Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Emergent Tangled Program Graphs in Multi-Task Learning

Stephen Kelly and **Malcolm I. Heywood**
Dalhousie University
{skelly,mheywood}@cs.dal.ca

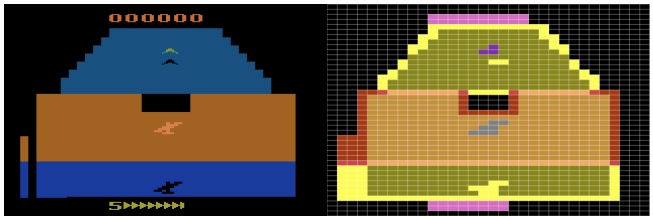
Abstract

We propose a Genetic Programming (GP) framework to address high-dimensional Multi-Task Reinforcement Learning (MTRL) through emergent modularity. A bottom-up process is assumed in which multiple programs self-organize into collective decision-making entities, or teams, which then further develop into multi-team *policy graphs*, or Tangled Program Graphs (TPG). The framework learns to play three Atari video games simultaneously, producing a single control policy that matches or exceeds leading results from (game-specific) deep reinforcement learning in each game. More importantly, unlike the representation assumed for deep learning, TPG policies start simple and adaptively complexify through interaction with the task environment, resulting in agents that are exceedingly simple, operating in real-time without specialized hardware support such as GPUs.

1 Introduction

Reinforcement learning (RL) for defining artificial agent behaviours represents one of the longest standing themes regarding the use of games as AI benchmarks [Yannakakis and Togelius, 2015]. In the most general case, agents are developed through direct interaction with game content as experienced by a human player, as opposed to requiring handcrafted (domain-specific) features to be defined a priori. With this goal in mind, a growing body of research has been employing a suite of Atari video games in which to demonstrate domain-independent RL [Mnih *et al.*, 2015; Hausknecht *et al.*, 2014; Machado *et al.*, 2017; van Steenkiste *et al.*, 2016; Naddaf, 2010]. In each case, the focus has been on developing game-specific agents for each game title. In this work, we investigate MTRL in the Atari environment. We demonstrate that in evolving solutions to multiple game titles simultaneously, agent behaviours for an individual game as well as single agents capable of playing *all* games emerge from the same evolutionary run. This paper is a condensed presentation of the work from [Kelly and Heywood, 2017b]. Additional MTRL results are available in [Kelly, 2018].

The Arcade Learning Environment (ALE) provides an Atari emulator geared towards benchmarking RL algorithms [Bellemare *et al.*, 2012]. The ALE is an interesting testbed because each game is unique and designed to be challenging for human players. Furthermore, artificial agents experience game-play just as a human would, via the a high-dimensional game screen (updated at 60Hz) and Atari joystick (18 discrete actions, or all possible combinations of paddle direction and fire button state, including 'no action'). As such, the ALE represents a *visual* RL domain. Moreover, Atari environments are considered partially observable because game entities often appear intermittently over sequential screen frames, causing visible flicker. As such, it is often impossible to observe the complete state of play from a single frame. Agents in this study stochastically skip frames with probability $p = 0.25$, with the previous action being repeated on skipped frames. This limits artificial agents to roughly the same reaction time as a human player and introduces stochasticity into the environment [Machado *et al.*, 2017].



(a) Atari Screen (b) Decimal State Variables

Figure 1: Screen quantization steps, reducing the raw Atari pixel matrix (a) to 1344 decimal state variables, visualized in (b).

Atari screen frames have a lot of redundant information. That is, visual game content is designed for maximizing entertainment, as opposed to simply conveying state, $\vec{s}(t)$, information. As such, we adopt a quantization approach to image preprocessing in which each frame, a 210×160 pixel matrix, is reduced to $42 \times 32 = 1,344$ decimal state variables in the range of $0 - 255$, visualized in Figure 1(b) for the game Zaxxon at time step (frame) t. Note that no feature extraction or game-specific variables are included in the final state representation, just a quantization of the original frame data.[1]

[1]Experiments with no input quantization, i.e. agents operating from the raw frame buffer, are reported in [Kelly *et al.*, 2018].

2 Evolving Tangled Program Graphs

2.1 Teams of Programs

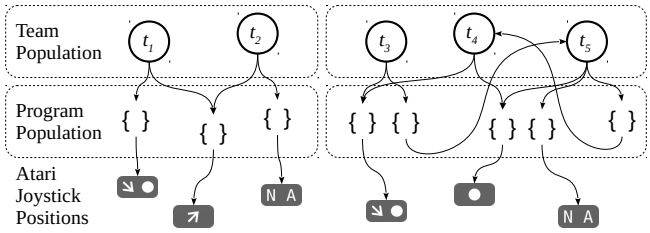
TPG is an extension of the Symbiotic Bid-Based (SBB) algorithm for evolving teams of programs [Lichodzijewski and Heywood, 2010]. SBB has been shown to build strong policies for a variety of reinforcement learning tasks [Doucette *et al.*, 2012; Kelly *et al.*, 2012; Lichodzijewski and Heywood, 2011; Kelly and Heywood, 2014; 2015]. A single team of programs represents the simplest stand-alone decision-making entity in the framework, in which a linear program representation is assumed [Brameier and Banzhaf, 2007]. Specifically, programs are simple register machines which may include: 1) two-argument instructions of the form $R[i] \leftarrow R[x] \circ R[y]$ where $\circ \in \{+, -, \times, \div\}$; 2) single-argument instructions of the form $R[i] \leftarrow \circ(R[y])$ where $\circ \in \{cos, ln, exp\}$; and 3) a conditional statement of the the form IF $(R[i] < R[y])$ THEN $R[i] \leftarrow -R[i]$. $R[i]$ is a reference to an internal register, while $R[x]$ and $R[y]$ may reference internal registers or state variables. Determining which state variables are actually used in the program, as well as the number of instructions and their operations, are both emergent properties of the evolutionary process.

Each program defines the context for one discrete action (e.g. Atari joystick position), where actions are assigned to the program at initialization and potentially modified by variation operators during evolution. In order to map a state observation to an action, each program in the team will execute relative to the current state, $\vec{s}(t)$, and return a single real valued 'bid', i.e. the content of register $R[0]$ after execution. The team then chooses the action of the program with the highest bid. As such, decision-making is an explicitly cooperative (group) behaviour.

Team development is driven by a generational genetic algorithm in which teams and programs are stored in separate populations and coevolved, Figure 2(a). Team (i.e. group) fitness is defined by a task-dependent objective (e.g. Atari game score). A fixed number of the least fit teams are deleted in each generation and replaced by sampling, cloning, and modifying surviving teams. Team variation operators may add/remove programs or modify the instruction set or action pointer of individual programs within a team. Naturally, if there is a performance benefit in smaller/larger teams and/or different program complements, this will be reflected in the surviving populations, i.e. team–program complexity is a developmental trait.

2.2 Policy Graphs

Programs are initialized with *atomic* actions defined by the task environment (i.e. Joystick positions, Figure 2(a)). In order to enable the evolution of hierarchically organized code under a completely open-ended process of evolution (i.e. emergent modularity [Nolfi, 1997]), program variation operators are allowed to introduce actions that index other teams within the team population. When a program's action is modified, it has an equal probability of referencing either an atomic action or another team. Thus, variation operators have the ability to *incrementally* construct multi-team *policy graphs*, Figure 2(b). Each vertex in the graph is a team, while



(a) Initial Populations (b) Emergence of Policy Graphs

Figure 2: Illustration of the relation between team and program populations at initialization (a) and during evolution as arbitrarily deep/wide *tangled program graphs* emerge (b).

each team member, or program, represents one outgoing edge leading either to another team or an atomic action. Naturally, decision-making in a policy graph begins at the root team (e.g. t_3 in Figure 2(b)), where each program in the team will produce one bid relative to the current state observation. Graph traversal then follows the program / edge with the largest bid, repeating the bidding process for the *same* state, $\vec{s}(t)$, at every team / vertex along the path until an atomic action is reached. Thus, in sequential decision-making tasks, the policy graph computes *one* path from root to action at every time step, where only a subset of programs in the graph (i.e those in teams along the path) require execution.

In summary, the critical idea behind TPG is that SBB [Lichodzijewski and Heywood, 2010] is extended to allow policy graphs to emerge, defining the inter-relation between teams. As programs composing a team typically index different subsets of the state space (i.e., the screen), the resulting policy graph will incrementally adapt, indexing more or less of the state space *and* defining the types of decisions made in different regions.

3 Empirical Experiment

Previous studies have established the ability of TPG to build game-specific controllers for 20 unique Atari games [Kelly and Heywood, 2017a]. The goal of this work is to produce both game-specific *and* multi-task policies from the *same* evolutionary run. We define two groups of games to be learned simultaneously, each containing 3 games for which (game-specific) TPG policies matched or exceeded test scores from deep reinforcement learning (DQN from [Mnih *et al.*, 2015]). The two groupings are not based on any intuition regarding multi-task compatibility. Game group A includes Centipede, Frostbite, and Ms. Pac-Man, three games with no obvious commonalities. Centipede is a vertically oriented shooting game, Frostbite is an adventure game, and Ms. Pac-Man is a maze task. Game group B contains Asteroids, Battle Zone, and Zaxxon, all shooting games in which the player gains points by aiming and firing a gun at on-screen targets. Note, however, that their similarities are relatively superficial, as each game title defines its own graphical environment, colour scheme, physics, objective(s), and scoring scheme. Furthermore, joystick actions are not necessarily correlated between game titles. For example, in Asteroids the 'down' action causes the spaceship avatar to enter hyperspace, disap-

pearing and reappearing at a random screen location. In Battle Zone, the 'down' action causes the first-person tank avatar to reverse, and foreground targets shrink, appearing to retreat into the distance. In Zaxxon, a third-person plane-flying / shooting game, the 'down' action is interpreted as 'pull-up', causing the plane to move vertically up the screen.

3.1 Experimental Setup

Five runs of TPG are conducted for 200 (300) generations in game group A (B). In order to support the development of multi-task policies, task switching is introduced such that the population is exposed to different game titles over the course of evolution. Thus, for each consecutive block of 10 generations, one game title is selected with uniform probability to be the *active* title. Each policy graph is evaluated in 5 episodes per generation under the active title, up to a lifetime maximum of 5 evaluations under each game title.

Thus, each policy stores a historical record of up to 15 evaluations (5 in each of the 3 titles). However, a policy's fitness in any given generation is its average score over 5 episodes in the *active title only*. Thus, selective pressure is only explicitly applied relative to a single game title. However, stochastically switching the active title at regular intervals throughout evolution implies that a policy's long-term survival is dependent on a level of competence in *all* games. Finally, to facilitate the development of multi-task policies, the single best policy graph for each game title (i.e. that with the highest mean score over 5 games) is protected from deletion, a simple form of elitism that ensures the population as a whole never entirely 'forgets' any game.

3.2 Test Results

Once for each block of 10 generations associated with the current active game title, all policies are tested in 30 episodes under *all* titles as per established test conditions [Mnih *et al.*, 2015; Kelly and Heywood, 2017a]. Figure 3 plots these test results for each game group, where subplots (a) and (c) report the best score achieved in each title by *any* policy, and subplots (b) and (d) report the best *multi-task* policy scores (i.e. the scores are from the same policy graph). Scores are normalized relative to DQN's score (from [Mnih *et al.*, 2015]) in the same game (100%) and random play (0%). Normalized score is calculated as 100 $\times$ (TPG score - random play score)/(DQN score - random play score). Normalizing scores makes it possible to plot TPG's progress relative to all games together regardless of the scoring scheme in different games, and facilitates making a direct comparison with DQN.

The Group A experiment produces one game-specific policy for each game that ultimately exceeds the level of DQN, Figure 3(a). Surprisingly, in the case of Frostbite and Centipede, TPG began with initial policies (i.e. generation 1, prior to any learning) that exceeded the level of DQN. In Centipede, this initial policy was degenerate, selecting the 'up-right-fire' action in every frame, but nonetheless accumulating a score of 12,890. While completely un-interesting, the strategy managed to exceed the test score of DQN (8,390) *and* the reported test score for a human professional video game tester (11,963). From this starting point, the single best policy in Centipede improves throughout evolution to become

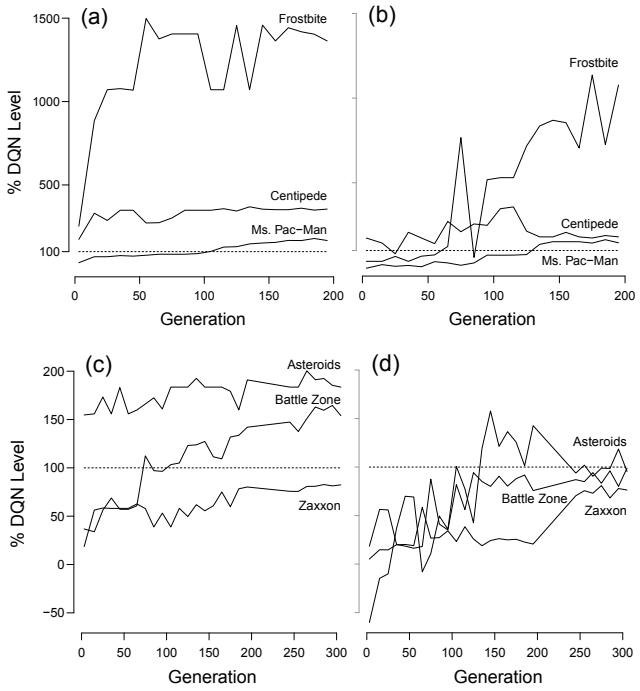


Figure 3: Incremental test results. See text for details.

more responsive and interesting; maintaining the strategy of shooting from the far right while gaining the ability to avoid threats through vertical and horizontal movement. Likewise, the single best policy in Frostbite begins at a high level of play relative to DQN, and significantly improves throughout evolution. From the group B experiment, the game-specific Asteroids policy begins at a high level of play relative to DQN, while game-specific champions in Battle Zone and Zaxxon slowly emerge during evolution, Figure 3(c).

Interestingly, the final champion policy in Frostbite defined a relatively long-term strategy, which involved building an igloo by jumping on horizontally-moving icebergs for an extended period of play ($\approx$ 1000 frames). When the igloo is complete, the agent promptly navigates to the entrance and enters the igloo (a trajectory consuming $\approx$ 200 frames), advancing to the next level. A long-term strategy also emerges for Ms. Pac-Man, in which the agent navigates directly to a power pill and eats it. Ghosts, which are normally threats to Ms. Pac-Man, become temporarily edible after a power pill is eaten. Thus, after eating the pill, the Ms. Pac-Man agent moves throughout the same section of maze, gaining points by eating ghosts.

Figures 3(b) and 3(d) report test scores for the best *multi-task* policy from each game group. While no single policy is initially capable of playing all three games at a level equivalent to DQN, a competent multi-task policy emerges by generation $\approx$ 125 (Group A) and $\approx$ 250 (Group B). Table 1 reports final TPG test scores along with DQN scores from [Mnih *et al.*, 2015].

By compartmentalizing decision-making over multiple independent modules (teams), and incrementally combining modules into policy graphs, three critical benefits are

Game	DQN	TPG-ST	TPG-MT	Tms	Ins	%IP
Centipede	8309 (±5237)	30689 (±734)	14683 (±1809)	3	588	17
Frostbite	328 (±251)	3836 (±243)	3092 (±416)	4	844	24
Ms. Pac-Man	2311 (±525)	4127 (±230)	3988 (±598)	5	1064	29
Asteroids	1629 (±542)	2389 (±1044)	1431 (±409)	5	999	27
Battle Zone	26300 (±7725)	40933 (±9892)	23700 (±7566)	6	1211	31
Zaxxon	4977 (±1235)	4484 (±1677)	4364 (±1780)	6	1244	32

Table 1: TPG-ST reports the best mean test score for each game achieved by any policy (i.e. each score is from a different policy graph). TPG-MT reports the best mean score for each game from a single multi-task policy graph. Complexity of the TPG-MT policy is also reported. The cost of making each decision is relative to the average number of teams visited (Tms), average number of instructions executed (Ins), and proportion of state space indexed (%IP).

achieved:

1) **Adaptive Complexity**. The number and complement of programs per team and teams per policy graph is an emergent property driven by environmental interaction. That is, policies are initialized in their simplest form and only complexify when/if simpler solutions are outperformed (see [Kelly and Heywood, 2017a] for empirical evidence of this property);

2) **State Space Selectivity**. Each program indexes a small proportion of the state space. As the the number of teams and programs in each policy graph increases, the policy will index more of the state space *and* optimize the decisions made in each region. However, each decision requires traversing a single path from root node to atomic action. As such, while the decision-making capacity of the policy graph expands through environment-driven complexification, the *cost* of making each decision, as measured by the number of programs which require execution, remains relatively low.

Figure 4 quantifies these first two properties by examining, for the best multi-task policy throughout evolution from Group A (Figure 3(b)), the number of teams per policy vs. teams visited per decision (Figure 4(a)) and the proportion of input space covered by the policy as a whole vs. the proportion indexed per decision (Figure 4(b)). Table 1 summarizes this data for the best multi-task policy from each game group. The run-time efficiency of policy graphs is a factor of how many instructions are executed to make each decision, ranging from 1064 in Ms. Pac-Man to 588 in Centipede for the Group A multi-task champion. For perspective, DQN performs millions of weight computations for each decision and defines the architecture a priori, using the same level of complexity for each game.

3) **Modular Task Decomposition**. As TPG policy graphs develop, they will subsume an increasing number of standalone modules (teams) into a hierarchical decision-making structure, or policy graph. Importantly, policy graphs are developed from the bottom up such that only root teams are subject to modification by variation operators. Thus, teams that are subsumed as interior nodes of a policy graph undergo no modification. This property allows a policy graph to avoid (quickly) unlearning tasks that were experienced in the past under task switching but are not currently the active task. This represents an alternative approach to avoiding "catastrophic forgetting" [Kirkpatrick *et al.*, 2016] during the continual, sequential learning of multiple tasks. The degree to which individual teams specialize relative to each objective experienced during evolution, i.e. the 3 game titles, can be characterized by looking at which teams contribute to decision-making at least once during test, relative to each game title.

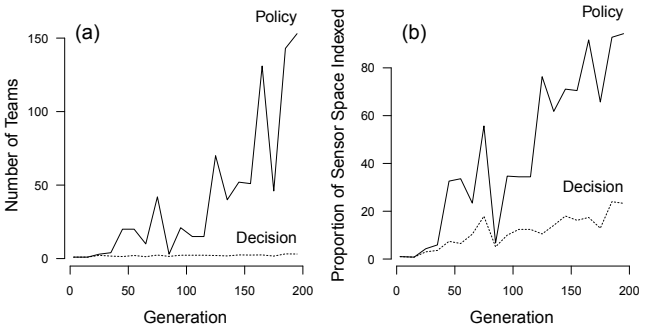


Figure 4: Complexity of best multi-task policy graph from Group A (Figure 3(b)). See text for details.

Figure 5 shows a multi-task TPG policy graph from generation 175 of the group A experiment. The Venn diagram indicates which teams are visited at least once while playing each game, over all test episodes. Naturally, the root team contributes to every decision (node labelled ABC, center of Venn diagram). 7 teams contribute to playing both Ms. Pac-Man and Frostbite (nodes labelled BC), while the rest of the teams specialize for a specific game title. In short, both generalist and specialist teams appear within the same policy and *collectively* define a policy capable of playing multiple game titles.

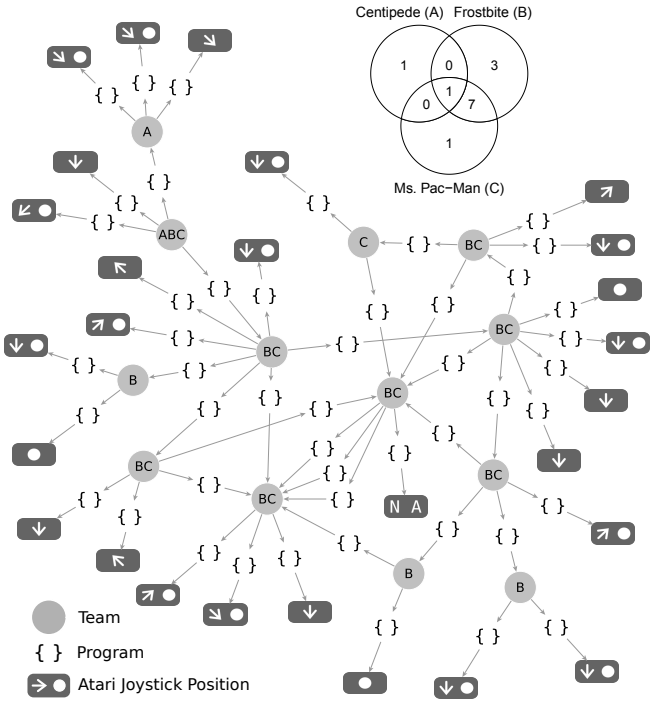


Figure 5: Example multi-task TPG policy graph from group A.

4 Conclusion

The simultaneous evolution of single-task *and* multi-task agent behaviours is demonstrated in the Atari environment. To do so, a new approach is proposed for providing emergent modularity in GP, or Tangled Program Graphs. The resulting agents match or exceed current state-of-the-art from (single-task) deep learning, while not requiring any more training resource than necessary for developing agent's under a single game title. Moreover, TPG agents are particularly elegant, supporting real-time operation without specialized hardware.

References

[Bellemare *et al.*, 2012] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2012.

[Brameier and Banzhaf, 2007] Markus Brameier and Wolfgang Banzhaf. *Linear Genetic Programming*. Springer, 1st edition, 2007.

[Doucette *et al.*, 2012] John A. Doucette, Peter Lichodzijewski, and Malcolm I. Heywood. Hierarchical task decomposition through symbiosis in reinforcement learning. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 97–104, 2012.

[Hausknecht *et al.*, 2014] Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. A neuroevolution approach to general Atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366, 2014.

[Kelly and Heywood, 2014] Stephen Kelly and Malcolm I. Heywood. On diversity, teaming, and hierarchical policies: Observations from the keepaway soccer task. In *European Conference on Genetic Programming*, volume 8599 of *LNCS*, pages 75–86. Springer, 2014.

[Kelly and Heywood, 2015] Stephen Kelly and Malcolm I. Heywood. Knowledge transfer from keepaway soccer to half-field offense through program symbiosis: Building simple programs for a complex task. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 1143–1150, 2015.

[Kelly and Heywood, 2017a] Stephen Kelly and Malcolm I. Heywood. Emergent tangled graph representations for Atari game playing agents. In *European Conference on Genetic Programming*, volume 10196 of *LNCS*, pages 64–79, 2017.

[Kelly and Heywood, 2017b] Stephen Kelly and Malcolm I. Heywood. Multi-task learning in atari video games with emergent tangled program graphs. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, 2017.

[Kelly *et al.*, 2012] Stephen Kelly, Peter Lichodzijewski, and Malcolm I. Heywood. On run time libraries and hierarchical symbiosis. In *IEEE Congress on Evolutionary Computation*, pages 3245–3252, 2012.

[Kelly *et al.*, 2018] Stephen Kelly, Robert Smith, and Malcolm I. Heywood. Emergent policy discovery for visual reinforcement learning through tangled program graphs: A tutorial. In *Genetic Programming Theory and Practice XVI*. Springer, 2018.

[Kelly, 2018] Stephen Kelly. *Scaling genetic programming to challenging reinforcement tasks through emergent modularity*. PhD thesis, Faculty of Computer Science, Dalhousie University, 2018.

[Kirkpatrick *et al.*, 2016] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *arXiv preprint 1612.00796*, 2016.

[Lichodzijewski and Heywood, 2010] Peter Lichodzijewski and Malcolm I. Heywood. Symbiosis, complexification and simplicity under GP. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference*, pages 853–860, 2010.

[Lichodzijewski and Heywood, 2011] Peter Lichodzijewski and Malcolm I. Heywood. The Rubik cube and GP temporal sequence learning: an initial study. In *Genetic Programming Theory and Practice VIII*, chapter 3, pages 35–54. Springer, 2011.

[Machado *et al.*, 2017] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *arXiv preprint 1709.06009*, 2017.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel. Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Naddaf, 2010] Yavar Naddaf. *Game-independent AI agents for playing Atari 2600 console games*. Masters thesis, University of Alberta, 2010.

[Nolfi, 1997] Stefano Nolfi. Using emergent modularity to develop control systems for mobile robots. *Adaptive behavior*, 5(3-4):343–363, 1997.

[van Steenkiste *et al.*, 2016] Sjoerd van Steenkiste, Jan Koutník, Kurt Driessens, and Jürgen Schmidhuber. A wavelet-based encoding for neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 517–524, 2016.

[Yannakakis and Togelius, 2015] Georgios N. Yannakakis and Julian Togelius. A panorama of artificial and computational intelligence in games. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(4):317–335, 2015.