# AlphaStar: An Evolutionary Computation Perspective

Kai Arulkumaran
Imperial College London
London, United Kingdom
ka709@ic.ac.uk

Antoine Cully
Imperial College London
London, United Kingdom
a.cully@imperial.ac.uk

Julian Togelius
New York University
New York City, NY, United States
julian@togelius.com

## ABSTRACT

In January 2019, DeepMind revealed AlphaStar to the world—the first artificial intelligence (AI) system to beat a professional player at the game of StarCraft II—representing a milestone in the progress of AI. AlphaStar draws on many areas of AI research, including deep learning, reinforcement learning, game theory, and evolutionary computation (EC). In this paper we analyze AlphaStar primarily through the lens of EC, presenting a new look at the system and relating it to many concepts in the field. We highlight some of its most interesting aspects—the use of Lamarckian evolution, competitive co-evolution, and quality diversity. In doing so, we hope to provide a bridge between the wider EC community and one of the most significant AI systems developed in recent times.

## CCS CONCEPTS

• **Computing methodologies** → **Multi-agent reinforcement learning**; **Neural networks**; *Bio-inspired approaches*;

## KEYWORDS

Lamarckian evolution, co-evolution, quality diversity

**ACM Reference Format:**
Kai Arulkumaran, Antoine Cully, and Julian Togelius. 2019. AlphaStar: An Evolutionary Computation Perspective. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3319619.3321894

## 1 BACKGROUND

The field of artificial intelligence (AI) has long been involved in trying to create artificial systems that can rival humans in their intelligence, and as such, has looked to games as a way of challenging AI systems. Games are created by humans, for humans, and therefore have external validity to their use as AI benchmarks [22].

After the defeat of the reigning chess world champion by Deep Blue in 1997, the next major milestone in AI versus human games was in 2016, when a Go grandmaster was defeated by AlphaGo [16]. Both chess and Go were seen as some of the biggest challenges for AI, and arguably one of the few comparable tests remaining is to beat a grandmaster at StarCraft (SC), a real-time strategy game. Both the original game, and its sequel SC II, have several properties that make it considerably more challenging than even Go: real-time play, partial observability, no single dominant strategy, complex rules that make it hard to build a fast forward model, and a particularly large and varied action space.

DeepMind recently took a considerable step towards this grand challenge with AlphaStar, a neural-network-based AI system that was able to beat a professional SC II player in December 2018 [20]. This system, like its predecessor AlphaGo, was initially trained using imitation learning to mimic human play, and then improved through a combination of reinforcement learning (RL) and self-play. At this point the algorithms diverge, as AlphaStar utilises population-based training (PBT) [9] to explicitly keep a population of agents that train against each other [8]. This part of the training process was built upon multi-agent RL and game-theoretic perspectives [2, 10], but the very notion of a population is central to evolutionary computation (EC), and hence we can examine AlphaStar through this lens as well[1].

## 2 COMPONENTS

### 2.1 Lamarckian evolution

Currently, the most popular approach to training the parameters of neural networks is backpropagation (BP). However, there are many methods to tune their hyperparameters, including evolutionary algorithms (EAs). A particularly synergistic approach is to use a memetic algorithm (MA), in which evolution is run as an outer optimisation algorithm, and individual solutions can be optimised by other means, such as BP, in an inner loop [12]. In this specific case, an MA can combine the exploration and global search properties of EAs with the efficient local search properties of BP.

PBT [9], used in AlphaStar to train agents, is an MA that uses Lamarckian evolution (LE)[2]: in the inner loop, neural networks are continuously trained using BP, while in the outer loop, networks are picked using one of several selection methods (such as binary tournament selection), with the winner's parameters overwriting the loser's; the loser also receives a mutated copy of the winner's hyperparameters [6]. PBT was originally demonstrated on a range of supervised learning and RL tasks, tuning networks with higher performance than had previously been achieved. It is perhaps most beneficial in problems with highly non-stationary loss surfaces, such as deep RL, as it can change hyperparameters on the fly.

As a single network may take several gigabytes of memory, or need to train for several hours, scalability is key for PBT. As a consequence, PBT is both asynchronous and distributed [13]. Rather than running many experiments with static hyperparameters, the same amount of hardware can utilise PBT with little overhead—the

[1] Note that we present a high-level overview of general interest, and have left aside the many deep links to the crossovers between EC and game theory [17].
[2] A more extensive literature review on LE can be found in the original paper.


*GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic*

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3321894

outer loop reuses solution evaluation from the inner loop, and requires relatively little communication. When considering the effect of non-stationary hyperparameters and pre-emption on weaker solutions, the savings are even greater.

Another consequence of these requirements is that PBT is steady state [19], as opposed to generational EAs such as classic genetic algorithms. A natural fit for asynchronous EAs and LE, steady state EAs can allow the optimisation and evaluation of individual solutions to proceed uninterrupted and hence maximise resource efficiency. The fittest solutions survive longer, naturally providing a form of elitism/hall of fame, but even ancestors that aren't elites may be preserved, maintaining diversity[3].

## 2.2 Co-evolution

When optimising an agent to play a game, like in AlphaStar, it is possible to use self-play for the agent to improve itself. Competitive co-evolutionary algorithms (CCEAs) can be seen as a superset of self-play, as rather than keeping only a solution and its predecessors, it is instead possible to keep and evaluate against an entire population of solutions. Like self-play, CEAs form a natural curriculum [7], but also confer an additional robustness as solutions are evaluated against a varied set of other solutions [15, 18].

Through the use of PBT in a CCEA setting, Jaderberg et al. [8] were able to train agents to play a first-person game from pixels, utilising BP-based deep RL in combination with evolved reward functions [1]. The design of CEAs have many aspects [14], and characterising this approach could lead to many potential variants. Here, for example, the interaction method was atypically based on sampling agents with similar fitness evaluations (Elo ratings), but many other heuristics exist.

## 2.3 Quality diversity

A major advantage of keeping a population of solutions—as opposed to a single one—is that the population can represent a diverse set of solutions. This is not restricted strictly to multi-objective optimisation problems, but can also be applied to single objectives, where behaviour descriptors (BDs; i.e., solution phenotypes) can be used to pick solutions in the end. Quality diversity (QD) algorithms explicitly optimise for a single objective (quality), but also search for a large variety of solution types, via BDs, to encourage greater diversity in the population [4]. Recently, Ecoffet et al. [5] used a QD algorithm to reach another milestone in playing games with AI—their system was the first to solve Montezuma's Revenge, a platform game notorious for its difficulty in exploring the environment.

In SC, there is no best strategy. Hence, the final AlphaStar agent consists of the set of solutions from the Nash distribution of the population—the set of complementary, least exploitable strategies [2]. In order to improve training, as well as increase the variety in the final set of solutions, it therefore makes sense to explicitly encourage diversity. As it does so, AlphaStar can also be classified as a QD algorithm. In particular, agents may have game-specific BDs, such as building extra units of a certain type, but also criteria to beat a certain other agent[4], criteria to beat a set of other agents, or even a mix of these. Furthermore, these specific criteria are also adapted online, which is relatively novel among QD algorithms [21].

[3]When given an appropriate selection pressure [11].

[4]A concept highly related to competitive fitness sharing in CCEAs [15].

There is more that could be done here though: it may be possible to extract BDs from human data [22], or even learn them in an unsupervised manner [3]. And, given a set of diverse strategies, a natural next step is to infer which might work best against a given opponent, enabling online adaptation.

# 3 DISCUSSION

While AlphaStar is a complex system that draws upon many areas of AI research, we believe a hitherto undersold perspective is that of it as an EA. In particular, it combines LE, CCEAs, and QD to spectacular effect. We hope that this perspective will give both the EC and deep RL communities the ability to better appreciate and build upon this significant AI system.

# REFERENCES


[1] David Ackley and Michael Littman. 1991. Interactions between learning and evolution. *Artificial life II* 10 (1991), 487–509.

[2] David Balduzzi, Karl Tuyls, Julien Perolat, and Thore Graepel. 2018. Re-evaluating evaluation. In *NeurIPS*.

[3] Antoine Cully and Yiannis Demiris. 2018. Hierarchical Behavioral Repertoires with Unsupervised Descriptors. In *GECCO*.

[4] Antoine Cully and Yiannis Demiris. 2018. Quality and diversity optimization: A unifying modular framework. *TEVC* 22, 2 (2018), 245–259.

[5] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2019. Go-Explore: a New Approach for Hard-Exploration Problems. *arXiv preprint arXiv:1901.10995* (2019).

[6] David E Goldberg and Kalyanmoy Deb. 1991. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*. Vol. 1. 69–93.

[7] W Daniel Hillis. 1990. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D: Nonlinear Phenomena* 42, 1-3 (1990), 228–234.

[8] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, et al. 2018. Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. *arXiv preprint arXiv:1807.01281* (2018).

[9] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, et al. 2017. Population based training of neural networks. *arXiv preprint arXiv:1711.09846* (2017).

[10] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, et al. 2017. A unified game-theoretic approach to multiagent reinforcement learning. In *NeurIPS*. 4190–4203.

[11] Brad L Miller and David E Goldberg. 1995. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems* 9, 3 (1995), 193–212.

[12] P Moscato. 1989. *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*. Technical Report. California Institute of Technology.

[13] Mariusz Nowostawski and Riccardo Poli. 1999. Parallel genetic algorithm taxonomy. In *KES*. 88–92.

[14] Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. 2012. Coevolutionary principles. In *Handbook of Natural Computing*. 987–1033.

[15] Christopher D Rosin and Richard K Belew. 1997. New methods for competitive coevolution. *Evolutionary Computation* 5, 1 (1997), 1–29.

[16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484.

[17] John Maynard Smith. 1982. *Evolution and the Theory of Games*. Cambridge University Press.

[18] Kenneth O Stanley and Risto Miikkulainen. 2004. Competitive coevolution through evolutionary complexification. *JAIR* 21 (2004), 63–100.

[19] Gilbert Syswerda. 1991. A study of reproduction in generational and steady-state genetic algorithms. In *Foundations of Genetic Algorithms*. Vol. 1. 94–101.

[20] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, et al. 2019. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/. (2019).

[21] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. 2019. Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions. *arXiv preprint arXiv:1901.01753* (2019).

[22] Georgios N Yannakakis and Julian Togelius. 2018. *Artificial Intelligence and Games*. Springer.