Dynamic Quality-Diversity Search

Roberto Gallotta
University of Malta
Msida, Malta
roberto.gallotta@um.edu.mt

Antonios Liapis
University of Malta
Msida, Malta
antonios.liapis@um.edu.mt

Georgios N. Yannakakis
University of Malta
Msida, Malta
georgios.yannakakis@um.edu.mt

ABSTRACT

Evolutionary search via the quality-diversity (QD) paradigm can discover highly performing solutions in different behavioural niches, showing considerable potential in complex real-world scenarios such as evolutionary robotics. Yet most QD methods only tackle static tasks that are fixed over time, which is rarely the case in the real world. Unlike noisy environments, where the fitness of an individual changes slightly at every evaluation, dynamic environments simulate tasks where external factors at unknown and irregular intervals alter the performance of the individual with a severity that is unknown a priori. Literature on optimisation in dynamic environments is extensive, yet such environments have not been explored in the context of QD search. This paper introduces a novel and generalisable Dynamic QD methodology that aims to keep the archive of past solutions updated in the case of environment changes. Our Dynamic QD intervention is applied on MAP-Elites and CMA-ME, two powerful QD algorithms, and we test their performance on a dynamic variant of the well-known lunar lander environment.

ACM Reference Format:
Roberto Gallotta, Antonios Liapis, and Georgios N. Yannakakis. 2024. Dynamic Quality-Diversity Search. In *Genetic and Evolutionary Computation Conference (GECCO '24 Companion), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3638530.3654257

1 INTRODUCTION

Solving an optimisation problem consists in finding the best performing solution(s) in the domain of interest. While many problems are fixed, many others change in different ways over time. These Dynamic Optimisation Problems (DOPs) [14] are of high interest to the research community, as they mirror real-world problems where time affects performance of solutions [4], constraints [5], or even the bounds of the solution space available to the solving algorithm [1]. Evolutionary Dynamic Optimisation (EDO) is a subfield of dynamic optimisation that leverages techniques inspired by biological evolution [15] to solve DOPs. Changes in the environment can be (a)periodic, occur at a non-fixed rate, and impact with a different severity. Different methods have been introduced to detect such changes, such as re-evaluating certain solutions [14]. Diversity in the population is either enforced during the search [16] or introduced when an environment change has been detected [18].

As EDO algorithms rely on definitions of diversity based on the genotype of solutions, they may fail in environments where diversity itself is affected by time. Conversely, quality-diversity (QD) algorithms are a family of evolutionary algorithms that explicitly takes into account the behaviour of solutions in order to keep a diverse population [17]. QD has been extensively used to illuminate the behavioural space by subdividing it into *niches* [17]. Each solution in a QD problem has an associated fitness that tells the algorithm the *quality* of the solution, and a set of behavioural characteristics, or *BCs*, of the solution. A niche of the search space contains one or more solutions with BCs that fall within a specific interval. By keeping the best solutions (*elites*) based on fitness in niches, the *diversity* of the population, stored in an *archive*, is maintained over time. However, the fitness and BC measures of solutions are not always reliable: in real-world problems, their estimations can be *stochastic* [13] or *noisy* [6]. Prior work on QD applied to noisy domains [6] and with changing archives [7] showed promising results. Therefore, we believe QD algorithms provide a solid foundation to solve DOPs, with some critical adaptations.

In this paper, we propose a framework to adapt existing QD algorithms to solve DOPs, improving their performance over their static counterpart. Our Dynamic Quality-Diversity (D-QD) approach, explained in Section 2, can search in environments that change at an unknown frequency and with unknown severity, on either the fitness landscape or the behavioural mapping. In particular, we modify two well-established QD algorithms: the Multi-dimensional Archive of Phenotypic Elites (MAP-Elites), first introduced in [12], and CMA-ME [8], an adaptation of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [10] to MAP-Elites. Both algorithms work on a 2-dimensional archive of solutions (a *feature map*), where new solutions are added to the archive only when either the niche they belong to is unoccupied, or is occupied by a solution with a lower fitness. However, these algorithms differ in the way new solutions are generated. In MAP-Elites, parent solutions are randomly sampled from occupied cells in the archive, and offspring are generated by applying genetic operators. In CMA-ME, instead, offspring are generated from a mean vector and covariance matrix, following the CMA-ES approach.

The code for this paper is available at https://github.com/gallorob/dynamic-quality-diversity.

2 DYNAMIC QUALITY-DIVERSITY

Dynamic QD (D-QD) algorithms are extensions of QD algorithms that explicitly take into account the dynamic nature of the environment they operate in. In this work, two well-known QD algorithms, namely MAP-Elites and CMA-ME, have been modified to work in

GECCO '24 Companion, July 14–18, 2024, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0495-6/24/07.
https://doi.org/10.1145/3638530.3654257

Algorithm 1 Dynamic QD Search

Require: environment env, detection strategy d, re-evaluation strategy r, algorithm a, total timesteps T_{end}

$A = \emptyset$ ▷ Initialise the archive
while $t < T_{\text{end}}$ **do**
  $o \leftarrow a(A)$ ▷ Generate offspring, or new solutions if $A = \emptyset$
  $s_d \leftarrow d(A)$ ▷ Sample solutions s_d for detection
  if outdated$(s_d, env(s_d))$ **then** ▷ Detect environment shifts
    $s_r \leftarrow r(A)$ ▷ Sample solutions s_r for re-evaluation
    $A \leftarrow A \cup env(s_r)$ ▷ Update archive with re-evaluated solutions
  end if
  $A \leftarrow A \cup env(o)$ ▷ Attempt adding offspring to archive
  $env \leftarrow \text{next}(env, t)$ ▷ Update the environment to the next timestep
end while
return A

such a setting. The Dynamic MAP-Elites (D-MAP-Elites) and Dynamic CMA-ME (D-CMA-ME) algorithms do this by implementing two core components: (a) an environment shift detection; and (b) re-evaluation of solutions in the archive. We include the pseudocode for our D-QD algorithm in Algorithm 1.

As the environment may change at any time, the *environment shift detection* component is vital as it triggers re-evaluation of past solutions. Detection happens only from the perspective of the current archive: if a solution's objective score or BCs differ from the previously recorded value (stored in the archive), we assume that there was an environment shift. We do not allow any margin of error in the comparison; we detect a shift regardless of the magnitude of the change. We thus assume our environments are not noisy: alternatively, the comparison between previous and current values would need an additional slack hyperparameter before determining whether a shift has indeed occurred or if the difference was the result of a noisy evaluation of the solution.

To avoid re-evaluating the entire archive, we explore two methods for detecting environment shifts. The first method is based on "oldest solutions" (d_O), which selects individuals based on their age, assuming older solutions are more likely to be outdated. The second method instead selects "replacee" solutions (d_R). Replacees are solutions in the archive that would be compared against newly generated offspring. This is beneficial regardless, as ideally an offspring with up-to-date scores should not replace an elite with out-dated scores. In either case, selected solutions are re-evaluated and, if any of their properties are different from their recorded values, we assume an environment shift has occurred and perform additional steps in other D-QD components. In d_R, additionally, it is possible that re-evaluated elites have different BCs than initially recorded, which leads to comparisons against additional occupied cells. In this case, which we call *cascading re-evaluations*, the number of re-evaluations may be different from the number of offspring. Finally, unlike d_O, d_R doesn't require maintaining a list of recently evaluated elites as all potential replacees are re-evaluated.

The issue of re-evaluating the archive of solutions is critical to both the efficiency and the performance of the algorithm. Re-evaluating all elites upon detecting an environment shift ensures the most up-to-date solutions but is computationally expensive. In our study, instead, we only re-evaluate replacees (e_R) when an environment shift is detected. This approach ensures that new solutions replace existing elites whose properties may be outdated. If a shift is detected, replacees undergo re-evaluation, potentially triggering further evaluations if they move in the archive. We note that, when using d_R, replacees are re-evaluated once (combining shift detection and archive re-evaluation). If d_O is used, replacees still require re-evaluation upon shift detection. While using replacees for shift detection may be more computationally efficient, it lacks the control and reliability associated with evaluating oldest elites.

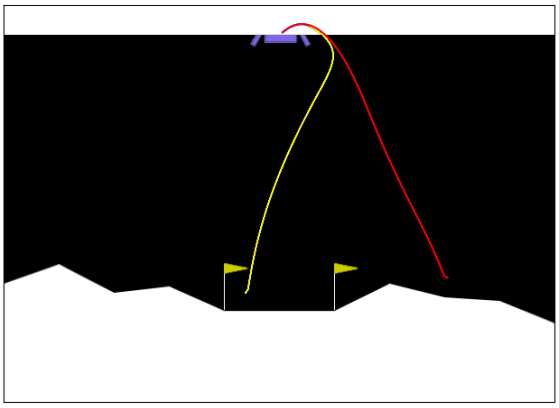

Figure 1: The complete trajectory of the same lander in the dynamic Lunar Lander environment, before an environment shift (yellow) and after (red).

3 EXPERIMENTAL PROTOCOL

We first validated our algorithm on a modified version of the simple sphere environment [8]. We adapted this environment by moving the sphere centre (i.e.: the global optimum) over time. This change affected the objective and BCs of solutions directly, giving us a perfect benchmark to validate our code implementation. Details of the environment and results from experiments can be found in [9].

In pursuit of a more challenging environment, we modified the well-known reinforcement learning physics-based *lunar lander* environment, available in the Gymnasium suite [21]. Here, a spaceship (lander) is tasked to safely land on a target location by controlling only the thrusters on the sides of the ship. The lander starts at a random angle and velocity. Each solution is a simple linear policy model, which we evolve to maximise the episodic fitness, i.e.: the cumulative reward it scores during the simulation. The BCs of a solution θ are also driven by the simulation: $BC_1(\theta) \in [-1, 1]$ is the x-coordinate of the lander position; and $BC_2(\theta) \in [-3, 0]$ is the y-coordinate of the lander velocity. Unlike the sphere environment, the objective score and BCs for the solutions are not modified; instead, the conditions of the simulation are modified dynamically and affect the QD components *indirectly*. We introduce dynamism in the environment by modifying the *wind strength* σ_w and *turbulence strength* σ_τ, keeping them clamped within the recommended values to ensure simulation stability. Fig. 1 demonstrates how a shift

in wind parameters (σ_w,σ_τ) affects the trajectory taken by the same lander. Every 10 iterations, σ_w and σ_τ are updated as described in Eq. (1) and Eq. (2), respectively.

$$\sigma_w \leftarrow \max(\min(\sigma_w + R \cdot U(0, \gamma_w), 20), 0) \quad (1)$$

$$\sigma_\tau \leftarrow \max(\min(\sigma_\tau + R \cdot U(0, \gamma_\tau), 2), 0), \quad (2)$$

where R is a value sampled from a Rademacher distribution (thus set to either +1 or −1), $U(0, x)$ is a value sampled from the uniform distribution between 0 and x, with γ_w and γ_τ being the shift strengths for each parameter. Based on preliminary experiments, we set the initial $\sigma_w = 10$ and $\sigma_\tau = 1$ before the first environment shift, and the shift strengths $\gamma_w = 0.15$ and $\gamma_\tau = 0.1$. QD experiments in the dynamic lunar lander environment use an archive size of 50×50. In part, the strength of the dynamic lunar lander environment is that its shifts are indirectly impacting the objective and BCs; at the same time, this means that there is less controllability compared to the dynamic sphere environment. To ensure changes in the environment were not chaotic, we analysed 1000 simulated shifts (following Equation (1) and Equation (2)), observing changes in QD properties. Results analysed with Pearson correlation (significance at $p < 0.05$) showed significant correlations between changes in σ_w and both objective ($\rho = 0.8$) and BC_1 ($\rho = -0.9$), while changes in σ_τ showed significant correlations with BC_2 ($\rho = -0.71$). This suggests a linear relationship across all QD components (f, BC_1, BC_2). Assessing the locality of changes, we used the mean Cosine Similarity Entropy [3] on the absolute changes when an environment shift occurs. We found that changes in either σ_w or σ_τ resulted in non-localised changes in both feature map and fitness landscape, i.e.: changes in the environment did affect all solutions indiscriminately.

Armed with a proper environment, we can proceed to test our Dynamic QD algorithm against two baselines: "No Updates" $\{d_\emptyset, e_\emptyset\}$, in which we never re-evaluate solutions, and "Update All" $\{d_R, e_\forall\}$, in which all solutions are re-evaluated whenever an environment shift is detected via replacees. We are interested in measuring the trade-off between how many solutions we can keep updated and how many additional re-evaluations are needed. To do so, it is necessary to compare the output of the algorithm against a perfect-information, perfectly up-to-date system, which we dub an *ideal archive*. The ideal archive is computed on every iteration, and contains only the *surviving* elites of the current archive after we re-evaluate all elites. We can then measure the survival rate ($\%_s$) as the ratio of elites that survive in the ideal archive over all elites in the algorithm's current archive. As all elites would survive if up-to-date, we want to maximise this metric (up to 100%). Then, we compute the Mean Squared Errors (MSEs) of the objective scores, BCs, and QD score of all surviving elites against their up-to-date values separately, avoiding aggregation biases. Low MSE values indicate closer alignment to the correct properties of each solution. Finally, to examine the computational efficiency of our approach, inspired by [22], we compute the Mean Evaluation Cost (MEC) and success rates. MEC measures the number of evaluations needed to reach a survival rate of 75% after each (known, not detected) environment shift, and the success rate measures the percentage of between-shifts intervals in which the survival rate threshold is reached. Ideally, an algorithm would have low MEC and high success rates, ensuring most solutions remain current in an efficient manner.

4 RESULTS

We report results of our experiments in Table 1. We test environment shifts detection with either oldest elites (d_O) or replacees (d_R), while re-evaluating only via replacees (e_R). Preliminary results reported in [9] indicated no clear advantage for other strategies. Results are averaged over 5 independent runs of each algorithm, using default hyperparameters provided by PyRibs [20]. We report the 95% confidence interval of these runs in Table 1. Finally, we also compare each variant against both baselines and the other variant via a Welch's T-Test, with significance established at $p < 0.05$.

From the presented results, we find that only the $\{d_O, e_R\}$ strategy applied to D-CMA-ME consistently outperforms the "No Updates" baseline, as well as outperforming the alternative $\{d_R, e_R\}$ variant. More specifically, $\{d_O, e_R\}$ greatly improves the survival rate in both algorithms, increasing it by 15% in D-MAP-Elites and by 34% in D-CMA-ME compared to $\{d_R, e_R\}$. For mean MSE values, we find that $\{d_O, e_R\}$ maintains more correct values for the surviving elites than $\{d_R, e_R\}$. While this difference is significant in D-CMA-ME, it also holds in D-Map-Elites without being statistically significant. Regarding computational efficiency, no variants are able to match the success rates of the "Update All" baseline. However, $\{d_O, e_R\}$ variants reach double and nearly 10 times the successes of $\{d_R, e_R\}$ variants for D-MAP-Elites and D-CMA-ME respectively, at less than twice the evaluation cost. From this analysis we can conclude that using D-QD can keep solutions up-to-date at a fraction of the computational effort of re-evaluating all solutions whenever an environment shift is detected. Detecting environment shifts via oldest elites may be slightly more computationally heavy, but keeps the archive more up-to-date.

We note that our analysis does not compare performance between D-MAP-Elites and D-CMA-ME. The main reason for this is that D-MAP-Elites worked on a fraction of the computational budget of D-CMA-ME, producing fewer offspring per generation. This allowed us to compare the dynamic variants within each algorithm as they would be implemented in their most standardised form.

Alongside the D-QD algorithms, we also presented a procedure to convert static domains into dynamic ones, as well as the necessary tools to analyse the resulting environments. This facilitates future research, as our findings are limited to the single environment we tested our D-QD algorithm on. Another limitation is that our framework does not handle noisy domains, which would impact its performance. Future work could explore memory strategies [19] to reintroduce older solutions and handle noisy environments effectively. Finally, the current paper uses performance metrics based on an ideal, up-to-date archive, while such tools are not available in real-world applications.

Despite these limitations, this project sets the foundation for more ambitious work where changes in the environment reflect real-world issues, such as machine deterioration [2] in evolutionary robotics and changes in user preferences in AI-assisted design tools [11]. Future work could focus on equipping algorithms with the ability to handle shifting designer goals and preferences implicitly, paving the way for more dynamic and adaptive systems.

Table 1: Performance metrics for D-MAP-Elites and D-CMA-ME on the dynamic Lunar Lander environment across 5 different runs. Metrics are the Survival rate ($\%_s$), error for the objective (MSE_{obj}), BCs (MSE_{BC_1} and MSE_{BC_2}), and QD-score (MSE_{QD}), averaged over all iterations, with 95% Confidence Interval, and MEC with survival threshold set to 75% along with ratio of successes over all actual environment shifts in parentheses. The D-QD variants $\{d_R, e_R\}$ and $\{d_O, e_R\}$, are tested against two baselines: "No Updates" as $\{d_\emptyset, e_\emptyset\}$ and "Update all" as $\{d_R, e_\forall\}$. Significantly better results between a variant and the "No Updates" baseline are shown with a †, between a variant and the "Update All" baseline are shown with a ‡, and between the two variants are shown with a *.

Algorithm	Strategy	$\%_s \uparrow$	$MSE_{obj} \downarrow$	$MSE_{BC_1} \downarrow$	$MSE_{BC_2} \downarrow$	$MSE_{QD} \downarrow$	$\text{MEC}_{75\%} \downarrow (\uparrow)$
D-MAP-Elites	$d_\emptyset, e_\emptyset$	59±7	310±97	0.13±0.04	0.60±0.04	13728±5224	50±0.0 (22%±0.08%)
	$d_R, e_\forall$	99±0.2	25±9.2	0.003±0.001	0.07±0.02	98±35	522±18.3 (99±0.0%)
	d_R, e_R	63±6	274±78	0.09±0.02	0.61±0.04	10508±3862	$80\pm0.8^{\ddagger}$ (22±0.1%)
	d_O, e_R	$74\pm4^{\dagger}$	228±69	$0.05\pm0.01^{\dagger}$	0.54±0.04	5107±2519	$142\pm1.6^{\ddagger}$ $(47\pm0.17\%^{\dagger,\star})$
D-CMA-ME	$d_\emptyset, e_\emptyset$	44±8	203±44	0.20±0.06	0.70±0.06	34120±21689	400±0.0 (10±0.02%)
	$d_R, e_\forall$	100±0.0	0.08±0.04	0.0±0.0	0.0±0.0	0.8±1.1	1584±32.7 (99±0.0%)
	d_R, e_R	49±5	182±18.5	$0.09\pm0.02^{\dagger}$	0.66±0.03	27775±12487	$701\pm10.3^{\ddagger}$ (9±0.14%)
	d_O, e_R	$78\pm1^{\dagger,\star}$	$83\pm7^{\dagger,\star}$	$0.02\pm0.004^{\dagger,\star}$	$0.42\pm0.02^{\dagger,\star}$	$1401\pm216^{\dagger,\star}$	1062±9.1 $(88\pm0.02\%^{\dagger,\star})$

5 CONCLUSION

This paper introduced a novel approach to dynamic optimisation which leverages QD search, and applied it to MAP-Elites and CMA-ME algorithms. The proposed D-QD methodology was tested on the well-known lunar lander environment, which was adapted to be dynamic over time. The multiple experiments with different setups and baselines revealed a clear-cut optimal D-QD variant for the D-CMA-ME algorithm, relying on oldest elites for detecting shifts in the environment. However, the efficiency of D-QD search on different performance metrics highlights the trade-off between accuracy and computation costs. We expect that this work will inspire research on the optimisation of dynamic environments under the powerful lens of quality-diversity search.

ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 programme under grant agreement No 951911.

REFERENCES

[1] Manuel Blanco Abello, Lam Thu Bui, and Zbignew Michalewicz. 2011. An adaptive approach for solving dynamic scheduling with time-varying number of tasks — Part I. In *Proceedings of the IEEE Congress of Evolutionary Computation.*

[2] Luis E. Garza Castanon and Adriana Vargas Martinez. 2009. Artificial Intelligence Methods in Fault Tolerant Control. In *Automation and Control*, Aleksandar Rodic (Ed.). IntechOpen.

[3] Theerasak Chanwimalueang and Danilo P. Mandic. 2017. Cosine Similarity Entropy: Self-Correlation-Based Complexity Analysis of Dynamical Systems. *Entropy* 19, 12 (2017).

[4] Helen G. Cobb and John J. Grefenstette. 1993. Genetic Algorithms for Tracking Changing Environments. In *Proceedings of the International Conference on Genetic Algorithms.*

[5] Chenggang Cui, Tian Feng, Ning Yang, and Junfeng Chen. 2015. Memory Based Differential Evolution Algorithms for Dynamic Constrained Optimization Problems. In *Proceedings of the International Conference on Computational Intelligence and Security.*

[6] Manon Flageat and Antoine Cully. 2020. Fast and stable MAP-Elites in noisy domains using deep grids. In *Proceedings of the Artificial Life Conference.*

[7] Matthew C. Fontaine, Scott Lee, L. B. Soros, Fernando De Mesentier Silva, Julian Togelius, and Amy K. Hoover. 2019. Mapping Hearthstone deck spaces through MAP-elites with sliding boundaries. In *Proceedings of the Genetic and Evolutionary Computation Conference.*

[8] Matthew C. Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K. Hoover. 2020. Covariance matrix adaptation for the rapid illumination of behavior space. In *Proceedings of the Genetic and Evolutionary Computation Conference.*

[9] Roberto Gallotta, Antonios Liapis, and Georgios N. Yannakakis. 2024. Dynamic Quality-Diversity Search. *arXiv preprint arXiv:2404.05769* (2024).

[10] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation* 9, 2 (2001).

[11] Antonios Liapis, Gillian Smith, and Noor Shaker. 2016. Mixed-initiative Content Creation. In *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, Noor Shaker, Julian Togelius, and Mark J. Nelson (Eds.). Springer, 195–214.

[12] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909* (2015).

[13] Frank Neumann, Mojgan Pourhassan, and Vahid Roostapour. 2019. *Analysis of Evolutionary Algorithms in Dynamic and Stochastic Environments.* Springer International Publishing.

[14] Trung Thanh Nguyen. 2011. *Continuous dynamic optimisation using evolutionary algorithms.* Ph. D. Dissertation. University of Birmingham.

[15] Trung Thanh Nguyen, Shengxiang Yang, and Juergen Branke. 2012. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation* 6 (2012).

[16] Trung Thanh Nguyen, Shengxiang Yang, Juergen Branke, and Xin Yao. 2013. Evolutionary Dynamic Optimization: Methodologies. In *Evolutionary Computation for Dynamic Optimization Problems*, Shengxiang Yang and Xin Yao (Eds.).

[17] Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. 2016. Quality Diversity: A New Frontier for Evolutionary Computation. *Frontiers in Robotics and AI* 3 (2016).

[18] Hendrik Richter. 2010. Memory Design for Constrained Dynamic Optimization Problems. In *Proceedings of the Applications of Evolutionary Computation.*

[19] Hendrik Richter and Shengxiang Yang. 2009. Learning behavior in abstract memory schemes for dynamic optimization problems. *Soft Computing* 13, 12 (2009).

[20] Bryon Tjanaka, Matthew C Fontaine, David H Lee, Yulun Zhang, Nivedit Reddy Balam, Nathaniel Dennler, Sujay S Garlanka, Nikitas Dimitri Klapsis, and Stefanos Nikolaidis. 2023. Pyribs: A Bare-Bones Python Library for Quality Diversity Optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference.*

[21] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. 2023. Gymnasium.

[22] Georgios N. Yannakakis, John Levine, and John Hallam. 2007. Emerging Cooperation With Minimal Effort: Rewarding Over Mimicking. *IEEE Transactions on Evolutionary Computation* 11, 3 (2007).