve in that dimension as well. Additional information on the stability of the algorithms is in Appendix G.

Figure 2b compares how the best performer and the best generalizer behave in different instances of the environment in which we change the pole length. We follow the same procedure as described in (Dulac-Arnold et al., 2021). The best performer achieves better return in the training configuration than the warm-start's. The best generalizer in turn achieves a lower return but we can observe how it is able to trade it for higher returns in configurations outside of the training regime, being better at

RL Algorithm	Avg. Perf. score (f_{perf})		Avg. Gen. score (f_{gen})	
Pareto point 1: Best performer	0.871	0.003	0.475	0.016
Pareto point 6	0.854	0.002	0.531	0.017
Pareto point 10: Best generalizer	0.770	0.014	0.570	0.019
warm-start SAC	0.845	0.009	0.487	0.027
ACME SAC HPT Perf	0.865	0.001	0.372	0.060
ACME SAC HPT Gen	0.845	0.012	0.518	0.040

Table 1: Average performance and generalizability scores (Equations 1 and 2, respectively) standard error of the mean for three algorithms in the Pareto-optimal set and SAC when using RWRL Cartpole as a training environment. We compute these metrics across 10 seeds.

zero-shot generalization. The same behavior holds when using RWRL Walker and Gym Pendulum as training environments (see results in Appendix G).

4.3 Analyzing the evolved RL algorithms

Next we analyze the algorithms that correspond to the best performer and best generalizer, both evolved from the warm-start graph representing SAC (see Appendix D). The policy loss L and critic loss L_{Q_i} observed from the graph structure for the best performer are the following:

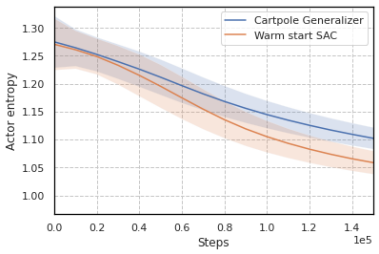
$$L^{perf} = \mathsf{E}_{(s_t;a_t;s_{t+1}) \sim D}\left[\log(\min(\pi(a_{t+1}|s_{t+1}),\gamma)) \quad \min_i Q_i(s_t,a_t)\right] \tag{4}$$

$$L^{perf}_{Q_i} = \mathsf{E}_{(s_t;a_t;r_t;s_{t+1}) \sim D}\left[(r_t + \gamma\,(Q_{targ_i}(s_{t+1},a_{t+1})) \quad Q_i(s_t,a_t))^2\right] \tag{5}$$

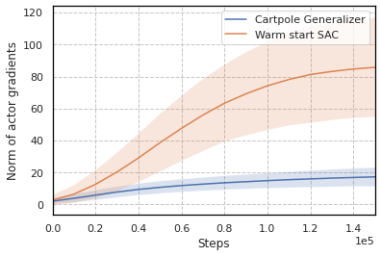
where $a_t \sim \pi(\cdot|s_t)$, $a_{t+1} \sim \pi(\cdot|s_{t+1})$, and D is an experience dataset extracted from the replay buffer. Likewise, the loss equations for the best generalizer are:

$$L^{gen} = \mathsf{E}_{(s_t;a_t;s_{t+1}) \sim D}\left[\log \pi(a_t|s_t) \quad \min_i Q_i(s_{t+1},a_t)\right] \tag{6}$$

$$L^{gen}_{Q_i} = \mathsf{E}_{(s_t;a_t;r_t;s_{t+1}) \sim D}\left[\text{atan}\left(\left(r_t + \gamma\left(\min_i Q_{targ_i}(s_{t+1},a_t) \quad \log \pi(a_t|s_t)\right) \quad Q_i(s_t,a_t)\right)^2\right)\right] \tag{7}$$



(a) Average entropy of the policy during training for RWRL Cartpole.



(b) Average gradient norm of the actor loss during training for RWRL Cartpole.

Figure 4: Analysis of the entropy and gradient norm of the actor when evaluating the best generalizer from RWRL Cartpole in comparison to the warm-start. As in meta training conditions, we run the analysis for a total of 150 episodes of 1,000 timesteps each.

While both algorithms resemble the warm-start SAC (see Appendix D), we can observe differences in the loss functions. On one hand, the best performer does not include the entropy term in the critic loss while the best generalizer does (i.e., they correspond to setting α to 0 and 1 in the original SAC algorithm (Haarnoja et al., 2018), respectively). This aligns with the hypothesis that, since ignoring the entropy pushes the agent to exploit more and explore less, the policy of the best performer overfits better to the training configuration compared to SAC. In contrast, the best generalizer is able to explore more (i.e., visit more of the state/action space) and thus achieve higher generalizability.

Figure 4a validates the latter observation showing a higher entropy for the best generalizer's actor compared to the warm-start's.

We can also establish other connections between the graphs and the fitness of the algorithms, such as the use of the arctangent in the critic loss of the best generalizer. In this case, supported by Figure 4b, we observe this operation serves as a way of clipping the loss, which makes gradients smaller and thus prevents the policy's parameters from changing too abruptly. In that sense, given a fixed number of training episodes, it has an early-stopping effect and results in a policy less overfitted. The equations for the resulting algorithms in the remaining environments, and an extended version of Figure 4 in which we train for more episodes, are both presented in Appendix G.

4.4 Discussion

We have seen in Figure 2 that evolution can discover algorithms that perform better than SAC on a particular environment. We emphasize that evolved algorithms are not hyperparameter-tuned, while SAC results are tuned. This suggests that if we allow hyperparameter tuning during or after evolution, our results could be enhanced. On the other hand, the Pareto-optimal set in Figure 2 is formed by combining 10 separate runs of evolution, since each individual run could converge to a different local optimum. We leave it to future work to improve the robustness of a single search experiment.

We also ran transfer experiments between the different environments (see Appendix G). We observed that, while evolved algorithms transfer reasonably well (especially the best performers), they do not perform better than SAC in the new environment. This might suggest that a direction of future work is to improve the transferrability of the evolved algorithms. At the same time, it poses an interesting research question of determining whether MetaPG is better suited to find "super algorithms" for specific environments or the new generation of all-purpose algorithms.

The analysis above focused on two extreme points (best performer and generalizer) in the Pareto-optimal set. It is possible to interpolate between these two points to form an ensemble loss function. Such loss function may give additional flexibility for practitioners when designing an RL system by encoding complex design choices into an interpolation across objectives.

The graph analysis above is preliminary and does not fully explain why algorithms generalize better. In Equation 6, there is an unusual structure $Q_i(s_{t+1}, a_t)$ where $a_t \quad \pi(\,|s_t)$. Our hypothesis is that using a_t instead of a_{t+1} may help to reduce the impact of extrapolation error of Q, and may introduce a form of smoothness regularization. We leave it as future work to further validate these hypotheses.

Finally, we found that the use of the arctangent in Equation 6 might benefit generalizability by serving as an early-stop before the policy overfits to the training configuration. We fixed a certain number of training episodes as a compromise between achievable returns and evaluation runtimes. In Appendix G we see that letting run for longer makes certain metrics across algorithms converge to similar values. We acknowledge that training until convergence is usually preferred; however, in certain applications the number of training episodes might be a constraint, so we find MetaPG's ability to exploit this kind of constraints beneficial in those setups. In that sense, other interesting RL objectives such as sample efficiency could be incorporated into our method as additional fitness metrics.

5 Conclusion

We presented MetaPG, a method that evolves RL algorithms to optimize multiple RL objectives simultaneously and applied it to discovering algorithms that perform well, generalize across environment configurations, and are stable. MetaPG addresses the joint optimization in RL, focusing on a triad of objectives with real-world implications. The experiments in RWRL Cartpole, RWRL Walker, and Gym Pendulum demonstrated that MetaPG discovered algorithms that outperform SAC, achieving a 3% and 17% improvement in performance and generalizability, respectively, and a reduction of 33% to 65% in the standard deviation of the results across seeds. We have analyzed the evolved algorithms and linked specific elements in their structure to fitness results, such as the removal of the entropy term to benefit performance.

AUTHOR CONTRIBUTIONS

JGL, ER, AF conceived the project. AF assembled the team. JGL initiated research ideas, ran experiments and analysis. ER, AF, JT advised on evolutionary algorithms, reinforcement learning, and real-world applications, respectively. ER built the evolutionary infrastructure, with contributions from YM and AP. JGL developed the search space, with contributions from YM, JD, AP, and JT. JGL wrote the paper.

ACKNOWLEDGMENTS

The authors want to thank Ramki Gummadi for helpful discussions, Hicham El Zein and Stephen Jonany for code contributions, and Izzeddin Gur for useful feedback.

REFERENCES

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation, 2018.

Abbas Abdolmaleki, Sandy Huang, Leonard Hasenclever, Michael Neunert, Francis Song, Martina Zambelli, Murilo Martins, Nicolas Heess, Raia Hadsell, and Martin Riedmiller. A distributional view on multi-objective policy optimization. In *International Conference on Machine Learning*, pp. 11–22. PMLR, 2020.

Ferran Alet, Martin F. Schneider, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Meta-learning curiosity algorithms. mar 2020. URL http://arxiv.org/abs/2003.05325.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning, 2017.

Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients, 2018.

Sarah Bechtle, Artem Molchanov, Yevgen Chebotar, Edward Grefenstette, Ludovic Righetti, Gaurav Sukhatme, and Franziska Meier. Meta Learning via Learned Loss. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 4161–4168. IEEE, jan 2021. ISBN 978-1-7281-8808-9. doi: 10.1109/ICPR48806.2021.9412010.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

Lukas Brunke, Melissa Greeff, Adam W. Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P. Schoellig. Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. aug 2021. URL http://arxiv.org/abs/2108.06266.

Xi Chen, Ali Ghadirzadeh, Marten Bjorkman, and Patric Jensfelt. Meta-Learning for Multi-objective Reinforcement Learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 977–983. IEEE, nov 2019. ISBN 978-1-7281-4004-9. doi: 10.1109/IROS40897.2019.8968092.

John D Co-Reyes, Yingjie Miao, Daiyi Peng, Esteban Real, Sergey Levine, Quoc V Le, Honglak Lee, and Aleksandra Faust. Evolving reinforcement learning algorithms. *arXiv preprint arXiv:2101.03958*, 2021.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

Esther Derman, Daniel J. Mankowitz, Timothy A. Mann, and Shie Mannor. Soft-robust actor-critic policy-gradient, 2018.

Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, pp. 1–50, 2021.

Aleksandra Faust, Anthony Francis, and Dar Mehta. Evolving Rewards to Automate Reinforcement Learning. may 2019. URL http://arxiv.org/abs/1905.07628.

Fabio Ferreira, Thomas Nierhoff, and Frank Hutter. Learning synthetic environments for reinforcement learning with evolution strategies, 2021.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/11d0e6287202fced83f79975ec59a3a6-Paper.pdf.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1515–1528. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/florensa18a.html.

Adam Gaier and David Ha. Weight agnostic neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/e98741479a7b998f88b8f8c9f0b6b6f1-Paper.pdf.

Juan Jose Garau-Luis, Edward Crawley, and Bruce Cameron. Evaluating the progress of deep reinforcement learning in the real world: aligning domain-agnostic and domain-specific research, 2021.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Lars Hertel, Pierre Baldi, and Daniel L. Gillen. Quantity vs. quality: On hyperparameter optimization for deep reinforcement learning, 2020.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning, 2020.

Rein Houthooft, Yuhua Chen, Phillip Isola, Bradly Stadie, Filip Wolski, OpenAI Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. In *Advances in Neural Information Processing Systems*, pp. 5400–5409, 2018.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721, 2021.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International conference on machine learning*, pp. 2342–2350. PMLR, 2015.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning, 2022.

Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Improving Generalization in Meta Reinforcement Learning using Learned Objectives. oct 2019. URL http://arxiv.org/abs/1910.04098.

Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. 2016.

John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pp. 379–384, 1989.

Junhyuk Oh, Matteo Hessel, Wojciech M. Czarnecki, Zhongwen Xu, Hado van Hasselt, Satinder Singh, and David Silver. Discovering Reinforcement Learning Algorithms. jul 2020. URL http://arxiv.org/abs/2007.08794.

Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, Frank Hutter, and Marius Lindauer. Automated Reinforcement Learning (AutoRL): A Survey and Open Problems. jan 2022. URL http://arxiv.org/abs/2201.03916.

A.T.D. Perera and Parameswaran Kamalaruban. Applications of reinforcement learning in energy systems. *Renewable and Sustainable Energy Reviews*, 137:110618, 2021. ISSN 1364-0321. doi: https://doi.org/10.1016/j.rser.2020.110618. URL https://www.sciencedirect.com/science/article/pii/S1364032120309023.

Hamed Rahimian and Sanjay Mehrotra. Distributionally robust optimization: A review, 2019.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Esteban Real, Chen Liang, David So, and Quoc Le. Automl-zero: Evolving machine learning algorithms from scratch. In *International Conference on Machine Learning*, pp. 8007–8019. PMLR, 2020.

Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

Jonathan Viquerat, Philippe Meliga, and Elie Hachem. A review on deep reinforcement learning for fluid mechanics: an update, 2021.

Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M Lucas, Adam Smith, and Sebastian Risi. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the genetic and evolutionary computation conference*, pp. 221–228, 2018.

Jie Xu, Yunsheng Tian, Pingchuan Ma, Daniela Rus, Shinjiro Sueda, and Wojciech Matusik. Prediction-Guided Multi-Objective Reinforcement Learning for Continuous Robot Control. In *Proceedings of the 37th International Conference on Machine Learning*, 2020a.

Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/2715518c875999308842e3455eda2fe3-Paper.pdf.

Zhongwen Xu, Hado van Hasselt, Matteo Hessel, Junhyuk Oh, Satinder Singh, and David Silver. Meta-Gradient Reinforcement Learning with an Objective Discovered Online. jul 2020b. URL http://arxiv.org/abs/2007.08433.

Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A Generalized Algorithm for Multi-Objective Reinforcement Learning and Policy Adaptation. aug 2019. URL http://arxiv.org/abs/1908.08342.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017.

Baohe Zhang, Raghu Rajan, Luis Pineda, Nathan Lambert, André Biedenkapp, Kurtland Chua, Frank Hutter, and Roberto Calandra. On the importance of hyperparameter optimization for model-based reinforcement learning. In Arindam Banerjee and Kenji Fukumizu (eds.), *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 4015–4023. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/zhang21n.html.

Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar, and Sergey Levine. The ingredients of real-world robotic reinforcement learning. *arXiv preprint arXiv:2004.12570*, 2020.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=r1Ue8Hcxg.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

A SEARCH SPACE DETAILS

In this section we present the details of the search space; we divide the nodes into input, output, and operation nodes. Output nodes correspond to losses computed by the algorithm, whose gradient with respect to the algorithm inputs is then computed in a training loop. In this process, agents then learn the policy by means of experience tuples coming from a replay buffer. MetaPG admits both continuous and discrete action spaces; specific nodes in the graphs —e.g., the networks— are adapted to work with the corresponding space.

During the evolution process, we fix a maximum number of nodes, which consists of the aforementioned input and output nodes, and several operation nodes. The majority of operation nodes treat input elements as tensors with variable shapes in order to maximize graph flexibility. Each node possesses a certain number of input and output edges, which are determined by the specific operation this node carries out. For example, a node that takes in two tensors and multiplies them element-wise has two input edges and a single output edge. A complete list of the nodes considered follows:

Input nodes We only encode canonical RL elements as inputs:

Policy network π

Two critic networks, Q_1 and Q_2, and two target critic networks, Q_{targ_1} and Q_{targ_2}

Batch of states s_t and next states s_{t+1}

Batch of actions a_t

Batch of rewards r_t

Discount factor γ

Output nodes The output of these nodes is used as loss function to compute gradient descent on:

Policy loss L

Critic loss L_{Q_i}

Operation nodes These nodes operate generally on tensors and can broadcast operations when input sizes do not match:

Addition: add two, three, or four tensors

Multiplication: compute element-wise product of two or three tensors

Subtract two tensors

Divide two tensors and add constant ϵ to the denominator

Neural network operations: Action distribtuion from state, stopping gradient computation

Operations with action distributions: Sample, Log-probability

Mean, sum, and standard deviation over last axis of array or over entire array

Cumulative sum, cumulative sum with discount

Squared difference

Multiply by a constant: -1, 0.1, 0.01, 0.5, 2.0

Minimum and maximum over last axis of a tensor

Minimum and maximum element-wise between two tensors

Other general operations: clamp, absolute value, square, logarithm, exponential

Trigonometry functions

B IMPLEMENTATION DETAILS

Multi-objective evolution Algorithm 1 details the evolution process for MetaPG , in which `Offspring` and `RankAndSelect` are NSGA-II subroutines (Deb et al., 2002).

Mutation In our work we initialize the population using a provided RL algorithm as a warm-start; all individuals are copies of this algorithm's graph. Once the population is initialized, individuals undergo mutations that change the structure of their respective graphs. Specifically, mutations consist of randomly selecting one parent individual in the population and either replacing one or more nodes in the graph or switching the connections for one or more edges. The specific number of nodes or edges that are affected by the mutation is randomly sampled for each different individual. To prevent introducing corrupted child graphs into the population, the framework checks operation consistency, i.e., for each operation, it makes sure the shapes of input tensors are valid. This avoids passing scalars to operations that only admit arrays and vice versa.

Algorithm 1 MetaPG Overview

Input: Training environments $\mathcal{E}$
Initialize: Initialize population P_0 of loss function graphs (random initialization or bootstrap with an algorithm such as SAC).

for L in P_0 **do** $L.score$ Eval$(L, \mathcal{E})$
end for
Q_0 Offspring(P_0) ▷ NSGA-II
for L in Q_0 **do** $L.score$ Eval$(L, \mathcal{E})$
end for
for $t = 1$ **to** G **do**
  R $P_{t-1} \bigcup Q_{t-1}$
  P_t RankAndSelect(R) ▷ NSGA-II
  Q_t Offspring(P_t) ▷ NSGA-II
  for L in Q_t **do** $L.score$ Eval$(L, \mathcal{E})$
  end for
end for
Output: Pareto-front of all loss function graphs.

Hashing In addition, to avoid repeated evaluations, MetaPG hashes (Real et al., 2020) all graphs in the population. Once the method produces a child graph and proves its consistency, it computes its hash value and compares it against all previously registered hash values. If the hash value coincides with that from an older individual in the population, its fitness scores are copied to the new individual. Otherwise, we evaluate the new individual using the underlying RL loss function encoded by its graph. In our case, we not only want to make sure that we do not evaluate the same graph twice, but also identify graphs that are different in form but identical in function. To that end, before hashing we prune all graphs so that only nodes that contribute to the output are taken into account. Then, we look at the gradients of the output losses with respect to the input parameters and use their concatenation as the hash value. In the process we use a fixed set of synthetic inputs with a batch size of 16.

Encoding multiple objectives MetaPG keeps the population to a fixed size during evolution. To decide which individuals should be removed in the process, the method makes use of different fitness scores that encode each of the RL objectives considered. These scores are not combined but treated separately in a multi-objective fashion. This means that, after evaluating a graph i, it will have fitness scores $\{f_{i;1}, f_{i;2}, ..., f_{i;F}\}$, where F is the number of objectives considered. Then, when comparing two graphs i and j, we say i has higher fitness than j iff $f_{i;k}$ $f_{j;k}, \forall k$, with at least one fitness score k' such that $f_{i;k'} > f_{j;k'}$. In this case we also say graph i Pareto-dominates graph j. If neither i Pareto-dominates j nor vice versa, we say both graphs are Pareto-optimal.

The process of removing individuals from the population follows the NSGA-II algorithm (Deb et al., 2002) which, assuming a maximum population size of P_{max} individuals:

1. From a set of P individuals, with $|P| > P_{max}$, it computes the set P_{opt} of Pareto-optimal fittest graphs. None of the graphs in P_{opt} is Pareto-dominated by any other graph in the population and, if a graph i in P is Pareto-dominated by at least one other graph, then i does not belong to the Pareto-optimal set.
2. If $|P_{opt}|$ P_{max}, the graphs are ranked based on their crowding distance in the fitness space. This favors individuals that are further apart from other individuals in the fitness space. The fittest P_{max} individuals of the Pareto-optimal set P_{opt} are kept in the population.
3. Otherwise, if $|P_{opt}| < P_{max}$, the set P_{opt} is kept in the population and the process is repeated taking P $P \setminus P_{opt}$ and P_{max} P_{max} $|P_{opt}|$.

C Environment Configurations

In this work we use three envrionments for our experiments: Cartpole and Walker from the RWRL Environment Suite (Dulac-Arnold et al., 2021), and Gym Pendulum. In Table 2 we list the training

configuration used for each and the parameters that we use to assess the generalizability of the policies. The parameters that are not listed are fixed to the default values for the environment in question. In the case of Gym Pendulum, we have two perturbation parameters; the generalizability score is computed by first sweeping through the perturbation values for one, taking the average f_{gen_1}, repeating the same process for the other to compute f_{gen_2}, and then computing the average of both to get the final score, i.e., $f_{gen} = (f_{gen_1} + f_{gen_2})/2$.

Environment parameter	Value
RWRL Cartpole	
Rollout length	1,000
Min. return	0
Max. return	1,000
Training episodes	150
Perturbation parameter (PP)	Pole length
PP Default value	1.0
PP Generalizability values	0.1 to 3.0 in steps of 0.1
RWRL Walker	
Rollout length	1,000
Min. return	0
Max. return	1,000
Training episodes	225
Perturbation parameter (PP)	Thigh length
PP Default value	0.225
PP Generalizability values	.1, .125, .15, .175, .2, .225, .25, .3, .35, .4, .45, .5, .55, .6, .7
Gym Pendulum	
Rollout length	2,000
Min. return	-2,000
Max. return	0
Training episodes	100
Perturbation parameter 1 (PP1)	Pendulum mass
PP1 Default value	1.0
PP1 Generalizability values	.1, .2, .4, .5, .75, 1.0, 1.5, 2.0, 3.0, 5.0, 7.5, 10.0
Perturbation parameter 2 (PP2)	Pendulum length
PP2 Default value	1.0
PP2 Generalizability values	.1, .2, .4, .5, .75, 1.0, 1.5, 2.0, 3.0, 5.0, 7.5, 10.0

Table 2: Environment parameters and perturbations.

D WARM-START SAC

We present the version of Soft Actor-Critic (SAC) (Haarnoja et al., 2018) used in this work as the warm-start algorithm to initialize the population. We first present the equations for the policy loss L^{WS} and critic loss $L^{WS}_{Q_i}$:

$$L^{WS} = \mathbb{E}_{(s_t;a_t) \sim D}\left[\log \pi(a_t|s_t) \quad \min_i Q_i(s_t, a_t)\right] \tag{8}$$

$$L^{WS}_{Q_i} = \mathbb{E}_{(s_t;a_t;r_t;s_{t+1}) \sim D}\left[\left(r_t + \gamma\left(\min_i Q_{targ_i}(s_{t+1}, a_{t+1}) \quad \log \pi(a_{t+1}|s_{t+1})\right) \quad Q_i(s_t, a_t)\right)^2\right] \tag{9}$$

where $a_t \sim \pi(\cdot|s_t)$, $a_{t+1} \sim \pi(\cdot|s_{t+1})$, and D is a dataset from the replay buffer. Then, in Figure 5 we represent these two equations that define the SAC algorithm in the form of a graph with typed input and outputs. MetaPG then modifies this graphs following the procedure described in Section 3.

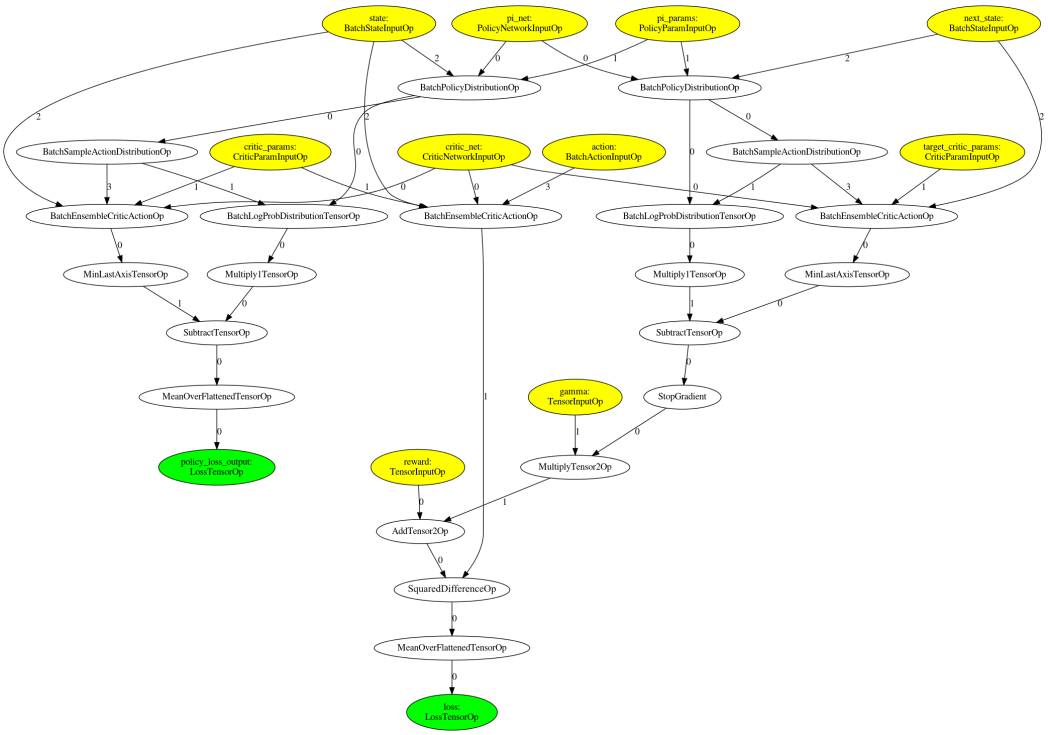


Figure 5: Soft Actor-Critic (SAC) algorithm represented as a graph to initialize the population as a warm-start algorithm.

E ADDITIONAL EVOLUTION DETAILS

This section outlines several additional implementation considerations of the evolutionary process:

Algorithms are initialized using the warm-start SAC graph, which consists of 33 nodes. Additional operation nodes are added to each individual until reaching the maximum amount of 60 nodes.

Evaluations for different individuals in the population are carried out in parallel, while evaluating across seeds for one algorithm is done sequentially. If, after training with 3 seeds or more, a specific algorithm yields a policy whose performance metric is lower than a certain threshold, MetaPG stops the evaluation for that individual and sets its fitnesses at that point. This is done to prevent spending too many resources on algorithms that are likely to yield bad policies, following the same rationale outlined in (Co-Reyes et al., 2021).

During mutation, there is a 50% chance an individual undergoes node mutation and a 50% chance it undergoes edge mutation.

During node mutation, there is a 50% chance of replacing one node, a 25% chance of replacing 2 nodes, a 12.5% chance of replacing 4 nodes, and a 6.25% chance of replacing 8 and 16 nodes, respectively.

During edge mutation, only one edge in the graph is replaced.

F ADDITIONAL RL TRAINING DETAILS

An individual encoding a RL algorithm in the form of a graph is evaluated by training an agent using such algorithm. We use an implementation based on an ACME agent (Hoffman et al., 2020). The configuration of the training setup are shown in Table 3.

Parameter	Value
Discount factor γ	0.99
Batch size	64 (RWRL Cartpole and Gym Pendulum) 128 (RWRL Walker)
Learning rate	3 10 ^{4}
Target smoothing coeff. τ	0.005
Replay buffer size	1,000,000
Min. num. samples in the buffer	10,000
Gradient updates per learning step	1
n step	1
Reward scale	5.0
Actor network	MLP (256, 256)
Actor activation function	ReLU
Tanh on output of actor network	Yes
Critic networks	MLP (256, 256)
Critic activation function	ReLU

Table 3: RL Training setup.

G ADDITIONAL RESULTS

In this section we present the additional results of the paper. We first introduce the remaining figures for RWRL Cartpole, then outline evolution results for RWRL Walker and Gym Pendulum, then show how different algorithms in the population for all three environments compare in terms of stability, then provide the equations of the evolved algorithms for RWRL Walker and Gym Pendulum, and finally provide more details on other metrics of evolved algorithms.

G.1 EVOLUTION RESULTS FOR RWRL CARTPOLE

Figure 6 shows the resulting population when running evolution using the RWRL Cartpole environment (Dulac-Arnold et al., 2021) and Table 4 shows the average fitness scores (standard error of the mean) for each algorithm in the Pareto-optimal set.

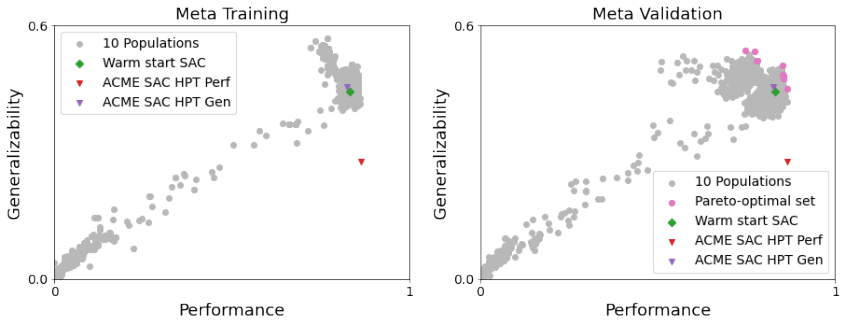


Figure 6: Meta training and meta validation stability-adjusted fitness scores (computed using Equation 3 across 10 seeds) for each RL algorithm in the population alongside the warm-start (SAC) and ACME SAC when using the RWRL Cartpole environment for training. We show the meta validated Pareto-optimal set of algorithms that results after merging the 10 populations corresponding to the 10 repeats of the experiment.

G.2 EVOLUTION RESULTS FOR RWRL WALKER

We present evolution results when running MetaPG with RWRL Walker as the training environment. In Figures 7 and 8 we show the resulting population and the performance across environment configurations for the best performer and the best generalizer in the Pareto-optimal set, respectively.

RL Algorithm	Avg. Perf. score (f_{perf})		Avg. Gen. score (f_{gen})	
Pareto 1: Best performer	0.871	0.003	0.475	0.016
Pareto 2	0.857	0.001	0.513	0.025
Pareto 3	0.857	0.002	0.514	0.025
Pareto 4	0.856	0.002	0.517	0.024
Pareto 5	0.855	0.002	0.520	0.023
Pareto 6	0.854	0.002	0.531	0.017
Pareto 7	0.798	0.010	0.540	0.016
Pareto 8	0.794	0.010	0.546	0.018
Pareto 9	0.783	0.007	0.579	0.026
Pareto 10: Best generalizer	0.770	0.014	0.570	0.019
warm-start SAC	0.845	0.009	0.487	0.027
ACME SAC HPT Perf	0.865	0.001	0.372	0.060
ACME SAC HPT Gen	0.845	0.012	0.518	0.040

Table 4: Average performance and generalizability scores (Equations 1 and 2, respectively) standard error of the mean for the 10 algorithms in the Pareto-optimal set and SAC when using RWRL Cartpole as a training environment. We compute these metrics across 10 seeds.

Exact numbers for each algorithm in the Pareto-optimal set can be found in Table 5. This table also shows the scores of the warm-start and ACME SAC. As covered in Appendix G.7, we do not hyperparameter-tune the warm-start before the experiments. As a result, the warm-start might perform poorly, as is the case in this environment. We can observe MetaPG is able to increase the fitness of the evolved algorithms during the evolution process.

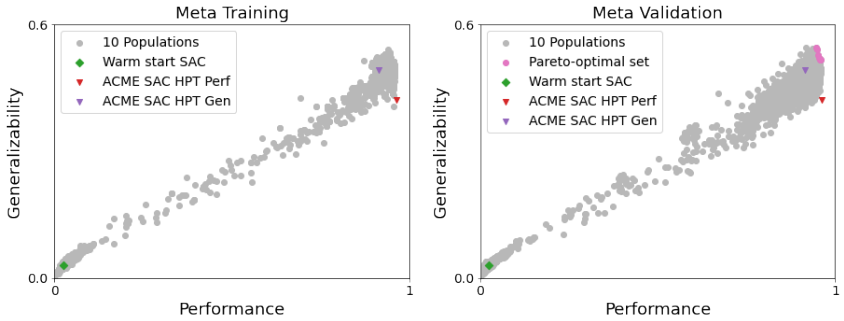


Figure 7: Meta training and meta validation stability-adjusted fitness scores (computed using Equation 3 across 10 seeds) for each RL algorithm in the population alongside the warm-start (SAC) and ACME SAC when using the RWRL Walker environment for training. We show the meta validated Pareto-optimal set of algorithms that results after merging the 10 populations corresponding to the 10 repeats of the experiment.

G.3 Evolution results for Pendulum

We present evolution results when running MetaPG with Gym Pendulum as the training environment. In Figures 9 and 10 we show the resulting population and the performance across environment configurations for the best performer and the best generalizer in the Pareto-optimal set, respectively. In the case of Pendulum, the generalizability fitness score is computed across the perturbation of two different parameters: the pendulum mass and the pendulum length. These parameters are changed separately, as opposed to varying both the mass and length of the pendulum in the same run. Exact numbers can be found in Table 6, in which an average improvement over the warm-start of 1% in performance and 16% in generalizability is achieved.

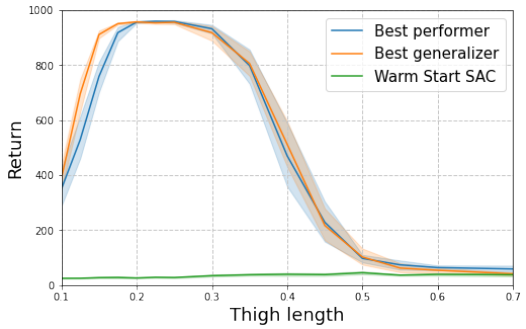


Figure 8: Average and standard deviation across seeds of the meta validation performance of the best performer, the best generalizer, and the warm-start (SAC) when training on a single configuration of RWRL Walker and evaluating on multiple unseen ones. The thigh length changes across environment configurations and a length of 0.225 is used as training configuration.

RL Algorithm	Avg. Perf. score (f_{perf})		Avg. Gen. score (f_{gen})	
Pareto 1: Best performer	0.963	0.002	0.544	0.018
Pareto 2	0.962	0.003	0.536	0.012
Pareto 3	0.960	0.002	0.542	0.015
Pareto 4	0.959	0.003	0.541	0.013
Pareto 5	0.960	0.005	0.541	0.009
Pareto 6	0.954	0.003	0.555	0.009
Pareto 7: Best generalizer	0.955	0.005	0.569	0.015
warm-start SAC	0.028	0.002	0.033	0.001
ACME SAC HPT Perf	0.968	0.003	0.444	0.014
ACME SAC HPT Gen	0.926	0.008	0.510	0.012

Table 5: Average performance and generalizability scores (Equations 1 and 2, respectively) standard error of the mean for the 7 algorithms in the Pareto-optimal set and SAC when using RWRL Walker as a training environment. We compute these metrics across 10 seeds.

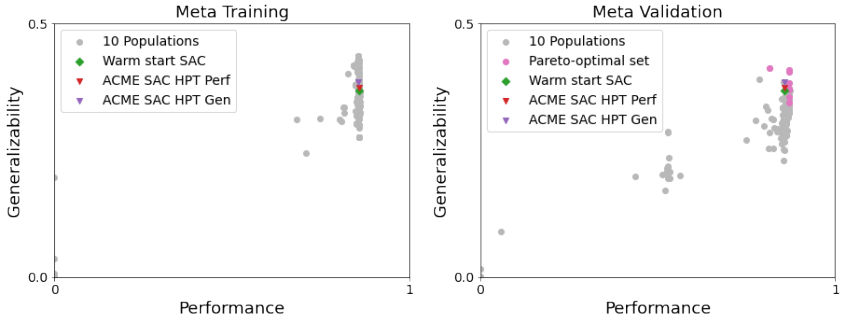


Figure 9: Meta training and meta validation stability-adjusted fitness scores (computed using Equation 3 across 10 seeds) for each RL algorithm in the population alongside the warm-start (SAC) and ACME SAC when using the Gym Pendulum environment for training. We show the meta validated Pareto-optimal set of algorithms that results after merging the 10 populations corresponding to the 10 repeats of the experiment.

G.4 Stability analyses

We present the stability results, which are accounted for by penalizing the standard deviation across seeds, following Equation 3. For each environment considered in this work, we select a subset of the meta validated graphs that covers all the explored fitness space and, in Figure 11, show the average

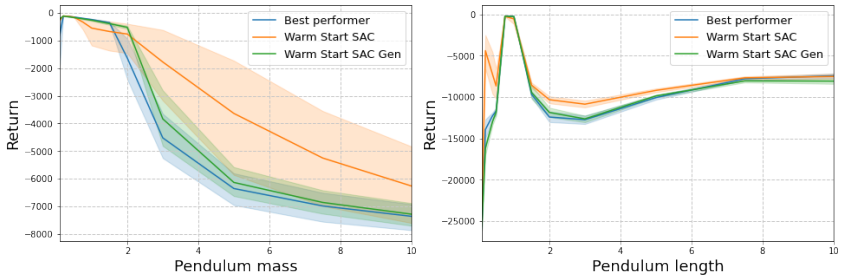
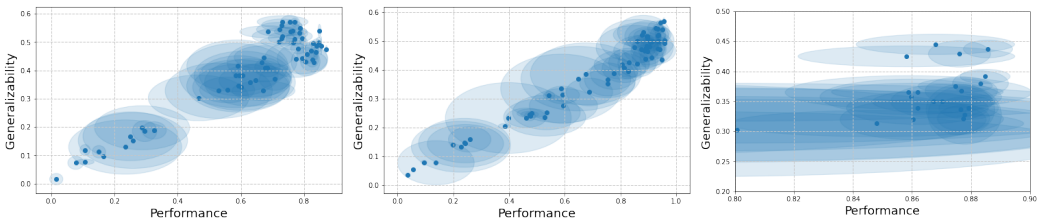


Figure 10: Average and standard deviation across seeds of the meta validation performance of the best performer, the best generalizer, and the warm-start (SAC) when training on a single configuration of Gym Pendulum and evaluating on multiple unseen ones. The pendulum mass and the pendulum length independently change across environment configurations (we change one at a time). The training configurations use a pendulum mass and a pendulum length of 1.0 and 1.0, respectively.

RL Algorithm	**Avg. Perf. score** (f_{perf})		**Avg. Gen. score** (f_{gen})	
Pareto 1: Best performer	0.887	0.010	0.360	0.011
Pareto 2	0.885	0.009	0.381	0.017
Pareto 3	0.887	0.010	0.391	0.014
Pareto 4	0.887	0.011	0.392	0.013
Pareto 5	0.887	0.011	0.393	0.007
Pareto 6	0.886	0.010	0.433	0.018
Pareto 7	0.886	0.011	0.437	0.019
Pareto 8: Best generalizer	0.868	0.034	0.445	0.021
warm-start SAC	0.879	0.022	0.383	0.015
ACME SAC HPT Perf	0.880	0.014	0.392	0.012
ACME SAC HPT Gen	0.879	0.014	0.400	0.009

Table 6: Average performance and generalizability scores (Equations 1 and 2, respectively) standard error of the mean for the 8 algorithms in the Pareto-optimal set and SAC when using Gym Pendulum as a training environment. We compute these metrics across 10 seeds.

and standard deviation of each fitness score. Algorithms in the Pareto-optimal set and those closer to it present lower variability, showing MetaPG is also successful in improving the stability of RL algorithms.



(a) RWRL Cartpole (b) RWRL Walker (c) Gym Pendulum

Figure 11: From a subset of the meta validated graphs, for each of them, we show the average fitness scores surrounded by an ellipse with semiaxes representing the standard deviation across seeds for each fitness score.

G.5 Best performer and best generalizer for RWRL Walker and Gym Pendulum

We present the loss equations for both the best performer and best generalizer when using RWRL Walker and Gym Pendulum as training environments. First, the best performer for RWRL Walker:

$$L^{perf} = \mathbb{E}_{(s_t;a_t;r_t;s_{t+1}) \sim D}\left[r_t + \gamma\left(\min_i Q_{targ_i}(s_{t+1}, \tilde{a}_{t+1}) \quad \text{atan}(\gamma/Q(s_t, a_t))\right) \quad Q(s_t, \tilde{a}_t)\right] \tag{10}$$

$$L^{perf}_{Q_i} = \mathbb{E}_{(s_t;a_t;r_t;s_{t+1}) \sim D}\left[\left(r_t + \gamma\left(\min_i Q_{targ_i}(s_{t+1}, \tilde{a}_{t+1}) \quad \text{atan}(\gamma/Q_i(s_t, a_t))\right) \quad Q_i(s_t, a_t)\right)^2\right] \tag{11}$$

In all cases, $\tilde{a}_t \sim \pi(\cdot|s_t)$, $\tilde{a}_{t+1} \sim \pi(\cdot|s_{t+1})$, and D is a dataset of experience tuples from the replay buffer. Next, the best generalizer for RWRL Walker:

$$L^{gen} = \mathbb{E}_{(s_t;a_t;s_{t+1}) \sim D}\left[\frac{0.2 \quad \log \pi(\tilde{a}_{t+1}|s_{t+1})}{Q_i(s_{t+1}, \tilde{a}_{t+1}) \quad 0.1 \quad \log \pi(\tilde{a}_{t+1}|s_{t+1})} \quad \min_i Q_i(s_t, \tilde{a}_{t+1})\right] \tag{12}$$

$$L^{gen}_{Q_i} = \mathbb{E}_{(s_t;a_t;r_t;s_{t+1}) \sim D}\left[\left(r_t + \gamma\left(Q_i(s_{t+1}, \tilde{a}_{t+1}) \quad 0.1 \quad \log \pi(\tilde{a}_{t+1}|s_{t+1})\right) \quad Q_i(s_t, a_t)\right)^2\right] \tag{13}$$

Now we present the best performer for Gym Pendulum:

$$L^{perf} = \mathbb{E}_{(s_t;a_t) \sim D}\left[2 \quad \text{atan}(\log \pi(\tilde{a}_t|s_t)) \quad \min_i Q_i(s_t, \tilde{a}_t)\right] \tag{14}$$

$$L^{perf}_{Q_i} = \mathbb{E}_{(s_t;a_t;r_t;s_{t+1}) \sim D}\left[\left(r_t + \gamma\left(Q_{targ_i}(s_{t+1}, \tilde{a}_t) \quad \log \pi(\tilde{a}_t|s_t)\right) \quad Q_i(s_t, a_t)\right)^2\right] \tag{15}$$

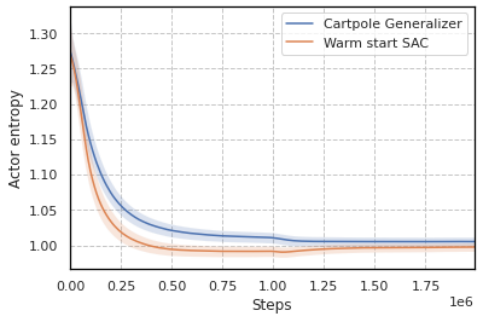
Finally, the equations for the best generalizer when using Gym Pendulum are:

$$L^{gen} = \mathbb{E}_{(s_t;a_t) \sim D}\left[\log(\log \pi(\tilde{a}_t|s_t)) \quad \min_i Q_i(s_t, \tilde{a}_t)\right] \tag{16}$$

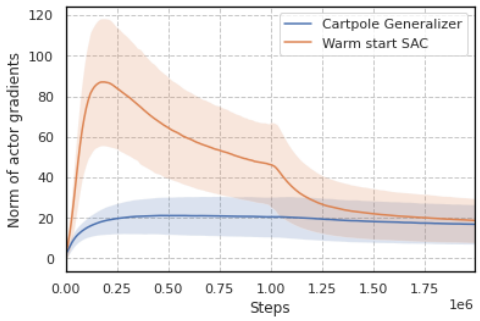
$$L^{gen}_{Q_i} = \mathbb{E}_{(s_t;a_t;r_t;s_{t+1}) \sim D}\left[\left(r_t + \gamma\left(Q_{targ_i}(s_{t+1}, \tilde{a}_t) \quad \log(\log \pi(\tilde{a}_t|s_t))\right) \quad Q_i(s_t, a_t)\right)^2\right] \tag{17}$$

G.6 Additional analysis on evolved algorithms for RWRL Cartpole

Figure 12 shows the entropy and norm of the gradients of the actor for the RWRL Cartpole best generalizer. We also show these same metrics for the warm-start algorithm. In both cases, we let the agents train for more episodes than those in the experimental setup.



(a) Average entropy of the policy during training for RWRL Cartpole.



(b) Average gradient norm of the actor loss during training for RWRL Cartpole.

Figure 12: Analysis of the entropy and gradient norm of the actor when evaluating the best generalizer from RWRL Cartpole in comparison to the warm-start.

G.7 Hyperparameter tuning for transfer and benchmark

Once an evolution experiment is over and the evolved algorithms are meta-validated, we compare them against: 1) ACME SAC (Hoffman et al., 2020), and 2) other RL algorithms that have been

evolved in a different environment. To that end, for each ACME benchmark and evolved algorithm transfer, we tune the hyperparameters of the algorithms. Since we consider two fitness scores in this work (performance and generalizability), we select the two hyperparameter configurations that lead to the best performance and best generalizability scores, respectively. We denote these two configurations as the best performer and best generalizer, respectively. To that end, we do a grid search across the sets of hyperparameters listed in Table 7.

Hyperparameter	**Values**
Discount factor γ	0.9, 0.99, 0.999
Batch size	32, 64, 128
Learning rate	1 10 ^{4}, 3 10 ^{4}, 1 10 ^{3}
Target smoothing coeff. τ	0.005, 0.01, 0.05
Reward scale	0.1, 1.0, 5.0, 10.0

Table 7: Hyperparameter values considered during the tuning process.

This process is only carried out once the evolution is over; the warm-start algorithm is not hyperparameter-tuned before evolution.

G.8 TRANSFERRING THE EVOLVED ALGORITHMS

We present the results of carrying out transfer experiments in which we take the best performer and best generalizer obtained after evolving in a specific environment and test them in the other two environments considered in this work. To that end, we follow the hyperparameter tuning procedure described above and therefore, for each different RL algorithm, we obtain the hyperparameter configurations that leads to the best performance and best generalizability, respectively. For example, taking the best performer from RWRL Walker (Walker Perf.) and testing it on RWRL Cartpole leads to two sets of fitness scores (best performer and best generalizer). The transfer results for RWRL Cartpole, RWRL Walker, and Gym Pendulum can be observed in Tables 8, 9, and 10, respectively.

	Evaluation and tuning environment: RWRL Cartpole							
	Best performance				**Best generalizability**			
RL Algorithm	f_{perf}		f_{gen}		f_{perf}		f_{gen}	
Cartpole	0.871	0.003	0.475	0.016	0.770	0.014	0.570	0.019
Walker Perf.	0.849	0.010	0.444	0.041	0.826	0.024	0.456	0.041
Walker Gen.	0.670	0.094	0.374	0.039	0.670	0.094	0.374	0.039
Pendulum Perf.	0.857	0.001	0.502	0.024	0.851	0.005	0.535	0.012
Pendulum Gen.	0.829	0.025	0.489	0.051	0.766	0.079	0.509	0.040
ACME SAC	0.865	0.001	0.372	0.060	0.845	0.012	0.518	0.040

Table 8: Transfer results (average fitness standard error of the mean) on RWRL Cartpole. The row highlighted in gray corresponds to the results of the evolution experiment in RWRL Cartpole. The rest correspond to the best performance and best generalizability configurations that result from doing hyperparameter tuning to the best performer and best generalizer evolved in different environments.

	Evaluation and tuning environment: RWRL Walker							
	Best performance				Best generalizability			
RL Algorithm	f_{perf}		f_{gen}		f_{perf}		f_{gen}	
Cartpole Perf.	0.959	0.004	0.502	0.025	0.958	0.006	0.533	0.012
Cartpole Gen.	0.031	0.002	0.032	0.001	0.027	0.003	0.035	0.001
Walker	0.963	0.002	0.544	0.018	0.955	0.005	0.569	0.015
Pendulum Perf.	0.611	0.145	0.296	0.066	0.611	0.145	0.296	0.066
Pendulum Gen.	0.929	0.024	0.510	0.045	0.927	0.024	0.518	0.035
ACME SAC	0.968	0.003	0.444	0.014	0.926	0.008	0.510	0.012

Table 9: Transfer results (average fitness standard error of the mean) on RWRL Walker. The row highlighted in gray corresponds to the results of the evolution experiment in RWRL Walker. The rest correspond to the best performance and best generalizability configurations that result from doing hyperparameter tuning to the best performer and best generalizer evolved in different environments.

	Evaluation and tuning environment: Gym Pendulum							
	Best performance				Best generalizability			
RL Algorithm	f_{perf}		f_{gen}		f_{perf}		f_{gen}	
Cartpole Perf.	0.875	0.013	0.352	0.023	0.874	0.017	0.364	0.015
Cartpole Gen.	0.843	0.022	0.337	0.014	0.843	0.022	0.337	0.014
Walker Perf.	0.873	0.013	0.342	0.018	0.843	0.030	0.395	0.020
Walker Gen.	0.864	0.015	0.349	0.030	0.754	0.075	0.401	0.013
Pendulum	0.887	0.010	0.360	0.011	0.868	0.034	0.445	0.021
ACME SAC	0.879	0.014	0.392	0.012	0.879	0.014	0.400	0.009

Table 10: Transfer results (average fitness standard error of the mean) on Gym Pendulum. The row highlighted in gray corresponds to the results of the evolution experiment in Gym Pendulum. The rest correspond to the best performance and best generalizability configurations that result from doing hyperparameter tuning to the best performer and best generalizer evolved in different environments.