# 7 Neuroevolution of Collective Systems

One of the most fascinating aspects of nature is that groups with millions or even trillions of elements can self-assemble into complex forms based only on local interactions and display, what is called, a collective type of intelligence. For example, ants can join to create bridges or rafts to navigate difficult terrain, termites can build nests several meters high without an externally imposed plan, and thousands of bees work together as an integrated whole to make accurate decisions on when to search for food or a new nest. Surprisingly, achieving these incredible abilities is a result of following relatively simple behavioral rules. These rules have been discovered through evolution that relies on cooperating individuals, i.e. through cooperative coevolution.

A fundamental driving force in evolution is competition. Individuals compete for resources, mates, and status. Groups of individuals battle for resources, but also may engage in direct conflict, including predators trying to catch prey who in turn try to avoid being caught. When the opponents discover new successful behaviors, the species also have to develop new mechanisms to survive. This process results in continual adaptation, i.e. competitive coevolution.

Cooperative and competitive coevolution can be used to drive neuroevolution as well. Mechanisms range from cooperating neurons and networks, and cellular automata defined by evolved neural networks, to establishing an arms race of increasingly competing networks. In many case, complex behavior results that would be difficult to discover in other ways.

## 7.1 Cooperative Coevolution

A fundamental insight in generating intelligent systems is that they do not exist in a vacuum: Intelligence often emerges from interactions with the environment. These interactions may originate from constraints of a physical body, with its limited sensory and motor abilities. They may originate from constraints posed by the physical surroundings: for instance Herb Simon's point that even though an ants path may appear complex to the outsider, the ant may be largely responding to the obstacles and contours in its path (Simon 1969). Most importantly, significant interactions originate from other agents. They may be adversarial, posing a threat or obstacle, or they may be cooperative, requiring collaboration to achieve a common goal.

Neuroevolution is well suited for building such interactive intelligent systems. The techniques focus on constructing intelligent systems from a large number of components that

work together. A fundamental principle is cooperative coevolution, i.e. evolving these components together to achieve effective behavior (Wiegand 2003). Such cooperation can take place at many levels: a single neural network; multiple neural networks in a multiagent system; in a competitive environment between multiple cooperative multiagent systems. The techniques are based on the same fundamental principle of shared fitness; but each are addressing the challenge of intelligent behavior at a different level.

### 7.1.1 Evolving a single neural network

At the most basic level the goal is to construct a single intelligent agent in an environment that returns a dedicated fitness for it. In other words, a neural network is formed by evolving a population of partial solutions, such as neurons, connections, or modules.

In the spirit of classifier systems (Holland and Reitman 1978), the first approaches of this kind focused on the evolution of cooperative neurons (Moriarty and Miikkulainen 1997; Potter and Jong 2000; Husbands and Mill 1991). For example in the SANE system there was a single population of neurons, each with its own input connections. The networks were formed according to blueprints, i.e. a separate population of individuals that specified which neurons from the population were included to form the network. The networks specified by each blueprint were evaluated in the task, and the neurons in the blueprint inherited the blueprint's fitness. Both the blueprint and the neuron population were evolved based on this fitness, thus encouraging discovery of partial solutions (i.e. neurons) that collaborate well with other neurons.

This principle was further enhanced in the ESP system (enforced subpopulations, Section 5.6) where, instead of a diverse set of blueprints, there was only one network structure: a fully connected network of $n$ neurons (Figure 7.1; Gomez and Miikkulainen 1997). However, each neuron in the network was evolved in a separate subpopulation—thus, each subevolution searched for a neuron that optimized performance for one location in the network. The networks were then formed by selecting one neuron from each subpopulation randomly to fill the corresponding location in the network. All the neurons started with random weights, and all the subpopulations were thus initially identical. However, over evolution, they gradually diverged and specialized: they discovered differentiated, computational roles for the neurons.

For instance in the task of evolving a network that can run through a maze as a simulated Khepera robot, several such roles could be identified. One subpopulation evolved neurons that would slow the robot down if there was an obstacle in front; another veered the robot to the right if there was an obstacle on the left; another veered left with an obstacle on the right. Although such discovery and specialization was evident, most importantly, each subpopulation usually performed at least two such subfunctions to some extent. The reason is that such redundancy makes the construction of competent individuals more robust, the neurons do not have to be perfect in what they do because other neurons in the network compensate for their flaws. Such construction also results in a more robust search: even if a suboptimal neuron is sometimes chosen from one of the subpopulations, the others cover for it. Thus, selection favors redundancy and thus more robust networks. This is a powerful fundamental principle of cooperative coevolution in general.

So far, the partial solutions (i.e. neurons) inherit the fitness of the full solution (i.e. a network) as is. However, such neuroevolution can be further enhanced by calculating fitness

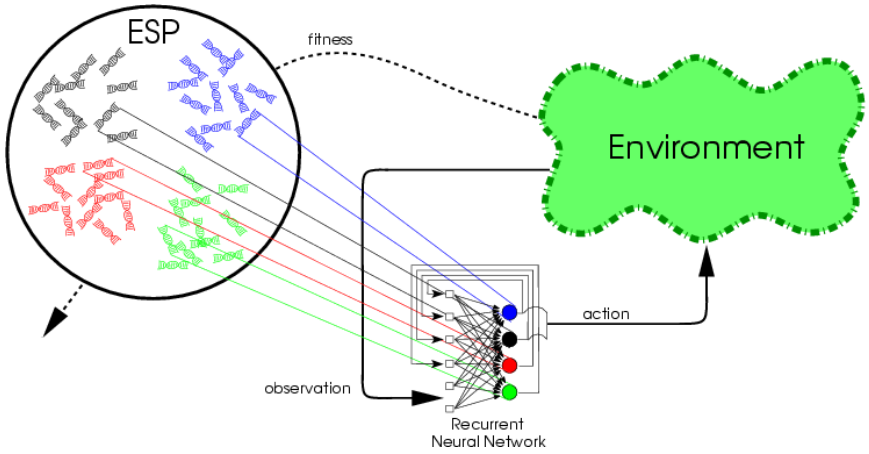



Figure 7.1: **Evolution of Subpopulations of Neurons**. In the cooperative coevolution of a single network, each subpopulation evolves one neuron for the network, which may be e.g. fully recurrent. The genetic encoding of each neuron specifies the neuron's connection weights to other neurons. Each neuron receives the fitness of the entire network evaluated in the task. Thus, neurons evolve to cooperate well with other neurons: the subpopulations optimize compatible subtasks and each subtask is encoded robustly in a couple of subpopulations. Such search for partial solutions is also efficient: the subtasks remain diverse, the approach avoids competing conventions, and the search space is compact. (Figure from Gomez 2003)

of individual neurons separately as well, and using it in combination with the inherited network fitness. This is possible through difference evaluation, i.e. evaluating the network in the task with and without the neuron, thus measuring how much better off (or worse off) the network is with each neuron. In control tasks such as double pole balancing and rover exploration, this approach can find significantly better solutions and find them significantly faster (Agogino, Tumer, and Miikkulainen 2005).

Based on these pioneering systems, it is already possible to see why the cooperative coevolution approach can be powerful. There are three main reasons: First, it has a build-in mechanism for maintaining diversity and avoiding premature convergence. A good network requires many different kinds of neurons. If e.g. the neural population in SANE starts to converge, the similar neurons perform poorly in a network, and are discarded in favor of those that are different. Second, it avoids the competing conventions problem. The neurons are chosen to distinct locations in the network, and optimized for performance for those specific locations. Third, it reduces the search space. Instead of having to optimize all the connection weights in the network at once, it is sufficient to optimize the weights of single neurons—which can be done easily in parallel multiple times. There are other ways to solve these problems in neuroevolution, including indirect encodings (Section 4.2.2), but the cooperative coevolution method is designed to tackle them explicitly.

This approach of cooperative coeveolution of compatible roles can be extended to other levels of granularity as well. A particularly powerful way of constructing recurrent neural networks is CoSYNE (Gomez, Schmidhuber, and Miikkulainen 2008), where individual

connections are evolved in separate subpopulations. However, although the general idea is a logical and compelling extension of ESP, it turned out that with such a large number of subtasks, it is difficult for evolution to converge to a compatible set. The solution is to focus the search in two ways. First, individual connections are not chosen randomly from each subpopulation to form a network, but instead the connections with the same index (i.e. location) in the subpopulation are combined into the network. Thus, the indices serve as simple blueprints, allowing search to focus on refining these networks. Second, in addition to the usual mutation and crossover in each subpopulation, a small subset of individuals are permuted within each subpopulation, thus exploring a different role for each of them. In this manner, the search can more effectively find good combinations of individual weights, which is especially important in highly recurrent neural networks. At the time, CoSYNE was able to discover solutions to the most challenging control tasks, such as balancing two poles simultaneously on a moving cart without precomputed velocity information, where other neuroevolution and reinforcement learning methods could not.

Interestingly, the cooperative coevolution approach has recently proven valuable at the higher level of granularity as well, i.e. neural architecture search for deep learning. As will be described in more detail in Chapter 10, the goal in neural architecture search is to find a design for a deep learning system that performs as well as possible when trained with gradient descent. This process requires finding optimal hyperparameter settings, network topologies, and layer types. It turns out that these elements can be coevolved in separate subpopulations to form entire architetures similarly to how neurons are evolved to form networks. For instance in the CoDeepNEAT method (Miikkulainen, Liang, et al. 2023), network modules consisting of a few layers and connections between them are coevolved in separate subpopulations, and a blueprint population is evolved to indicate how these modules are combined to form complete networks. Each of these subpopulations are evolved with NEAT to form complex recurrent structures. In essence, CoDeepNEAT is thus a combination of SANE, ESP, and NEAT, applied at the level of large deep learning architectures.

Compared to other neural architecture search methods, CoDeepNEAT is particularly powerful in exploring new architectures, because its search space is relatively unconstrainted. It is also possible to seed it with human designs and find novel combinations of them that the humans may have missed. For instance in the domain of image captioning, CoDeepNEAT was initialized with the types of layers and connections that existed in the state-of-the-art architecture at the time, the Show&Tell network (Vinyals et al. 2015). It was able to find a network that improved performance 5%. Interestingly, it did so by employing a principle that is not common in human designs: The best networks included multiple parallel pathways of processing that were brought together in the end. This principle will still need to be evaluated more generally, but it illustrates the kind of discoveries that are possible using the cooperatie evolutionary approach.

### 7.1.2 Evolving a team

At the next higher level of coordination, neuroevolution can be used to construct teams, i.e. groups of individual agents that solve problems cooperatively.

The most straightforward extension from the single agent construction is to evolve each agent in a separate subpopulation, and reward each agent based on the success of the entire

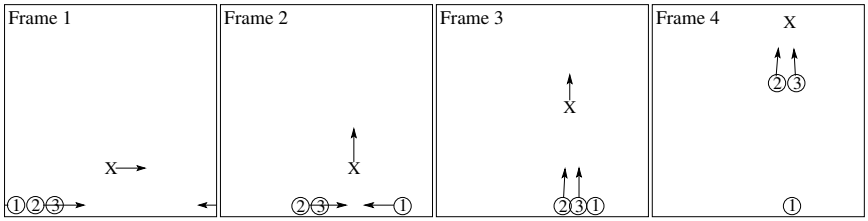



Figure 7.2: **Role-based Cooperation through Stigmergy**. Similarly to a single-network evolution, team members can be evolved in separate subpopulations and rewarded based on team success. In a toroidal world, three predator agents tried to capture a prey (X) that always runs away from the nearest predator and is as fast as the predators. Two of the predators (2, 3) evolved chaser roles, and the third (1) a blocker role: The chasers push the prey to the waiting blocker around the torus. Remarkably, evolution of agents in separate subpopulations was more effective than evolution of a central controller for the entire team. It was also more efficient to not bother with communication with other team members (even through visual sensing); each team member knew their role, and it was most effective for them to simply observe the prey, i.e. to communicate through stigmergy. For an animation of this behavior, see https://neuroevolutionbook.com/neuroevolution-demos. (Figure from Yong and Miikkulainen 2010)

team. Predator-prey, or pursuit-evasion, scenario is a good way to illustrate the approach. In the simplest such scenario, a team of three predators was evolved to capture a single non-evolving (algorithmic) prey that always moves away from the nearest predator (Yong and Miikkulainen 2010). However, the prey is as fast as the predators. Thus, in an unbounded (e.g. toroidal) field it could never be caught, unless the predators evolve a cooperation strategy.

Such a strategy was indeed evolved reliably using the ESP approach outlined above (Figure 7.2). Each predator agent was controlled by an ESP neural network, i.e. a recurrent network evolved from their own subpopulation of neurons. At a hierarchically higher level, the three agents were evolved in parallel, and evaluated based on how often the entire team was able to capture the prey. Indeed, two behavioral roles emerged: two of the agents behaved as chasers, forcing the prey to run straight away from them in a path that extended around the toroidal space. The remaining agent behaved as a blocker, staying in place waiting for the chasers to push the prey to it—the prey had nowhere to go and was captured.

Upon further analysis, two remarkable observations were made. First, such a cooperative approach was more effective than evolving a single network to control all three agents. Second, it was more effective to evolve it without any direct communication between the agents, even as simple as simply sensing each other's location. Each agent would only sense the prey's location, and based on the role they had evolved into, knew what the other agents were likely doing, and what they needed to do themselves. In other words, their coordination was based on stigmergy, i.e. communication through the environment.

Both of these are powerful principles that can be harnessed more generally in building complex systems. They suggest that in similar domains, discovering compatible behaviors can be easier than discovering comprehensive strategy for the entire team. Each behavior,

or role, can be flexible and robust on its own, compensating for inaccuracies in the other agents' behavior—such robustness is difficult to discover in a central control system. Also, when cooperation is based on such roles, it may be enough to observe simply the current state of the problem: The subsequent behavior of each role can be assumed without direct observation or communication, making the problem solving more effective. The situation is similar to playing soccer with a team that has practiced together and knows each other well: You know what the others are doing even without looking, and you know what you need to do by observing the opponents. A possible generalization of this idea is the evolution of ensembles: Each ensemble member discovers a role that solves only part of the problem, but when combined with the other roles in the ensemble, constitutes a full solution.

While role-based cooperation is often effective, sometimes the behavior has to be more flexible. In the soccer analogy, you may be playing a pick-up game: You do not know the other players in your team, and have to be constantly observing them to decide what you should do. More generally, the number of agents required in different roles may vary over time, and the agents may need to be able to switch roles For instance, in robotic soccer the behaviors are different depending on which team has the ball and where in the field. A team of agents sent to rescue people in a disaster may require cleaning up rubble, stabilizing structures, searching for targets, transporting them out, and each agent should be able to take on any of these roles as needed.

An entirely different kind of evolutionary approach may be needed to construct such teams. Instead of evolving specialists, it is necessary to evolve generalists. This goal can be achieved e.g. by evolving a homogeneous team, i.e. each member of the population is evaluated based on how well it performs as part of a team that consists of clones of itself (Bryant and Miikkulainen 2018). For the team to be successful, it needs its members to perform different roles at different times. Thus, evolution favors individuals that can adapt their behavior to the situation, assuming appropriate behaviors that are compatible with those of the other team members.

Such behavior can be demonstrated naturally in a civilization-type game environment. The agents are settlers who have to perform various tasks at various times, including division of labor into construction, mining, agriculture, defense, etc. One such demonstration focused on legions defending multiple cities against barbarians. The barbarians were controlled algorithmically, attacking cities with little defense, retreating when outnumbered, and spawning at a regular rate in the countryside to replace those eliminated by the legions. The legions were rewarded based on minimal damage to the cities, i.e. time they were occupied by the barbarians.

Unlike in the role-based cooperation approach outlined above, in the adaptive teams approach it is useful for the agents to observe each other continuously (i.e. to communicate), in addition to the barbarians and the state of the cities. It is through such global awareness that the agents evolve to decide what role they should take on. It requires developing an internal model of the other agents and their behavior—a rudimentary theory of mind if you will. Some of the legions take on the task of defending the cities under attack, while others prepare to defend cities that are likely to be attacked soon, and yet others proactively hunt down the barbarians in the countryside. While perfect fitness is not possible due to randomness and occasionally algorithmic changes to the barbarian's strategy, the adaptive approach does help them obtain better fitness. In a sense, the adaptation helps them deal

with the uncertainty and unstability in the domain. Such robustness can serve as an important ingredient in building intelligent agents that can cope with the messiness of the real world.

Interestingly, for such coordination and communication to evolve, selection must operate at the team level rather than at the individual level Floreano et al. 2007. How such high-level selection can be established is an interesting question that has implications to biology as well, e.g. in understanding evolutionary breakthroughs (Section 14.6) and major transitions (Section 9.1.5)

## 7.2 Competitive Coevolution

While cooperation of multiple elements or agents is a powerful approach in building complex behavior, so is competition. That is, the agents evolve to outdo each other, and the population thus collectively discovers increasingly more powerful behaviors in an evolutionary arms race. Competitive coevolution is useful because it defines an open-ended fitness function automatically, but it is also difficult to guarantee that progress is made continuously in an absolute sense. The process can be set up to discover a single effective behavior, or it can be set up to evolve multiple competing behaviors. These approaches are described in the subsections below.

### 7.2.1 Evolving single neural networks

One challenge in constructing complex behavior through neuroevolution is that it is difficult to design a suitable objective function. One approach is to make it very general and high-level, such as survival, or number of games won, or number of offspring generated. This approach poses little constraints on how such fitness is achieved, and evolution can find creative solutions, but the signal may be too weak to make much progress. Another approach is to specify a number of detailed components that are believed to be part of successful behavior, such as high speed, sharp turns, or accurate shooting, each providing part of the fitness. It is possible to make incremental progress in this manner, but it is difficult to make sure that robust solutions emerge, let alone creative solutions.

Competitive coevolution solves these problems by defining fitness in terms of the behaviors in the current population. Individuals compete with other individuals, and their fitness is determined based on how well they do in this competition. As the population improves, it becomes more difficult to achieve high fitness, thereby establishing an open-ended, automatic mechanism of shaping the fitness function.

Competitive coevolution is thus similar to curricular, or incremental, learning in general machine learning. Generative adversarial networks (GANs; Goodfellow et al. 2014) are based on a similar mechanism, as are game-playing systems based on self-play such as AlphaZero (Silver et al. 2018). One of the earliest such systems was based on neuroevolution: Blondie24 evolved a neural network activation function for checkers (and later chess). Starting without any built-in expert knowledge, it evolved into an expert-level player (Fogel 2001; Fogel et al. 2004; Chellapilla and Fogel 1999). There is a large literature on competitive coevolution since the 1950s, including analyses based on game theory (Adami, Schossau, and Hintze 2016; Ficici and Pollack 2001; de Jong and Pollack 2004; Samuel 1963). There are many examples in this book as well, including those in Chapter 9.

The main challenge in competitive coevolution is to make sure that it actually makes progress toward better solutions. Since fitness is defined in relation to other solutions, improvement is not guaranteed in any absolute sense. It is possible to achieve higher fitness simply by exploiting weaknesses in the current candidates. Therefore, it is often useful to maintain a collection of previous candidates and evaluate fitness against them as well as the current population. In this manner, good candidates are indeed better than anything discovered by evolution so far.

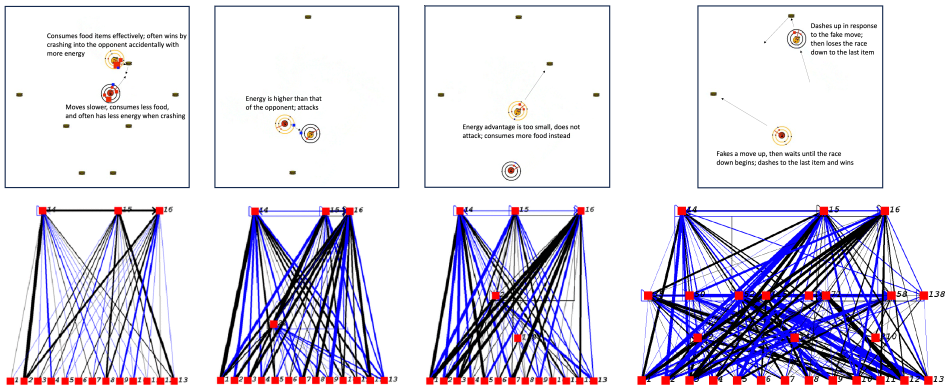
However, a mechanism such as NEAT supports absolute progress in its very nature. As reviewed in Section 3.4, NEAT starts with a minimal network and gradually complexifies it over evolution. Through mutation and crossover, it adds more nodes and connections to the existing networks. The earlier structures are still there—evolution elaborates on them instead of replacing them. Therefore, the earlier behaviors are likely to be there as well, and the newer behaviors are likely to be more elaborate and effective. Therefore, it is likely that the newer solutions perform better in comparison to the earlier ones, thereby guiding evolution towards absolute progress.

This process was demonstrated in an experiment where neural network controllers were evolved for a combined foraging, pursuit, and evasion task (Stanley and Miikkulainen 2004). Two simulated Khepera-like robots were placed in a closed environment with scattered food items. They were able to sense the distance to the opponent and the food items around them, distance to the nearest wall, and difference between their opponent's and their own energy. The robots moved around by powering their two wheels; they gained strength by consuming the food items and lost strength by moving. They would win the game by crashing into their opponent when they had higher strength than the opponent. Thus, performing well required not only sensing and moving but also estimating how much energy they and their opponent would gain and lose by consuming and moving. Fitness was defined as the average win rate over the four highest species champions.

Because NEAT starts small and complexifies (as was discussed in Section 3.4), it was possible to understand the complexification that took place in the networks and behaviors throughout the coevolutionary process. Evolution first discovered a simple foraging behavior that was often successful by chance: The agent occasionally crashed into the opponent when it had more energy than the opponent (Figure 7.3*a*). It then evolved a hidden node that allowed it to make an informed switch between behaviors: Attack when it had high energy, and rest when it did not (Figure 7.3*b*). Another added node made it possible to predict the agent's own and its opponent's energy usage from afar and attack only when a win was likely (Figure 7.3*c*). The most complex strategy, with several more nodes and complex recurrent connections between them, allowed the agent to predict also the opponent's behavior, encourage it to make mistakes, and take advantage of the mistakes to win (Figure 7.3*d*).

Note that such an analysis and explainability is possible precisely because the networks are evolved in a principled manner through elaboration. Even though large deep-learning networks could perhaps be trained in this task, they would remain opaque and not provide much insight into how the network establishes its behavior. Consequently, they could not be trusted in the same way as NEAT networks can.

Interestingly, the elaboration process turned out crucial in discovering such complex behavior. In a further experiment, a population was initialized with the final architecture




(*a*) Forage (*b*) Forage/attack (*c*) Predict energy (*d*) Cause a mistake

Figure 7.3: **Discovering Complex Behavior through Competitive Coevolution.** Two simulated Khepera robots need to consume food, pursue the opponent when they have higher energy than the opponent, and evade it when their energy is lower. When the robots collide, the one with higher energy wins. In the top row, the dark ovals are food items, and the red and yellow circles are the two robots. The red line indicates the direction the robot is facing, the outer ring indicates the opponent sensor values, and the inner ring the food sensor values. The rings are yellow for the robot with higher energy. In the bottom row, the network nodes are depicted as red squares and numbered in the order they were created. Positive connections are black and negative are blue, recurrent connections are indicated by triangles, and the width of the connection is proportional to its strength. The approach discovered (*a*) a foraging strategy that resulted in high energy and was often successful when accidentally crashing on the opponent, (*b*) a hidden node that allowed it to switch between following and resting based on energy, (*c*) a way to model and compare opponent's and their own energy, and (*d*) eventually how to fake a move towards a far-away food item (top), causing the opponent to (i) dash to it and then (ii) spend most of its energy to get to the last item (left) but (iii) failing to get to it first, thereby (iv) providing an easy win. Complexifying evolution thus provides a way of understanding network performance; in this experiment, it provides a clear example of how a single competitive coevolution population can discover increasingly complex behaviors. For animations of these behaviors, see https://neuroevolutionbook.com/neuroevolution-demos. (Bottom figures from Kenneth O. Stanley 2003)

from Figure 7.3*d*, i.e. all individuals had the same architecture with randomized weights. This architecture supports the complex behavior, and therefore it should be easy for evolution to discover the right weights. Surprisingly, it was not; each complexification step builds on a prior, simpler architecture that already performs some desired behaviors. It is therefore relatively easy to add a complexification to improve upon that behavior. In multiple such small steps, a complex behavior eventually develops. In contrast, discovering everything at once is very difficult, and such evolution does not get past the first few simple behaviors.

Thus, the foraging/pursuit/evasion experiment demonstrates how coevolution can be harnessed to discover complex behavior. It is achieved collectively in a simple population where every individual tries to solve the same problem, and they simply compete against

each other. The coevolutionary setup can be made more complex by incorporating multiple populations that try to outdo each other explicitly. In a sense, one population discovers solutions and the other discovers more challenging problems. One example is given in the next section; another (POET) later in Chapter 9.

### 7.2.2 Evolving multiple teams

At the next higher level of complexity, multiple cooperative teams coevolve in a competitive environment. Each team challenges the other teams to perform better, thus establishing an evolutionary arms race: Over time, each team outsmarts the other multiple times, leading to increasingly complex behavior for all teams.

Competitive coevolutionary dynamics have been studied comprehensively theoretically e.g. using game theory, and it is relatively well understood (Popovici et al. 2012; Mitchell 2006). Absolute improvement is sometimes difficult to establish, and the process can go wrong in multiple ways: For instance, instead of getting better, the teams may simply become more weird. Later teams may even lose to the earlier ones. However, in many natural tasks the more complex behavior often subsumes the earlier behaviors, which does lead to improvement in an absolute sense.

Once again a good domain to study such competitive-cooperative dynamics is predator-prey tasks (Rawal, Rajagopalan, and Miikkulainen 2010). Extending the approach of the previous section, a simulation can be set up two evolve both the prey and the predator populations—let's call them zebras and hyenas. Again in a toroidal world, the zebras can run away from the hyenas, but hyenas can catch them by approaching from multiple sides.

At the very first stages of evolution (generations 50-75), the zebras evolve an individual strategy of running away from the nearest predator, replicating the algorithmic behavior in the previous section. Correspondingly, the predator team evolves a two-blocker, one-chaser strategy (Figure 7.4 Phase 1). In the next phase (generations 75-100; Phase 2), the prey evolves a new strategy of running in a small circle with the chaser following at its tail. This strategy is effective because the blockers simply wait and to the prey. Next (generations 100-150; Phase 3), one of the blocker predators evolves to act as a chaser as well, approaching the prey from different directions. As a response (generations 150-180; Phase 4), the prey evolves a baiting strategy, letting both chasers get close and then escaping away from them both. Next (generations 180-250; Phases 5–6), the predators evolve to change roles between blockers and chasers dynamically, so that they can better sandwich the prey. As a result (generations 250-300, Phase 7), the prey adjusts its strategy, letting all predators get close, and then escaping between them. In the next few hundred generations (300-450, Phases 8–9), both of these strategies become gradually more refined and precise, eventually resulting in about 50-50 chance of the prey escaping and getting caught—similar to what is seen in biology.

However, an interesting next step is to add another prey to the prey team—the prey can now evolve cooperation in order to confuse the predators. This is one of the most effective strategies used by prey in nature, and it also evolves reliably in the simulations. First (in 150 further generations), the predators mostly capture one prey at a time, but are often confused by the other, and fail. Then (generations 150-200, Phase 1), they are able to adapt their single-prey sandwiching strategy to herd the two prey together and capturing both of them. Remarkably, the prey are able adapt their strategy in the same way (generations 200-300,

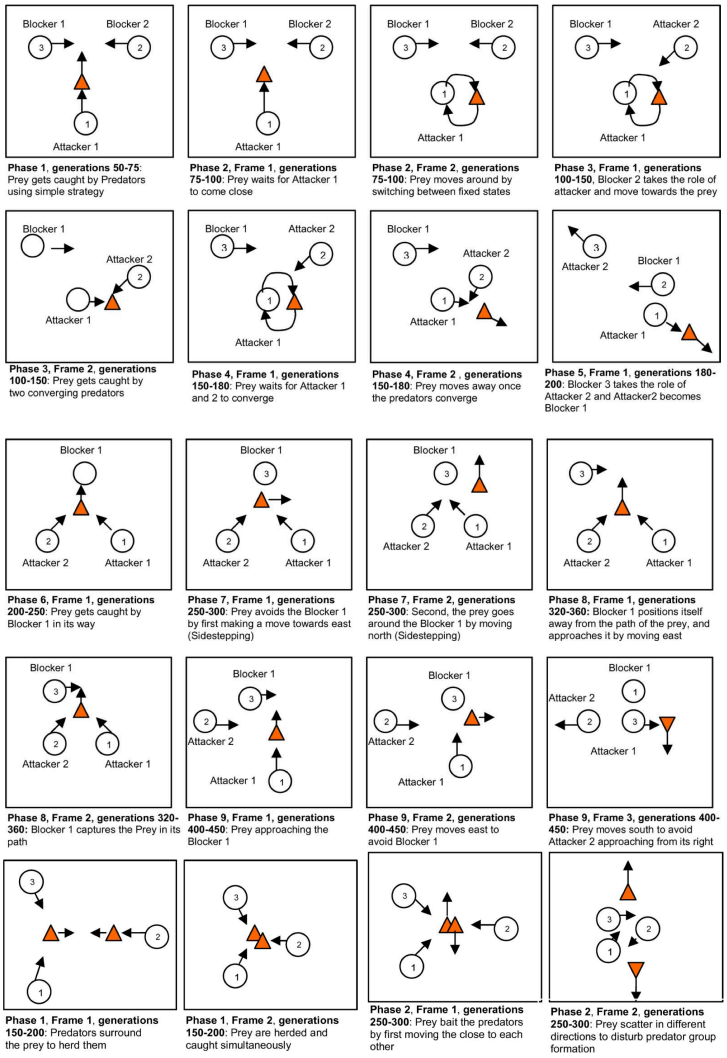



Figure 7.4: **Evolutionary Arms Race of Increasingly Complex Pursuit and Evasion Strategies**. Through multiple phases, the predator and prey populations alternate in gaining the upper hand in the competition, which serves as a challenge and opportunity for evolution to improve the disadvantaged population. The later behaviors largely subsume the earlier ones, and therefore there is a progression in an absolute sense toward more complex and effective behaviors that would otherwise be difficult to discover. The simulation also serves to shed light on observed animal behaviors such as cooperative hunting and herding, and escaping by confusing the predators. It thus demonstrates both a way to construct complex intelligent agents, as well as to understand how intelligence may have emerged in biological evolution. For animations of these behaviors, see https://neuroevolutionbook.com/neuroevolution-demos. (Figures from Rawal, Rajagopalan, and Miikkulainen 2010)

Phase 12, baiting the predators together, and then escaping in opposite directions, leaving the predators confused. In further evolution again, both of these strategies become more precise, resulting in about even chance of escape and capture in the end.

This example is interesting for two reasons: First, it illustrates how neuroevolution can be used to understand how the behaviors observed in nature may have emerged through coevolution. Sometimes, when observing biological behavior as is, it is difficult to understand aspects of it. However, behavior, like other aspects of biology, are product of evolution, and should be understood in the light of how evolution may have constructed it, through all the intermediate stages that may no longer be visible. Evolutionary computation simulations may be used to uncover them; for instance, why it may be beneficial for the prey to let the predators get close before escaping. These opportunities will be discussed in more detail in Chapter 14.

Second, the example demonstrates a successful coevolutionary arms race. Complex behavior is discovered through multiple stages, each a stepping stone to the next. The imbalance of performance at each state forms a challenge to the disadvantaged population, and evolution discovers ways to meet that challenge. In this manner, such competitive-cooperative coevolution may be a crucial ingredient in open-ended evolution, and perhaps also in establishing major transitions (Miikkulainen and Forrest 2021). Opportunities for such advances are discussed more in Section 9.1.

## 7.3 Cellular Automata

Many collective systems in nature are made out of many components that are highly interconnected. The absence of any centralized control allows them to quickly adjust to new stimuli and changing environmental conditions. Additionally, because these collective intelligence systems are made of many simpler individuals, they have in-built redundancy with a high degree of resilience and robustness. Individuals in this collective system can fail, without the overall system breaking down.

A simplified yet powerful platform to study collective systems in various contexts are cellular automata (CAs). They offer insights into how individual behaviors, when aggregated, can lead to the emergence of remarkable and often unexpected group-level phenomena. Constructing intelligent or life-like systems from a large number of cooperating components is central to CAs, and as we will see in this Chapter, they allow complex patterns to emerge based only on the local and self-organized interaction of cells. CAs have recently seen a renaissance and renewed interest in the machine learning community by scaling them up and combining them with deep neural networks.

Originally proposed in the 1940s, Cellular Automata mimic developmental processes in multi-cell organisms including morphogenesis. A CA is a spatially extended decentralized system that contains a grid of similarly structured cells, which are locally connected and updated periodically in discrete time steps. At every time step, the status of each cell can be represented as a state, which is then transitioned into the next state per the update rule. The specific transition depends on the current state of the cell and the neighboring cells (often this neighborhood is defined as the cells directly bordering to the cell in question, but larger neighborhood are also possible). For example, in a particular CA devised by John Conway in 1970 called Conway's Game of Life, a few rules govern the transition at each timestep, such as: an alive cell that has fewer than two alive neighbors dies while a cell becomes alive if it has exactly three neighbors. These automata serve as effective models for a range of physical and biological processes. For instance, they have been employed to simulate fluid

dynamics, the emergence of galaxies, seismic events like earthquakes, and the formation of intricate biological patterns.

A CA's transition rule can be specified as a lookup table that determines for each local neighborhood configuration, what should the state of the central cell be in the next timestep. While the states are either 0 or 1 in the e.g. Conway's Game of Life, we'll shortly see that cells can have more states or even be described by not only a single number but a hidden state vector instead. In Conway's Game of Life, the specific transition rules were human-defined. However, in some instances it can make sense to search for specific rules that lead to desired behaviors or patterns. For example, researchers such as Melanie Mitchell have shown that it is possible to optimize CA transition rules with evolutionary algorithms (Mitchell, Crutchfield, Das, et al. 1996). This way rules can be found that perform a specific type of computation, such as determining if the initial CA configuration has more 1s than 0s.

Instead of evolving rule tables directly (which can quickly become prohibitively large when the number of CA states increases), rules can also take the form of programs (John R Koza 1994) or neural network (Wulff and Hertz 1992). Here, a copy of the same program/neural network runs in each cell, taking information of a CA neighbors and potentially previous cell states into account to determine which state the cell should take next. Because each cell shares the same trainable parameters, the whole system can be viewed as a type of indirect encoding, in which the size of the grown patterns can potentially be much larger than the size of the underlying program/neural network.

A popular benchmark to test the abilities of these systems is to grow forms resembling simple 2D patterns. Originally proposed by developmental biologist Lewis Wolptert in the 1960s, the French flag problem (Wolpert, Tickle, and Arias 2015) is such a task, and asks how embryonic cells could differentiate into complex patter, such as the three differently colored stripes of a French flag. The inquiry extends to understanding how these patterns can scale proportionally with tissue size, for example, such that the grown French flag pattern is always one-third blue, one-third white, and one-third red. In an impressive early demonstration of collective intelligent systems, Julian Francis Miller 2004 showed that a genetic cell program can be evolved that allows growing a French flag-like pattern from a single cell, which can even self-repair when being damage. When the cell's update function is a neural network, it is now often called a Neural Cellular Automata (NCA), and we'll have a closer look at those next.

### 7.3.1 Evolving Neural Cellular Automata

In a Neural Cellular Automata (NCAs) (Wulff and Hertz 1992), a neural network updates the states of each cell based on communicating with their local neighbors. The same neural network is applied to each grid cell, resembling the iterative application of a convolutional filter (Gilpin 2019). In other words, NCAs can be viewed as a indirect encodings in which identical modules are applied with identical weight parameters accross the space of cells. More recently, the use of neural networks for growing CAs has seen a resurgence, in particular because of their integration with popular deep learning frameworks and tools for automatic differentiation.

Because NCAs are neural networks, they can naturally be evolved with the NEAT algorithm. However, here NEAT-evolved neural networks are applied slightly different to what

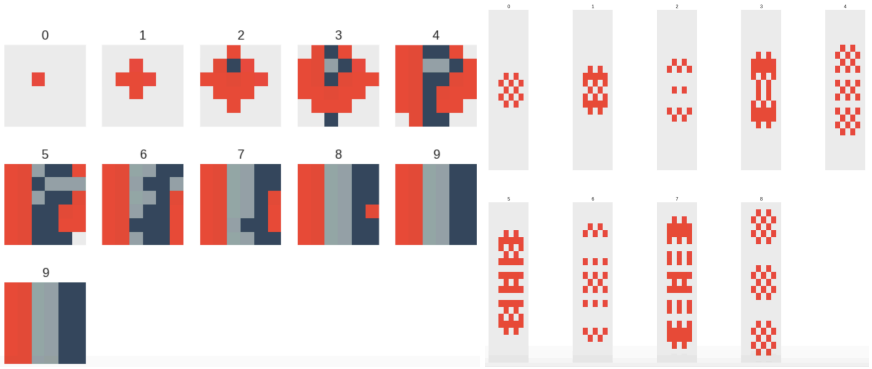



Figure 7.5: **CA-NEAT**. Examples of NEAT evolved neural networks that have learned to grow a French flag-like pattern (left) and to perform pattern replication (right), only through the local interaction of cells. (Figures from Nichele et al. 2017)

we have seen in previous sections. In an NCA, a collection of cells, each controlled by a copy of the same evolving neural network, need to learn to collaborate to perform the task at hand. For example, Nichele et al. 2017 studied how cell in such a system can learn to grow a certain target pattern together, based only on the local information they receive from the neighboring grid cells. Fitness was assigned based on how closely the resulting pattern resembles the target pattern during the growth process. In addition to growing a particular target pattern, the system can also be trained to replicate a certain pattern, which is another fundamental property of biological systems. Here, the neural network is task to replicate a given seed pattern a specific number of times.

NEAT is able to solve both of these tasks. Figure 7.5,left shows an example where a NEAT-evolved network grows a French flag-like pattern iteratively starting from an initial seed cell (Nichele et al. 2017). Figure 7.5,right demonstrates how an evolved neural networks learned to replicate an initial mosaic pattern along one axis, taking a total of eight developmental steps (Figure 7.5,right). To appreciate the complexity of this task, we invite the reader to design the rules to solves these tasks by hand.

How far can we push this approach? Can we learn to grow patterns of arbitrary complexity? While NEAT was able to discover networks that can grow simple shapes and learn to replicate them, further experiments showed that it can struggled to grow more complex shapes such as a Norwegian flag type pattern. The reason for this is likely that the evolutionary optimization algorithm gets stuck in some local optima of the fitness landscape. We have seen similar phenomena in Section 5.3, when trying to re-evolve CPPNs to generate specific target patterns like the skull image. In a similar vain, evolution here likely depends on discovering the proper stepping stones towards the solution and the developmental dynamics of NCAs likely makes this optimization problem even more complicated.

While more open-ended search methods such as Quality Diversity, could potentially help in evolving a network that produces a certain target image, recent work demonstrated that, if a target pattern is given, NCAs can also be trained efficiently by supervised gradient descent Mordvintsev et al. 2020. However, training NCAs through differentiable programming requires an already given target structure, which we often do not have available. Most

of the time, we might have specific functional requirements instead (e.g. a robot of any shape that locomotes fast), and as we will see in the next section, for this type of creative discovery, neuroevolution is very powerful.

### 7.3.2 Growing functional machines

In the previous section we have seen that NCAs can be evolved to grow unanimated artefacts, such as 2D patterns. However, in nature entire organisms are grown from a single cell that move and interact with the world. Additionally, as a consequences of the developmental recipes, such systems constantly renew their cells and have the ability to repair themselves. Can NCAs be extended to accomplish similar feats?

In this section, we revisit the domain introduced in Section 4.3.1 where we explored how CPPNs can be used to encode the morphology of soft, mobile robots. In that work, a CPPN was queried with the location of each voxel and would then output a voxel material type. CPPNs were able to create high-performing soft robots with regular patterns such as symmetry and repetition. However, each voxel needed access to its global location in space and while this is not necessarily a problem in simulated soft-robots, in a modular physical robots (where each module is identical), this information might not be directly available. Can we design soft robots using a collective approach, where each voxel determines its material solely through local cell-to-cell communication? Drawing parallels with biological systems, each cell should be able to determine its function through local interactions alone.

Here we will look at such a completely distributed approach, which is based on evolving NCAs (Horibe, Walker, and Risi 2021). In this example, the NCA is a rather simple neural network with a fixed topology consisting of three layers. The input dimension of the neural network is $3 \times 9 \times 2 = 54$. The hidden layer is set to 64. The neural network has five outputs that determine the next state (i.e. material type) of each voxel such as muscle or bone and one output that determines if the cell is alive. The same neural network is applied to each voxel neighboring a voxel that is already alive. Robots are grown from an initial seed cell in the center position of the 3D grid for a certain amount of timesteps until they are placed in the simulation environment. This results in the robot's voxel materials to be actuated and they can be tested for their ability to locomote. Instead of using NEAT, the parameters of these networks with a fixed architecture are evolved through a simple genetic algorithm, in which parents are selected uniformly at random. Genomes are mutated by adding Gaussian noise to the neural network's weight vectors. The GA performs simple truncation selection, with the top $T$ individual becoming the parents for the next generation. Following a technique called elitism, top Mth individuals are passed on to the next generation without mutation.

Similar to the CPPN-encoded soft robots, evolved NCAs are able create high-performing 3D soft robots through a process of growth and local communication alone. However, unlike CPPN they are able to do so without requiring a global coordinate frame. Some of the example grown robots are shown in Figure 7.6, top. Once grown, the creatures display different walking gates, such as the L-Walker that resembles an L-shaped form; it moves by opening and closing the front and rear legs connected to its pivot point at the bend of the L or the Crawler, which has multiple short legs and its legs move forward in concert.

Collective systems offer the advantage of being highly resilient to perturbances and disruptions, as they are designed with built-in redundancies and lack a single point of failure.

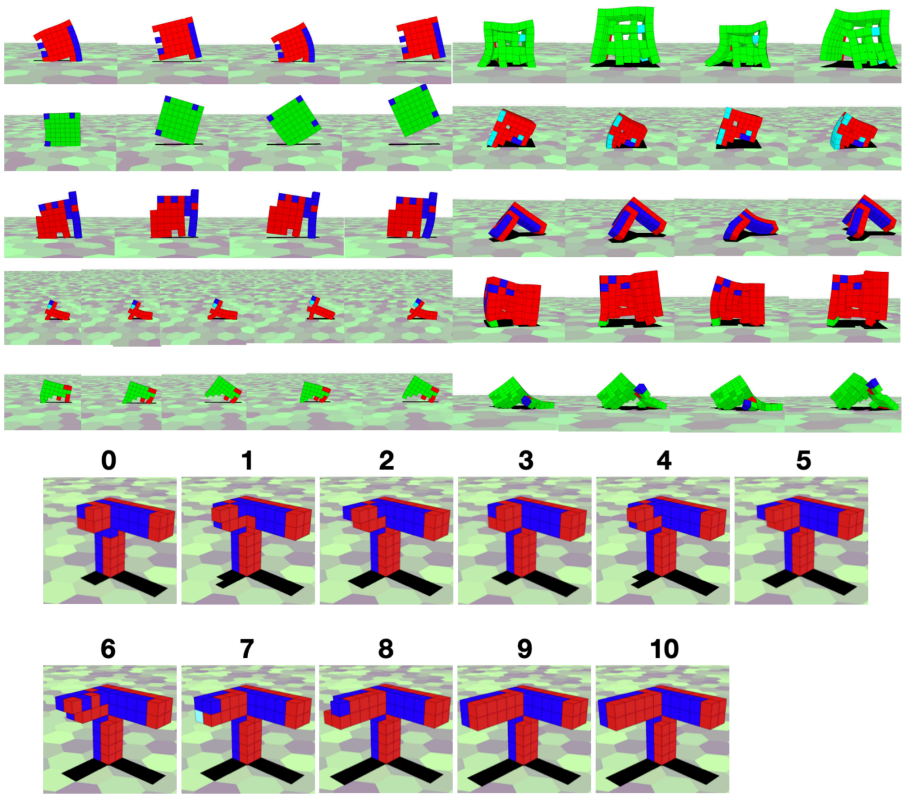



Figure 7.6: **Examples of grown soft robots (top) and recovery from morphological damage (bottom)**. Evolution was able to discovery a variety of NCAs that were able to grow 2D and 3D soft voxel robots with different walking gaits. After damage, a second NCA is able to regrow damage components, solely through the local communication of cells. (Figures from Horibe, Walker, and Risi 2021)

For example, the morphogenetic systems of many biological organisms gives them amazing regenerative capabilities, allowing them to repair and reconfigure their morphology in response to damage or changes in components. Primitive organisms such as Hydra and Planaria are particularly capable of regeneration and can thus achieve complete repair, no matter what location of the body part is cut off (Beane et al. 2013). But also more complex creatures such as salamanders are capable of regenerating an amputated leg. Can our artificial collective system show a similar kind of resilience and adaptability?

To explore this question, we can remove parts of the fully developed robots and rerun the same NCA for several developmental steps to observe whether the damaged areas regenerate. As it turns out, it was not possible to evolve one NCa that controls both the initial growth and the damage recovery. However, training a second NCA whose sole purpose it is to regrow a damaged morphology showed more success. In other words, one neural CA grows the initial morphology and the other CA is activated once the robot is damaged. This way, robots were often able to regrow damaged components, which allowed

them to restore their ability to locomote (Figure 7.6,bottom). Nevertheless, small discrepancies in the restored morphology could lead to significant loss of locomotion ability. In Section 7.3.5, we will revisit this task and explore how the synergistic integration of neuroevolution and differentiable programming can ultimately enable the same neural network to not only grow a robot but also facilitate a higher accuracy in damage and locomotion recovery.

### 7.3.3 Case study: Evolving Video Game Levels with NCAs and QD

So far in this chapter we have explored approaches where the goal is to grow *one* particular artefact that satisfies certain functional or visual criteria. To evolve a diversity of designs, such as the robot morphology, the algorithm needed to be run multiple times from scratch. In this section, we'll look at a case study that evolves a diversity of neural cellular automata with a QD-algortihm with the goal to generate a diversity of different video game levels (Earle et al. 2022). Level generation can serve as a good benchmark for evolving NCAs because such artefacts often need to satisfy a diverse range of criteria, from being aesthetically pleasing, to fun to play, and after all functional (i.e. a level needs to be playable). Additionally, we have seen in Section 7.3.1 that it can be difficult to learn to control the complex dynamics of a self-organizing system such as NCAs to grow into a particular target shape. Because QD algorithms can take advantage of stepping stones discovered along the way, we will see that they are better able to navigate this complex fitness landscapes.

A well-suited video game to study these algorithms is the old school *The Legend of Zelda*[1]. In the simplified Zelda clone in these experiments, the agent has to navigate 2D levels and locate the key that will open the level's exit door, while killing monsters. Zelda levels often show some level of symmetry, and therefore symmetry (both horizontal and vertical) in addition to the path-length from the goal to the exit, were chosen as the MAP-Elites dimensions of interest. In a straightforward application of QD and NCAs, each elite in the map would be an NCA that produces a map with a particular level of symmetry and path-lengh. However, a designer would ideally have more than one level with a specific path-length to chose from. To address this issue, each NCA can be treated as a whole level "generator" and tested for its ability to generate a diversity of different levels with the same path-length, given different random initial states as input.

With the QD dimensions defined, we need a measure for the quality of each NCA generator, which is evaluated based on three different criteria: validity, reliability, and intra-generator diversity. The validity term quantifies how well the generated level conforms to the soft constraints of the particular game. For example, in the case of Zelda this constrain means levels should form one connected region with the generator receiving a lower score for each additional region that is not connected to the main region. The reliability terms captures how reliable one NCA generates structures with a particular QD measure. For example, an NCA in Zelda is penalized if it produces levels with very different path length every time it generates a new level from a different initial state. The last term, intra-generator diversity measures the amount of diversity in a batch of levels generated by the same NCA (given different starting seeds). The term was added to prevent generators from ignoring the latent seed input and collapsing to only producing one particular level design.

1. Copyright (c) 1986 Nintendo.

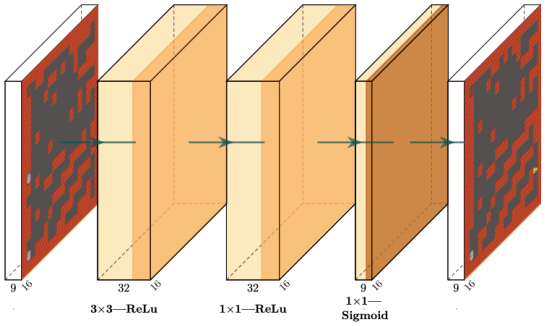



Figure 7.7: **NCA Architecture for Game Level Generation.** A convolutional network repeatedly transform levels based on the local interaction of $3 \times 3$ cells. Levels are evaluated after being modified for a fixed number of iterations. (Figures from Earle et al. 2022)

These three terms are then ultimately combined to measure the quality of a particular NCA, with the goal to have a generator that produces a distribution of valid levels with reliable behavioral descriptors.

A detailed view of the NCA architecture is shown in Figure 7.7. It comprises 3 convolutional layers, utilize ReLU and sigmoid activation functions, and has 32 hidden channels. The NCA's output retains the dimensions and channel count of its input. However, it employs an arg max function on a channel-by-channel basis to yield a discrete representation of the subsequent state. To generate a game level using an NCA, we input a onehot-encoded random starting level (also termed as "latent seed"). This process is reiterated using the NCA's output until the level either stabilizes or reaches a predetermined step limit. The QD algorithm used to evolve the NCAs is a variant of the classical Map-Elites algorithm we have encountered in Section 5.4. It uses CMA-ME (Fontaine et al. 2020), which combines the MAP-Elites like archiving of solutions with the adaptation mechanism of CMA-ES, which is particularly well suited for continuous domains.

The approach is able to grow a diversity of levels along the dimensions of interest path-length and symmetry (Figure 7.8). The maps are all solvable, satisfying the required game constraints such as only producing one key, one door, and one avatar. One interesting question is: How does the NCA approach compare to a CPPN-like generation of levels, which does not go through the process of growth? QD-algorithms are particularly well suited to compare different representation since they can illuminate how well the approach covers the search space along dimensions of interest. To make the comparison fair, each CPPN also needs to become a "generator", allowing it produce not just one map but multiple ones. This can be achieved by augmenting the CPPN with a latent vector input, in addition to the typical $x$, $y$ coordinates.

Counter-intuitively, the NCA-based approach was able to explore a larger space of the levels and the individual generators produced more diverse outputs than the CPPN-based encoding and an additional variational autoencoder (VAE) -inspired decoder architecture. One would assume that having global information would in fact make it easier to produce a diversity of levels. How is the NCA able to produce global coherence without the global information available to a CPPN or VAE decoder? Looking at a level growth sequence

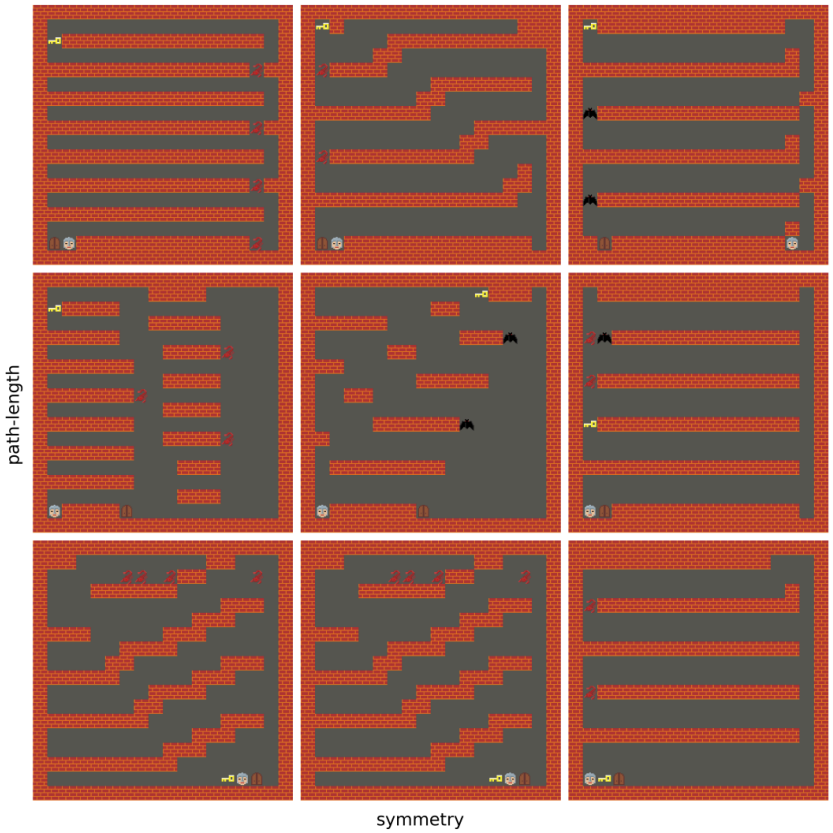



Figure 7.8: **NCA grown Zelda levels.** The MAP-Elites based approach is able to evolve NCA that are able to produce a variety of different Zelda maps along the dimensions path-length and symmetry. (Figures from Earle et al. 2022)

reveals some interesting insights (Figure 7.9). During the intermediate growth process, we can see that the levels often contains multiple keys or doors but at the end the process converges towards a solution with just one key and one door. These intermediate tiles seems to be used as a type of external memory, which propagate spatial information across the level to form patterns with global complexity. Surprisingly, through these iterative local interactions alone, the NCA is able to generate levels that satisfy high-level functional constraints.

Producing patterns with global coherence through local interactions alone is an essential ability seen in many collective intelligence systems in nature. In the next section we will investigate the opportunities of such advances for the growth of neural networks themselves.

### 7.3.4 Neural Developmental Programs

One of the most impressive feeds of a collective system cooperating is the self-assembling of billions of cells into a human brain. While most current neural networks in machine learning are hand-designed and learning is restricted to optimizing connection weights,

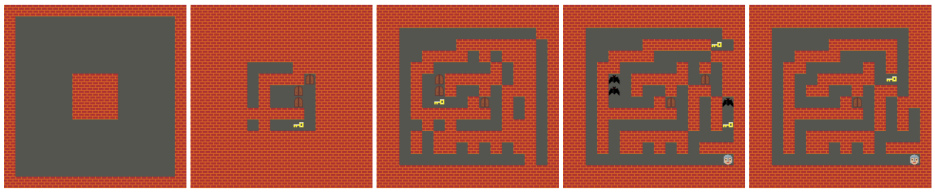



Figure 7.9: **NCA Level Growth.** Shown are the intermediate growth states of a Zelda level. The growth process starts with a fixed initial seed at the center of the level until a stable configuration is reached. The final level has a high path-length, while satisfying the Zelda-game level constraints. (Figures from Earle et al. 2022)

biological neural networks are grown through a process of local communication and self-organization. In the previous sections we have seen that NCAs can learn to grow 2D structures, game levels, and even locomoting 3D soft robotics. Can they also learn to grow and self-assemble an artificial neural network?

In Section 4.2.2 on grammatical indirect encodings, we have encountered early work in this direction with an approach called cellular encodings (Gruau and Whitley 1993; Gruau, Whitley, and Pyeatt 1996). In a cellular encoding, a program evolved through genetic programming guides the growth of a policy network. This pioneering work was maybe ahead of its time, with direct encodings such as NEAT being able to outperform it in terms of the number of evaluations needed to find a solution for simple tasks such as pole balancing. The cellular encoding approach has therefore been less well adopted than conceptually simpler and more direct encoding approaches.

However, with the recent advances in training NCAs to produce complex patterns more efficiently, a cellular encoding based on neural networks (instead of GP), could potentially serve as a powerful indirect encoding. Related approach such as ES-HyperNEAT also progressively grow networks (Section 4.3.4), but do not take advantage of the collective collaboration between cells during the growth process. In nature, these abilities seem essential in enabling the remarkable robustness and adapability of collective intelligent systems.

A step towards this direction are Neural Developmental Program (NDP) approaches (Najarro, Sudhakaran, and Risi 2023). NDPs are building on the ideas behind neural CAs by extending them to growing graph-like structures. The basic idea is that copies of the same neural network-based program (the NDP) run in each neuron of the policy network, learning how the policy network should grow and adapt based on each neurons' local information received from its neighbors. Starting from a single neuron at the start of each agent's lifetime, the NDP needs to grow a complete policy network able to solve the task at hand. Note that while the approach grows a neural architecture, the goal here is different to techniques like NEAT and the other neural architecture search methods we will have a closer look at in Chapter 10. While these methods change the architecture of the neural networks during evolution, the idea in NDPs is to grow neural networks during a developmental phase. The benefits of this approach is that the development of the neural network can be shaped by the experience and take advantage of sensory information from the environment to drive the neural developmental process.

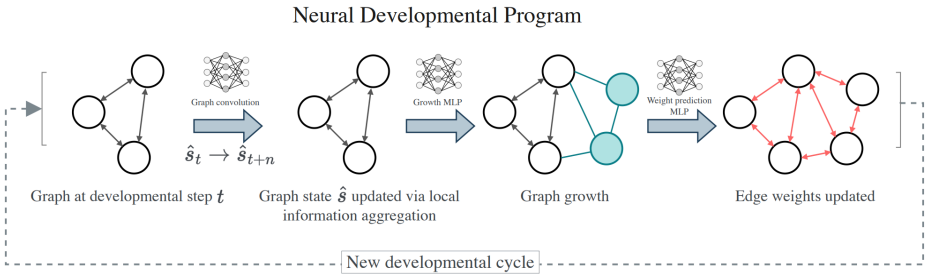



Figure 7.10: **Neural Developmental Program Approach.** During the stage of information aggregation, the graph systematically transmits the state $s$ of each node to its adjacent nodes over $n$ iterations. The replication model MLP takes as input the updated node state $s_{t+n}$ and decides which nodes should replicate. Another MLP comes into play to deduce the weights of the edges connecting each node pair, using their combined embeddings. Once the network is grown for the given number of developmental steps, it is then evaluated to solve a specific task. (Figures from Najarro, Sudhakaran, and Risi 2023)

A more detailed view of the NDP approach is shown in Figure 7.10. Each node in the growing graph has an internal state vector, whose values are updated during the developmental process based on the local communication between nodes. The NDP has thee MLPs: One of these MLPs is responsible for updating the aforementioned hidden states of the nodes, while a second MLP a state of a node as input and predicts if this node should replicate. The third MLP takes the state of two hidden nodes as input and outputs the edge weight between them.

A good initial test to evaluate the expressiveness of these NDPs is to task them with growing graphs with properties found in many biological neural networks. One predominant topological characteristics of these biological networks is small-worldness, which means networks that are characterized by small average shortest path length and and relatively larger clustering co-efficient. And in fact, optimizing an NDP directly for these two properties with CMA-ES does indeed lead to a graph satisfying the small-worldness criteria. A more complex task involves optimizing the NDP to grow a policy neural network that enables an agent to interact successfully with its environment. When applied to various control tasks such as CartPole, LunarLander and HalfCheetah, CMA-ES is able to find high-performing NDPs. Looking into the growth sequence of one of these network for the CartPole balancing tasks, shows the rapid profiliation of nodes during the first few developmental stages (Figure 7.11).

While there are many open research direction in regards to developing more powerful NDPs, the fact that NDP can capture some of the fundamental patterns seen in biological networks through self-organisation and local growth alone suggest they can be a good base for further exploration. For example, the NDP model can be used to study diversity maintenance in neural population. And in fact, a key issue with training the original NDPs is that if all neurons differentiate into the same type, growth-related decisions become uniform, leading to homogeneous ANN structures incapable of producing complex behaviors. Two biological-inspired key modifications can resolve this issue (Nisioti et al. 2024). First,

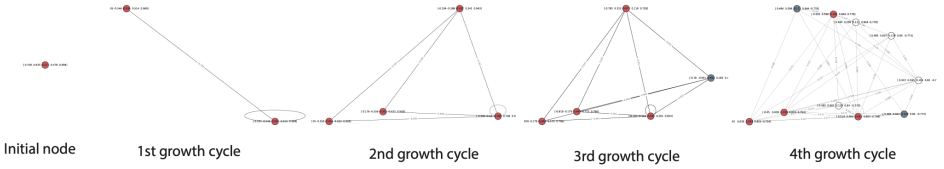



Figure 7.11: **NDP growth of a network solving the cartpole task.** The networks begins as a solitary node and progressively develops into a more complex network, encompassing 2, 4, 5, and ultimately 10 neurons, along with 33 weighted edges, over the course of four growth stages. Within this network, the red nodes function as sensory neurons, the white nodes serve as hidden neurons, and the blue nodes operate as output neurons. Above each neuron, there is a vector displayed, representing the node embeddings. These embeddings are indicative of the state of each neuron throughout the stages of network development. (Figures from Najarro, Sudhakaran, and Risi 2023)

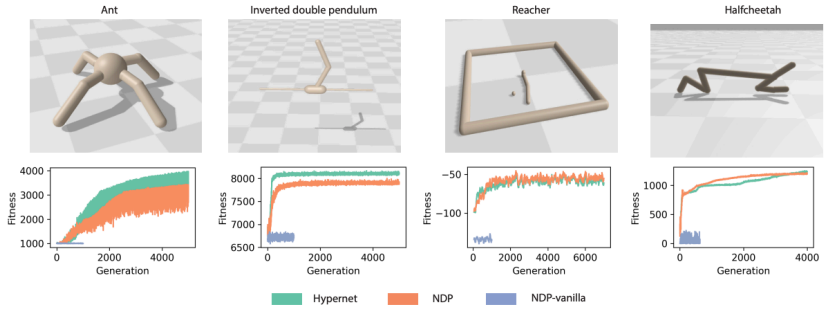



Figure 7.12: **NDP performance across tasks**. The original vanilla NDP is compared to a version that includes intrinsic states and lateral inhibition. These two key mechanisms allow the NDP to perform significantly better in more complex domains. (Figures from Nisioti et al. 2024)

introducing intrinsic states that remain unchanged during growth ensures that diversity is preserved in the network. By initializing networks with a small set of cells, each with a distinct intrinsic state, we introduce diversity at the start of growth. As the network expands, these intrinsic states are replicated, resulting in cell lineages similar to biological networks. Second, lateral inhibition, a mechanism believed to play a crucial role in maintaining diversity during biological development. This mechanism prevents neighboring cells from taking similar actions for a limited number of steps when one cell makes a decision. Both of these mechanisms combined allow the NDP to perform much better, and similarly to a Hypernet-based approach across a diversity of more complex control tasks such as the ant, inverted double pendulum, reacher, and halfcheetah (Figure 7.12).

Another key limitation of the original NDP model is that it was temporally constrained to a pre-environmental phase and did not account for an agent's lifetime, let alone lifelong learning. That is, the networks were grown during a developmental phase but static while the agent interacts with the environment. However, as we will explore more in Section 12.3, for many tasks lifetime adaptation is critical. A Lifelong NDP version (LNDP) introduce a

mechanism that enables plasticity and structural adaptation throughout an agent's lifetime Plantec et al. 2024. This is achieved through local computations based on the activity of individual neurons in the ANN and the global reward signals from the environment. The resulting system establishes a family of plastic neural networks that bridges the gap between indirect developmental encodings and meta-learned plasticity rules. This method perform similarly to the original NDP on tasks not requiring lifetime adaptation such as Cartpool. However, when applied to a foraging task that necessitates the agent to learn and remember the position of a randomly placed food source, the LNDP performs significantly better.

More broadly, the NDP highlights the differences between approaches that are based on bottom-up self-organization vs. the established top-down engineering. While these approaches have yet to be able to compete with current state-of-the-art methods, they offer an exciting alternative to achieving more robust and adaptive forms of neural networks.

### 7.3.5 Synergistic Combinations of Neuroevolution and Differentiable Programming

Neuroevolution tends to generally work better when it is less constrained, taking advantage of the power of evolution for creative discovery. For example, we have seen that neuroevolution is well suited to evolve neural networks that grow soft robots able to locomote or video game levels with interesting properties. However, these algorithms can struggle when tasked to reevolve certain target pattern that requires traversing many different stepping stones (Section 5.3). The same is true for evolving morphogenetic systems that are tasked with growing more complex target pattern.

If a target is given, such as a particular 2D or 3D structure, it makes sense to take advantage of efficient differentiable programming to directly optimize for growing that target. For example, NCA can be trained efficiently through gradient descent to grow certain 2D images (Mordvintsev et al. 2020) or even functional 3D Minecraft structures that can regrow damaged components (Sudhakaran et al. 2021) (Figure 7.13).

Returning to the task of evolving NCA to create resilient soft robots showcases an interesting approach that combines the benefits of evolution for creative discovery and differentiable programming for efficient optimization (Horibe et al. 2022). Figure 7.14 shows an overview of this hybrid approach, which works as follows: (1) A diversity of morphologies are discovered through evolutionary optimization. (2) A neural cellular automata is trained to regrow a target morphology found by evolution under different damages through differentible programming. (3) The resulting NCA is able to grow a soft robot, while being able to recover from extreme forms of damage.

In summary, combining evolutionary algorithms with differentiable programming techniques offers a promising approach for developing systems that are both innovative and resilient. Evolutionary processes excel at exploring a vast search space of potential solutions, producing a diversity of designs and behaviors that are often not achievable through gradient-based methods alone. This creative potential is particularly advantageous in open-ended domains like soft robotics, where unconventional solutions can emerge. On the other hand, once a target design or structure is identified, differentiable programming shines in its ability to fine-tune and optimize the system efficiently, enabling robust growth and regeneration capabilities.

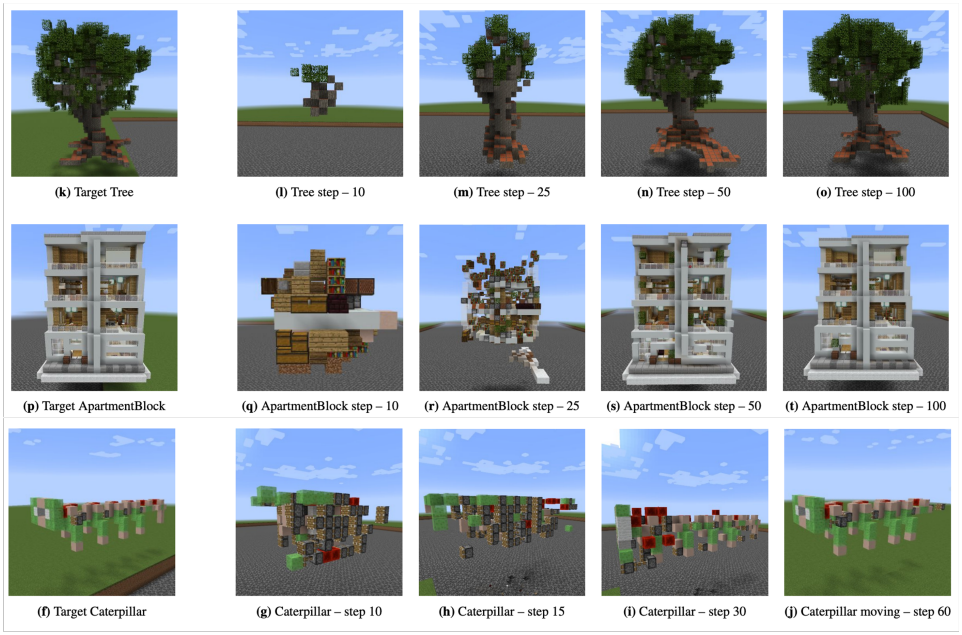



Figure 7.13: **Learning to grow different 3D target structures.** An NCA is trained through gradient descent to grow given target pattern. The approach is able to grow both static structures (k and p) and functional machines, such as a locomoting caterpillar (f). The caterpillar can even regenerate into two creatures, when cut in half. (Figures from Sudhakaran et al. 2021)

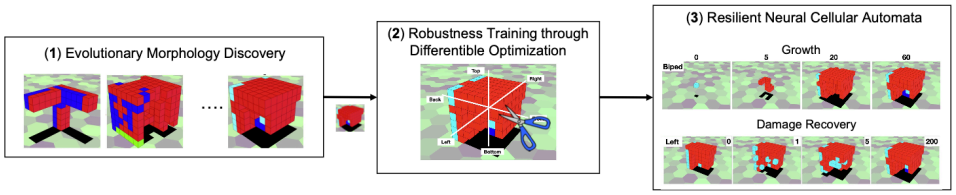



Figure 7.14: **Combining Evolutionary Discovery and Differentible Programming.** (1) Evolutionary optimization is used to discover a wide range of diverse morphologies. (2) A neural cellular automaton (NCA) is then trained to regenerate these target morphologies, even after different types of damage. (3) The trained NCA can successfully grow a soft robot and recover it from severe damage. (Figures from Horibe et al. 2022)

## 7.4 Chapter Review Questions

1. **Conceptual Understanding:** What are the fundamental differences between cooperative and competitive coevolution, and how do they contribute to neuroevolution?
2. **Cooperative Coevolution:** Describe the concept of shared fitness in cooperative coevolution. How does it ensure effective collaboration among components?

3. **Evolving Single Neural Networks:** How does the ESP system (Enforced Subpopulations) improve upon the SANE system in evolving neural networks?
4. **Specialization in Subpopulations:** Why is redundancy within subpopulations important in the context of ESP, and how does it lead to robust networks?
5. **Evolving Teams:** In the predator-prey scenario, how do stigmergy-based coordination strategies lead to effective team behaviors without direct communication?
6. **Competitive Coevolution:** How does competitive coevolution establish an open-ended fitness function, and what challenges does it face in ensuring progress?
7. **Evolutionary Arms Race:** Using the zebras and hyenas example, explain how alternating advantages between predator and prey populations drive increasingly complex behaviors.
8. **Cellular Automata:** What role do local interactions play in the emergence of complex patterns in Cellular Automata (CAs), and how are these principles applied to neural CAs?
9. **Applications of Neural CAs:** How can Neural Cellular Automata (NCAs) be used to solve tasks like the French flag problem or pattern replication? What are their advantages over traditional approaches?
10. **Evolving Resilient Systems:** Explain the hybrid approach combining neuroevolution and differentiable programming for growing and regenerating resilient soft robots. How does each method contribute to the overall system's functionality?