Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions

Rui Wang **Joel Lehman** **Jeff Clune*** **Kenneth O. Stanley***
Uber AI Labs
San Francisco, CA 94103
ruiwang,joel.lehman,jeffclune,kstanley@uber.com
*co-senior authors

Abstract

While the history of machine learning so far largely encompasses a series of problems posed by researchers and algorithms that learn their solutions, an important question is whether the problems themselves can be generated by the algorithm at the same time as they are being solved. Such a process would in effect build its own diverse and expanding curricula, and the solutions to problems at various stages would become stepping stones towards solving even more challenging problems later in the process. The *Paired Open-Ended Trailblazer* (POET) algorithm introduced in this paper does just that: it pairs the generation of environmental challenges and the optimization of agents to solve those challenges. It simultaneously explores many different paths through the space of possible problems and solutions and, critically, allows these stepping-stone solutions to transfer between problems if better, catalyzing innovation. The term *open-ended* signifies the intriguing potential for algorithms like POET to continue to create novel and increasingly complex capabilities without bound. We test POET in a 2-D bipedal-walking obstacle-course domain in which POET can modify the types of challenges and their difficulty. At the same time, a neural network controlling a biped walker is optimized for each environment. The results show that POET produces a diverse range of sophisticated behaviors that solve a wide range of environmental challenges, many of which cannot be solved by direct optimization alone, or even through a direct-path curriculum-building control algorithm introduced to highlight the critical role of open-endedness in solving ambitious challenges. The ability to transfer solutions from one environment to another proves essential to unlocking the full potential of the system as a whole, demonstrating the unpredictable nature of fortuitous stepping stones. We hope that POET will inspire a new push towards open-ended discovery across many domains, where algorithms like POET can blaze a trail through their interesting possible manifestations and solutions.

1 Introduction

Machine learning algorithms are often understood as tools for solving difficult problems. Challenges like image classification and associated benchmarks like ImageNet [1] are chosen by people in the community as solvable and researchers focus on them to invent ever-more high-performing algorithms. For example, ImageNet was proposed in 2009 and modern deep neural networks such as *ResNet* [2] began to beat humans in 2015 [3, 4]. In reinforcement learning (RL) [5], learning to play Atari was proposed in 2013 [6] and in recent years deep reinforcement learning (RL) algorithms reached and surpassed human play across a wide variety of Atari games [7–12]. The ancient game of Go has been a challenge in AI for decades, and has been the subject of much recent research: AlphaGo and its

variants can now reliably beat the world's best human professional Go players [13–15]. Each such achievement represents the pinnacle of a long march of increasing performance and sophistication aimed at solving the problem at hand.

As compelling as this narrative may be, it is not the only conceivable path forward. While the story so far relates a series of challenges that are conceived and conquered by the community algorithm by algorithm, in a more exotic alternative only rarely discussed (e.g. [16, 17]), the job of the algorithm could be to conceive *both* the challenges and the solutions at the same time. Such a process offers the novel possibility that the march of progress, guided so far by a sequence of problems conceived by humans, could lead *itself* forward, pushing the boundaries of performance autonomously and indefinitely. In effect, such an algorithm could continually invent new environments that pose novel problems just hard enough to challenge current capabilities, but not so hard that all gradient is lost. The environments need not arrive in a strict sequence either; they can be invented in parallel and asynchronously, in an ever-expanding tree of diverse challenges and their solutions.

We might want such an *open-ended* [18–22] process because the chain that leads from the capabilities of machine learning today to e.g. general human-level intelligence could stretch across vast and inconceivable paths of stepping stones. There are so many directions we could go, and so many problems we could tackle, that the curriculum that leads from here to the farthest reaches of AI is beyond the scope of our present imagination. Why then should we not explore algorithms that *self-generate* their own such curricula? If we could develop and refine such algorithms, we might find a new approach to progress that relies less upon our own intuitions about the right stepping stones and more upon the power of automation.

In fact, the only process ever actually to achieve intelligence at the human level, natural evolution, *is* just such a self-contained and open-ended curriculum-generating process. Both the problems of life, such as reaching and eating the leaves of trees for nutrition, and the solutions, such as giraffes, are the products of the same open-ended process. And this process unfolds not as a single linear progression, but rather involves innumerable parallel and interacting branches radiating for more than a billion years (and is still going). Nature is also a compelling inspiration because it has avoided convergence or stagnation, and continues to produce novel artifacts for beyond those billion years. An intriguing question is whether it is possible to conceive an algorithm whose results would be worth waiting a billion years to see. Open-ended algorithms in their most grandiose realization would offer this possibility.

While a small community within the field of artificial life [18, 19, 21–28] has studied the prospects of open-ended computation for many years, in machine learning the prevailing assumption that algorithms should be *directed* (e.g. towards the performance objective) has pervaded the development of algorithms and their evaluation for many years. Yet this explicitly directed paradigm has begun to soften in recent years. Researchers have begun to recognize that modern learning algorithms' hunger for data presents a long-term problem: the data available for progress diminishes as the appropriate tasks for leveling up AI competencies increase in complexity. This recognition is reflected in systems based on *self-play* (which is related to *coevolution* [29–31]), such as competitive two-player RL competitions [14, 32] wherein the task is a function of the competition, which is itself changing over time. Generative adversarial networks (GANs) [33], in which networks interact as adversaries, similarly harness a flavor of self-play and coevolution [34] to generate both challenges and solutions to those challenges.

While not explicitly motivated by open-endedness, recognition of the importance of self-generated curricula is also reflected in recent advances in automatic curriculum learning for RL, where the intermediate goals of curricula are automatically generated and selected. A number of such approaches to automatic curriculum building have been proposed. For example, the curriculum can be produced by a generator network optimized using adversarial training to produce tasks at the appropriate level of difficulty for the agent [35], by applying noise in action space to generate a set of feasible start states increasingly far from the goal (i.e. a reverse curriculum) [36], or through intrinsically motivated goal exploration processes (IMGEPs) [16]. The POWERPLAY algorithm offers an alternative approach to formulating continual, concurrent searches for new tasks through an increasingly general problem solver [17]. In Teacher-Student Curriculum Learning [37], the teacher automatically selects sub-tasks that offer the highest slope of learning curve for the student. Ideas from the field of *procedural content generation* (PCG) [38, 39], where methods are developed to generate diverse sets of levels

and other contents for games, can also generate progressive curricula that help improve generality of deep RL agents [40].

The ultimate implication of this shift towards less explicitly-directed processes is to tackle the full grand challenge of open-endedness: Can we ignite a process that on its own unboundedly produces increasingly diverse and complex challenges at the same time as solving them? And, assuming computation is sufficient, can it last (in principle) forever?

The *paired open-ended trailblazer* (POET) algorithm introduced in this paper aims to confront open-endedness directly, by evolving a set of diverse and increasingly complex environmental challenges at the same time as collectively optimizing their solutions. A key opportunity afforded by this approach is to attempt *transfers* among different environments. That is, the solution to one environment might be a stepping stone to a new level of performance in another, which reflects our uncertainty about the stepping stones that trace the ideal curriculum to any given skill. By radiating many paths of increasing challenge simultaneously, a vast array of potential stepping stones emerges from which progress in any direction might originate. POET owes its inspiration to recent algorithms such as minimal criterion coevolution (MCC) [27], which shows that environments and solutions can effectively co-evolve, novelty search with local competition [41], MAP-Elites [42, 43], Innovation Engines [44], which exploit the opportunity to transfer high-quality solutions from one objective among many to another, and the combinatorial multi-objective evolutionary algorithm (CMOEA), which extends the Innovation Engine to combinatorial tasks [45, 46]. POET in effect combines the ideas from all of these approaches to yield a new kind of open-ended algorithm that optimizes, complexifies, and diversifies as long as the environment space and available computation will allow.

In this initial introduction of POET, it is evaluated in a simple 2-D bipedal walking obstacle-course domain in which the form of the terrain is evolvable, from a simple flat surface to heterogeneous environments of gaps, stumps and rough terrain. The results establish that (1) solutions found by POET for challenging environments cannot be found directly on those same environmental challenges by optimizing on them only from scratch; (2) neither can they be found through a curriculum-based process aimed at gradually building up to the same challenges POET invented and solved; (3) periodic transfer attempts of solutions from some environments to others–also known as "goal switching" [44]–is important for POET's success; (4) a diversity of challenging environments are both invented and solved in the same single run. While the 2-D domain in this paper offers an initial hint at the potential of the approach for achieving open-ended discovery in a single run, it will be exciting to observe its application to more ambitious problem spaces in the future.

The paper begins with background on precedent for open-endedness and POET, turning next to the main algorithmic approach. It finally introduces the 2-D obstacle-course domain and presents both qualitative an quantitative results.

2 Background

This section first reviews several approaches that inform the idea behind POET of tracking and storing stepping stones, which is common in evolutionary approaches to behavioral diversity. It then considers a recently proposed open-ended coevolutionary framework based on the concept of minimal criteria and, lastly, evolution strategies (ES) [47], which serves as the optimization engine behind POET in this paper (though in principle other reinforcement learning algorithms could be substituted in the future).

2.1 Behavioral Diversity and Stepping Stones

A common problem of many stochastic optimization and search algorithms is becoming trapped on local optima, which prevents the search process from leaving sub-optimal points and reaching better ones. In the context of an open-ended search, becoming trapped is possible in more than one way: In one type of failure, the domains or problems evolving over the course of search could stop increasing their complexity and thereby stop becoming increasingly interesting. Alternatively, the simultaneously-optimizing *solutions* could be stuck at sub-optimal levels and fail to solve challenges that are solvable. Either scenario can cause the search to stagnate, undermining its open-endedness.

Population-based algorithms going back to novelty search [48] that encourage *behavioral* (as opposed to genetic) diversity [42–46, 49] have proven less susceptible to local optima, and thus naturally align

more closely with the idea of open-endedness as they focus on *divergence* instead of *convergence*. These algorithms are based on the observation that the path to a more desirable or innovative solution often involves a series of waypoints, or *stepping stones*, that may *not* increasingly resemble the final solution, and are not known ahead of time. Therefore, divergent algorithms reward and preserve diverse behaviors to facilitate the preservation of potential stepping stones, which could pave the way to both a solution to a particular problem of interest and to genuinely open-ended search.

In the canonical example of *novelty search* (NS) [48], individuals are selected purely based on how different their behaviors are compared to an archive of individuals from previous generations. In NS, the individuals in the archive determine the *novelty* of a solution, reflecting the assumption that genuinely novel discoveries are often stepping stones to further novel discoveries. Sometimes, a chain of novel stepping stones even reaches a solution to a problem, even though it was never an explicit objective of the search. NS was first shown effective in learning to navigate deceptive mazes and biped locomotion problems [48].

Other algorithms are designed to generate, retain, and utilize stepping stones more explicitly. To retain as much diversity as possible, quality diversity (QD) algorithms [41–43, 49] keep track of many different niches of solutions that are (unlike pure NS) being optimized simultaneously and in effect try to discover stepping stones by periodically testing the performance of offspring from one niche in other niches, a process referred to as *goal switching* [44].

Based on this idea, an approach called the Innovation Engine [44] is able to evolve a wide range of images in a single run that are recognized with high-confidence as different image classes by a high-performing deep neural network trained on ImageNet [50]. In this domain, the Innovation Engine maintains separate niches for images of each class, including the image that performed the best in that class (i.e. produced the highest activation value according to the trained deep neural network classifier). It then generates variants (perturbations/offspring) of a current niche champion and not only checks whether such offspring are higher-performing in that niche (class), but also checks whether they are higher-performing in any other niche. Interestingly, the evolutionary path to performing well in a particular class often passes through other (oftentimes seemingly unrelated) classes that ultimately serve as stepping stones to recognizable objects. In fact, it is *necessary* for the search to pass through these intermediate, seemingly unrelated stages to ultimately satisfy other classes later in the run [44]. More recently, to evolve multimodal robot behaviors, the *Combinatorial Multi-Objective Evolutionary Algorithm* (CMOEA) [45, 46] builds on the Innovation Engine, but defines its niches based on different *combinations of subtasks*. This idea helps it to solve both a multimodal robotics problem with six subtasks as well as a maze navigation problem with a hundred subtasks, highlighting the advantage of maintaining a diversity of pathways and stepping stones through the search space when a metric for rewarding the most promising path is not known. POET will similarly harness goal-switching within divergent search.

While the general ideas of promoting diversity and preserving stepping stones grew up initially within the field of evolutionary computation, they are nevertheless generic and applicable in fields such as deep reinforcement learning (RL). For example, a recent work utilizes diversity maximization to acquire complex skills with minimal supervision, which improves learning efficiency when there is no reward function. The results then become the foundation for imitation learning and hierarchical RL [51]. To address the problem of reward sparsity, Savinov et al. [52] extend the archive-based mechanism from NS that determines novelty in behaviors to determine novelty in the observation space and then to formulate the novelty bonus as an auxiliary reward. Most recently, the Go-Explore algorithm by Ecoffet et al. [12] represents a new twist on QD algorithms and is designed to handle extremely sparse and/or deceptive "hard exploration" problems in RL. It produced dramatic improvements over the previous state of the art on the notoriously challenging hard-exploration domains of Montezuma's Revenge and Pitfall. Go-Explore maintains an archive of interestingly different stepping stones (e.g. states of a game) that have been reached so far and how to reach them. It then returns to and explores from these stepping stones to find new stepping stones, and repeats the process until a sufficiently high-quality solution is found.

2.2 Open-Ended Search via Minimal Criterion Coevolution (MCC)

Despite the recent progress of diversity-promoting algorithms such as NS and QD, they remain far from matching a genuinely open-ended search in the spirit of nature. One challenge in such algorithms is that although they provide pressure for ongoing divergence in the solution space, the

environment itself remains static, limiting the scope of what can be found in the long run. In this spirit, one fundamental force for driving open-endedness could come from *coevolution* [31], which means that different individuals in a population interact with each other while they are evolving. Such multi-agent interactions have recently proven important and beneficial for systems with coevolutionary-like dynamics such as generative adversarial networks (GANs) [33], self-play learning in board games [14], and competitions between reinforcement-learning robots [32]. Because the environment in such systems now contains other changing agents, the challenges are no longer static and evolve in entanglement with the solutions.

However, in most, if not all, coevolutionary systems, the abiotic (non-opponent-based) aspect of the environment remains fixed. No amount of co-evolution with a fixed task (e.g. pushing the other enemy off a pedestal, or beating them at Go) will produce general artificial intelligence that can write poetry, engage in philosophy, and invent math. Thus an important insight is that environments themselves (including ultimately the reward function) must too change to drive truly open-ended dynamics. Following this principle, an important predecessor to the POET algorithm in this paper is the recent *minimal criterion coevolution* (MCC) algorithm [27], which explores an alternative paradigm for open-endedness through coevolution. In particular, it implements a novel coevolutionary framework that pairs a population of evolving *problems* (i.e. environmental challenges) with a co-evolving population of *solutions*. That enables new problems to evolve from existing ones and for new kinds of solutions to branch from existing solutions. As problems and solutions proliferate and diverge in this ongoing coupled interaction, the potential for open-endedness arises.

Unlike conventional coevolutionary algorithms that are usually divided between competitive coevolution, where individuals are rewarded according to their performance against other individuals in a competition [29], and cooperative coevolution, where instead fitness is a measure of how well an individual performs in a cooperative team with other evolving individuals [30], MCC introduce a new kind of coevolution that evolves two interlocking populations whose members earn the right to reproduce by satisfying a *minimal criterion* [25, 26, 53] with respect to the other population, as both populations are gradually shifting simultaneously. For example, in an example in Brant and Stanley [27] with mazes (the problems) and maze solvers (the solutions, represented by neural networks), the minimal criterion for the solvers is that they must solve at least one of the mazes in the maze population, and the minimal criterion for the mazes is to be solved by at least one solver.

The result is that the mazes increase in complexity and the neural networks continually evolve to solve them. In a single run without any behavior characterization or archive (which are usually needed in QD algorithms), MCC can continually generate novel, diverse and increasingly complex mazes and maze solvers as new mazes provide new opportunities for innovation to the maze solvers (and vice versa), hinting at open-endedness. However, in MCC there is no force for *optimization* within each environment (or maze in the example experiment). That is, once a maze is solved there is no pressure to *improve* its solution; instead, it simply becomes a potential stepping stone to a solution to another maze. As a result, while niches and solutions proliferate, there is no pressure for *mastery* within each niche, so we do not see the best that is possible. Additionally, in MCC problems have to be solvable by the *current* population, so complexity can only arise through drift. As an example, imagine an environmental challenge E' that is similar to a currently solved problem E, but a bit harder such that none of the current agents in the population can solve it. Further imagine that if we took some of the current agents (e.g. the agent that solves E) and optimized them to solve E', at least one agent could learn to solve E' in a reasonable amount of time. MCC would still reject and delete E' if it was generated because none of the current population solves it right away. In contrast, in POET, we accept problems if (according to some heuristics) they seem likely to be improved upon and/or solved after some amount of dedicated optimization effort (so POET will likely keep E'), enabling a more direct and likely swifter path to increasingly complex, yet solvable challenges.

2.3 Evolution Strategies (ES)

In the POET implementation in this paper, ES plays the role of the optimizer (although other optimization algorithms should also work). Inspired by natural evolution, ES [54] represents a broad class of population-based optimization algorithms. The method referred to here and subsequently as "ES" is a version of ES popularized by Salimans et al. [47] that was recently applied with large-scale deep learning architectures to modern RL benchmark problems. This version of ES draws inspiration from Section 6 of Williams [55] (i.e. REINFORCE with multiparameter distributions), as well as

from subsequent population-based optimization methods including Natural Evolution Strategies (NES) [56] and Parameter-Exploring Policy Gradients (PEPG) [57]. More recent investigations have revealed the relationship of ES to finite difference gradient approximation [58] and stochastic gradient descent [59].

In the typical context of RL, we have an environment, denoted as $E(\cdot)$, and an agent under a parameterized policy whose parameter vector is denoted as w. The agent tries to maximize its reward, denoted as $E(w)$, as it interacts with the environment. In ES, $E(w)$ represents the stochastic reward experienced over a full episode of an agent interacting with the environment. Instead of directly optimizing w to maximize $E(w)$, ES seeks to maximize the expected fitness over a population of w, $J(\theta) = \mathbb{E}_{w \sim p_\theta(w)}[E(w)]$, where w is sampled from a probability distribution $p_\theta(w)$ parameterized by θ. Using the log-likelihood trick [55] and estimating the expectation above over n samples allows us to write the gradient of $J(\theta)$ with respect to θ as:

have to check ES by Salimans

$$\nabla_\theta J(\theta) \approx \frac{1}{n} \sum_{i=1}^{n} E(\theta_i) \nabla_\theta \log p_\theta(\theta_i).$$

Intuitively, this equation says that for every sample agent θ_i, the higher the performance of that agent $E(\theta_i)$, the more we should move θ in the direction of the gradient that increases the likelihood of sampling that agent, which is $\nabla_\theta \log p_\theta(\theta_i)$. Following the convention in Salimans et al. [47], $\theta_i, i = 1, \ldots, n$, are n samples of parameter vectors drawn from an isotropic multivariate Gaussian with mean θ and a covariance σ^2, namely, $\mathcal{N}(\theta, \sigma^2 I)$. (The covariance can be fixed or adjusted over time depending on the implementation.) Note that θ_i can be equivalently obtained by applying additive Gaussian noise $\epsilon_i \sim \mathcal{N}(0, I)$ to a given parameter vector θ as $\theta_i = \theta + \sigma\epsilon_i$. With that notation, the above estimation of the gradient of $J(\theta)$ with respect to θ can be written as:

$$\nabla_\theta J(\theta) \approx \frac{1}{n\sigma} \sum_{i=1}^{n} E(\theta + \sigma\epsilon_i)\epsilon_i.$$

Algorithm 1 illustrates the calculation of one optimization step of ES, which can efficiently be parallelized over distributed workers [47]. Once calculated, the returned optimization step is added to the current parameter vector to obtain the next parameter vector. Note that implementations of ES including our own usually follow the approach of Salimans et al. [47] and rank-normalize $E(\theta + \sigma\epsilon_i)$ before taking the weighted sum, which is a variance reduction technique. Overall, ES has exhibited performance on par with some of the traditional, simple gradient-based RL algorithms on difficult RL domains (e.g. DQN [7] and A3C [60]), including Atari environments and simulated robot locomotion [47, 61]. More recently, NS and QD algorithms have been shown possible to hybridize with ES to further improve its performance on sparse or deceptive deep RL tasks, while retaining scalability [61], providing inspiration for its novel hybridization within an CMOEA- and MCC-like algorithm in this paper.

Algorithm 1 ES_STEP

1: **Input:** an agent denoted by its policy parameter vector θ, an environment $E(\cdot)$, learning rate α, noise standard deviation σ
2: Sample $\epsilon_1, \epsilon_2, \ldots, \epsilon_n \sim \mathcal{N}(0, I)$
3: Compute $E_i = E(\theta + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
4: **Return:** $\alpha \frac{1}{n\sigma} \sum_{i=1}^{n} E_i \epsilon_i$

3 The Paired Open-Ended Trailblazer (POET) Algorithm

POET is designed to facilitate an open-ended process of discovery within a single run. It maintains a (1) population of environments (for example, various obstacle courses) and (2) population of agents (for example, neural networks that control a robot to solve those courses), and (3) each environment is paired with an agent to form an *environment-agent pair*. POET in effect implements an ongoing divergent coevolutionary interaction among all its agents and environments in the spirit of MCC [27], but with the added goal of explicitly optimizing the behavior of each agent within its paired environment in the spirit of CMOEA [45, 46]. It also elaborates on the minimal criterion in MCC by aiming to maintain only those newly-generated environments that are not too hard *and* not too easy for the current population of agents. The result is a *trailblazer* algorithm, one that continually forges new paths to both increasing challenges and skills within a single run. The new challenges are embodied

the idea of POET
have to check MCC & CMOEA

by the new environments that are continually created, and the increasing skills are embodied by the neural network controllers attempting to solve each environment. Existing skills are harnessed both by optimizing agents paired with environments and by attempting to transfer current agent behaviors to new environments to identify promising stepping stones.

The fundamental algorithm of POET is simple: The idea is to maintain a list of active environment-agent pairs `EA_List` that begins with a single starting pair $(E^{\text{init}}(\cdot), \theta^{\text{init}})$, where E^{init} is a simple environment (e.g. an obstacle course of entirely flat ground) and θ^{init} is a randomly initialized weight vector (e.g. for a neural network). POET then has three main tasks that it performs at each iteration of its main loop:

1. generating new environments $E(\cdot)$ from those currently active,
2. optimizing paired agents within their respective environments, and
3. attempting to transfer current agents θ from one environment to another.

Generating new environments is how POET continues to produce new challenges. To generate a new environment, POET simply mutates (i.e. randomly perturbs) the encoding (i.e. the parameter vector) of an active environment. However, while it is easy to generate perturbations of existing environments, the delicate part is to ensure both that (1) the paired agents in the originating (parent) environments have exhibited sufficient progress to suggest that reproducing their respective environments would not be a waste of effort, and (2) when new environments are generated, they are not added to the current population of environments unless they are neither too hard nor too easy for the current population. Furthermore,[3] priority is given to those candidate environments that are most novel, which produces a force for diversification that encourages many different kinds of problems to be solved in a single run.

These checks together ensure that the curriculum that emerges from adding new environments is smooth and calibrated to the learning agents. In this way, when new environments do make it into the active population, they are genuinely stepping stones for continued progress and divergence. The population of active environments is capped at a maximum size, and when the size of the population exceeds that threshold, the oldest environments are removed to make room (as in a queue). That way, environments do not disappear until absolutely necessary, giving their paired agents time to optimize and allowing skills learned in them to transfer to other environments.

POET optimizes its paired agents at each iteration of the main loop. The idea is that every agent in POET should be continually improving within its paired environment. In the experiments in this paper, each such iteration is a step of ES, but any reinforcement learning algorithm could conceivably apply. The objective in the optimization step is simply to maximize whatever performance measure applies to the environment (e.g. to walk as far as possible through an obstacle course). The fact that each agent-environment pair is being optimized independently affords easy parallelization, wherein all the optimization steps can in principle be executed at the same time.

Finally, attempting *transfer* is the ingredient that facilitates serendipitous cross-pollination: it is always possible that progress in one environment could end up helping in another. For example, if the paired agent θ^A in environment $E^A(\cdot)$ is stuck in a local optimum, one remedy could be a transfer from the paired agent θ^B in environment $E^B(\cdot)$. If the skills learned in the latter environment apply, it could revolutionize the behavior in the former, reflecting the fact that the most promising stepping stone to the best possible outcome may not be the current top performer in that environment [42, 44, 62]. Therefore, POET continually attempts transfers among the active environments. These transfer attempts are also easily parallelized because they too can be attempted independently.

More formally, Algorithm 2 describes this complete main loop of POET. Each component is shown in pseudocode: creating new environments, optimizing each agent, and attempting transfers. The specific implementation details for the subroutines MUTATE_ENVS (where the new environments are created) and EVALUATE_CANDIDATES (where transfers are attempted) are given in Supplemental Information.

As noted above, the independence of many of the operations in POET, such as optimizing individual agents within their paired environments and attempting transfers, makes it feasible to harness the power of many processors in parallel. In the implementation of the experiment reported here, each run harnessed 256 parallel CPU Cores. Our software implementation of POET, which will be released as open source code shortly, allows seamless parallelization over any number of cores: workers are

Algorithm 2 POET Main Loop

Input: initial environment $E^{\text{init}}(\cdot)$, its paired agent denoted by policy parameter vector θ^{init}, learning rate α, noise standard deviation σ, iterations T, mutation interval N^{mutate}, transfer interval N^{transfer}
Initialize: Set `EA_list` empty
Add $(E^{\text{init}}(\cdot), \theta^{\text{init}})$ to `EA_list`
for $t = 0$ **to** $T - 1$ **do**
  if $t > 0$ **and** $t \mod N^{\text{mutate}} = 0$ **then**
    `EA_list` = MUTATE_ENVS(`EA_list`) **# new environments created by mutation**
  end if
  $M = \text{len}(\texttt{EA\_list})$
  for $m = 1$ **to** M **do**
    $E^m(\cdot), \theta^m_t$ = `EA_list`[m]
    $\theta^m_{t+1} = \theta^m_t +$ ES_STEP$(\theta^m_t, E^m(\cdot), \alpha, \sigma)$ **# each agent independently optimized**
  end for
  for $m = 1$ **to** M **do**
    if $M > 1$ **and** $t \mod N^{\text{transfer}} = 0$ **then**
      $\theta^{\text{top}} =$ EVALUATE_CANDIDATES$(\theta^1_{t+1}, \ldots, \theta^{m-1}_{t+1}, \theta^{m+1}_{t+1}, \ldots, \theta^M_{t+1}, E^m(\cdot), \alpha, \sigma)$
      if $E^m(\theta^{\text{top}}) > E^m(\theta^m_{t+1})$ **then**
        $\theta^m_{t+1} = \theta^{\text{top}}$ **# transfer attempts**
      end if
    end if
    `EA_list`$[m] = (E^m(\cdot), \theta^m_{t+1})$
  end for
end for

managed through Ipyparallel to automatically distribute all current tasks so that requests for specific operations and learning steps can be made without concern for how they will be distributed.

Provided that a space of possible environmental challenges can be encoded, the hope is that the POET algorithm can then start simply and push outward in parallel along an increasingly challenging frontier of challenges, some of which benefit from the solutions to others. The next section describes a platform designed to test this capability.

4 Experiment Setup And Results

An effective test of POET should address the hypothesis that it can yield an increasingly challenging set of environments, many with a satisfying solution, all in a single run. Furthermore, we hope to see evidence for the benefit of cross-environment transfers. The experimental domain in this work is a modified version of the "Bipedal Walker Hardcore" environment of the OpenAI Gym [63]. Its simplicity as a 2-D walking domain with various kinds of possible terrain makes it easy to observe and understand qualitatively different ambulation strategies simply by viewing them. Furthermore, the environments are easily modified, enabling numerous diverse obstacle courses to emerge to showcase the possibilities for adaptive specialization and generalization. Finally, it is relatively fast to simulate, allowing many long experiments compared to the more complex environments where we expect this algorithm to shine in the future.

4.1 Environment and Experiment Setup

The agent's hull is supported by two legs (the agent appears on the left edge of figure 1). The hips and knees of each leg are controlled by two motor joints, creating an action space of four dimensions. The agent has ten LIDAR rangefinders for perceiving obstacles and terrain, whose measurements are included in the state space. Another 14 state variables include hull angle, hull angular velocity, horizontal and vertical speeds, positions of joints and joint angular velocities, and whether legs touch the ground [63].

action space : 4
state space : 24

Guided by its sense of the outside world through LIDAR and its internal sensors, the agent is required to navigate, within a time limit and without falling over, across an environment of a generated terrain

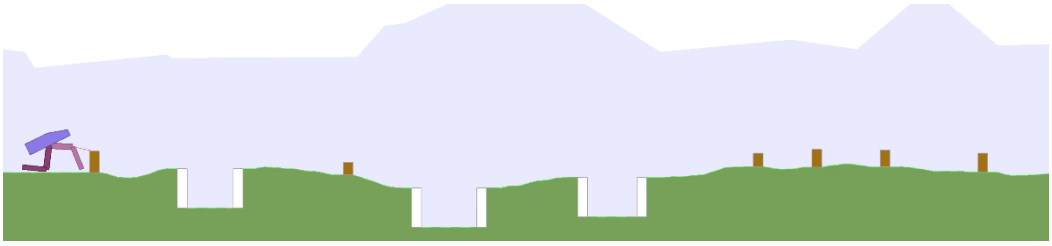


Figure 1: **A landscape from the Bipedal Walker environment**. Possible obstacles include stumps, gaps, stairs, and surfaces with different amounts of roughness.

that consists of one or more types of obstacles. These can include stumps, gaps, and stairs on a surface with a variable degree of roughness, as illustrated in Figure 1. As described in the following equation, reward is given for moving forward, with almost no constraints on agents' behaviors other than that they are encouraged to keep their hulls straight and minimize motor torque. If the agent falls, the reward is -100.

$$\text{Reward per step} = \begin{cases} -100, & \text{if robot falls} \\ 130 \times \Delta x - 5 \times \Delta\text{hull\_angle} - 0.00035 \times \text{applied\_torque}, & \text{otherwise.} \end{cases}$$

The episode immediately terminates when the time limit (2,000 time steps) is reached, when the agent falls, or when it completes the course. In this work, we define an environment as *solved* when it both reaches the far end of the environment and obtains a score of 230 or above. Based on our observations, the score of 230 ensures that the walker is reasonably efficient.

Following the architecture of controllers for the same domain by Ha [64], all controllers in the experiments are implemented as neural networks with 3 fully-connected layers with $tanh$ activation functions. The controller has 24 inputs and 4 outputs, all bounded between -1 and 1, with 2 hidden layers of 40 units each. Each ES step (Algorithm 1) has a population size of 512 (i.e. the number of parameters sampled to create the weighted-average for one gradient step is 512) and and updates weights through the Adam optimizer [65]. The learning rate is initially set to 0.01, and decays to 0.001 with a decay factor of 0.9999 per step. The noise standard deviation for ES is initially set to 0.1, and decays to 0.01 with a decay factor of 0.999 per step. When any environment-agent pair accepts a transfer or when a child environment-agent pair is first created, we reset the state of its Adam optimizer, and set the learning rate and noise standard deviation to their initial values, respectively.

4.2 Environment Encoding and Mutation

Intuitively, the system should begin with a single flat and featureless environment whose paired policy can be optimized easily (to walk on flat ground). From there, new environments will continue to be generated from their predecessors, while their paired policies are simultaneously optimized. The hope is that a wide variety of control strategies and skill sets will allow the completion of an ever-expanding set of increasingly complex obstacle courses, all in a single run.

To enable such a progression, we adopt a simple encoding to represent the search space of possible environments. As enumerated in Table 1, there are five types of obstacles that can be distributed throughout the environment: (1) stump height, (2) gap width, (3) step height, (4) step number (i.e. number of stairs), and (5) surface roughness. Three of these obstacles, e.g. stump height, are encoded as a pair of parameters (or *genes*) that form an interval from which the actual value for each instance of that type of obstacle in a given environment is uniformly sampled. As will be described below, in some experiments, some obstacle types are intentionally omitted, allowing us to restrict the experiment to certain types of obstacles. When selected to mutate for the first time, obstacle parameters are initialized to the corresponding initial values shown in Table 1. For subsequent mutations, an obstacle parameter takes a *mutation step* (whose magnitude is given in Table 1), and either adds or subtracts the step value from its current value. Note that for surface roughness, both the initial value and every subsequent mutation step are sampled uniformly from $(0, 0.6)$ because the impact of roughness on the agent was found to be significant sometimes even from very small differences. The value of any given parameter cannot exceed its *maximum value*.

Because the parameters of an environment define a distribution, the actual environment sampled from that distribution is the result of a random *seed*. This seed value is stored with each environment so that environments can be be reproduced precisely, ensuring repeatability. (The population of many environments and the mutation of environments over time still means that training overall does not occur on only one deterministic environment.) With this encoding, all possible environments can be uniquely defined by the values for each obstacle type in addition to the seed that is kept with the environment.

Any environments that satisfy the reproduction eligibility condition in Line 7 of Algorithm 3 (which is given in Supplemental Information) are allowed to mutate to generate a child environment. In our experiments, this condition is that the paired agent of the environment achieves a reward of 200 or above, which generally indicates that the agent is capable of reaching the end of the terrain (though slightly below the full success criterion of 230). To construct `child_list` in Line 15 in Algorithm 3, the set of eligible parents is sampled uniformly to choose a parent, which is then mutated to form a child that is added to the list. This process is repeated until `child_list` reaches `max_children`, which is 512 in our experiments. Each child is generated from its parent by independently picking and mutating some or all the available parameters of the parent environment and then choosing a new random seed. The *minimal criterion* (MC) $50 < E^{\text{child}}(\theta^{\text{child}}) < 300$ then filters out child environments that appear too challenging or too trivial for the current level of capability of agents. In case the number of child environments that satisfy the MC is more than the maximum number of children admitted per reproduction, those with lower $E^{\text{child}}(\theta^{\text{child}})$ are admitted until the cap is reached.

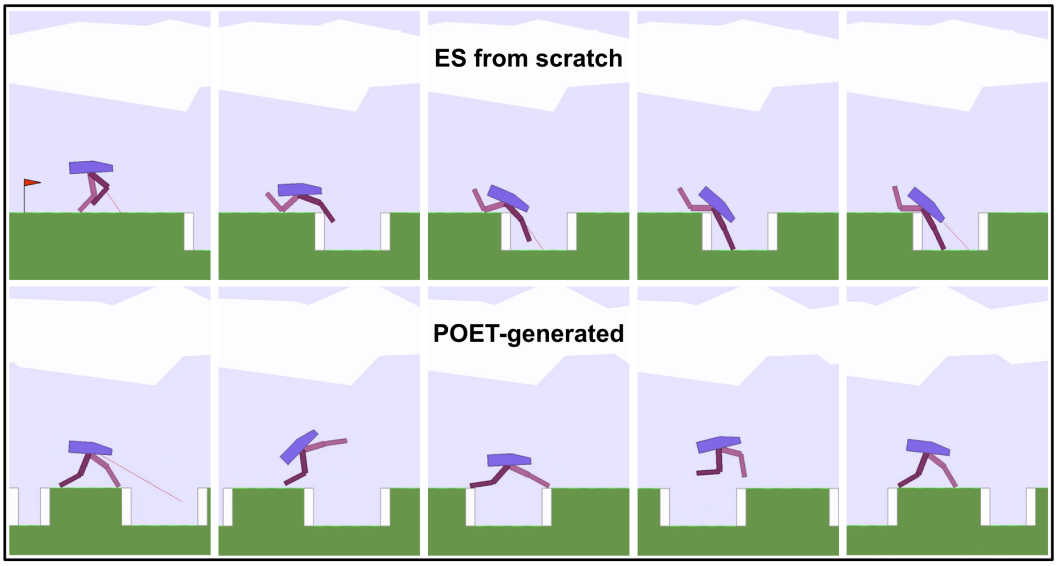
Obstacle Type	Stump Height	Gap Width	Step Height	Step Number	Roughness
Initial Value	$(0.0, 0.4)$	$(0.0, 0.8)$	$(0.0, 0.4)$	1	uniform$(0, 0.6)$
Mutation Step	0.2	0.4	0.2	1	uniform$(0, 0.6)$
Max Value	$(5.0, 5.0)$	$(10.0, 10.0)$	$(5.0, 5.0)$	9	10.0

Table 1: **Environmental parameters (genes) for the Bipedal Walker problem.**

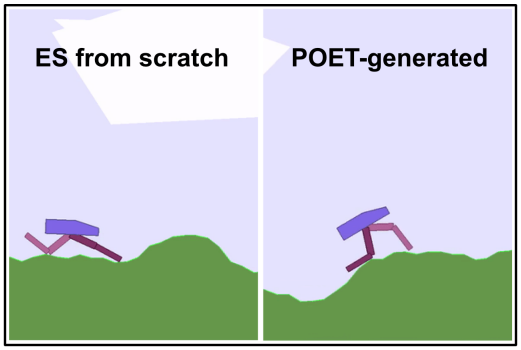
4.3 Results

An important motivating hypothesis for POET is that the stepping stones that lead to solutions to very challenging environments are more likely to be found through a divergent, open-ended process than through a direct attempt to optimize in the challenging environment. Indeed, if we take a particular environment evolved in POET and attempt to run ES (the same optimization algorithm used in POET) on that specific environment from scratch, the result is often premature convergence to degenerate behavior. The first set of experiments focus on this phenomenon by running POET with only *one* obstacle type enabled (e.g. gaps in one case, roughness in another, and stumps in another). That way, we can see that even with a single obstacle type, the challenges generated by POET (which POET solves) are too much for ES on its own.

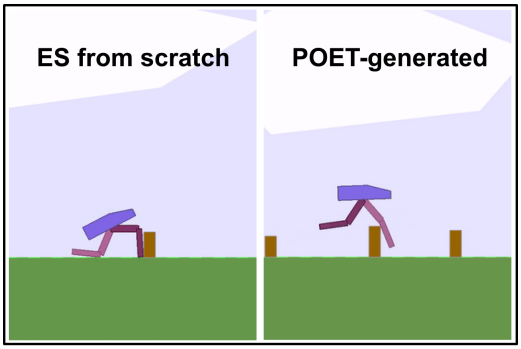
Figure 2 shows example results that illustrate this principle. Three challenging environments that POET created are shown, with wide gaps, rugged hillsides, and high stumps, respectively. To demonstrate that these environments are challenging, we ran ES to directly optimize randomly initialized agents for them. More specifically, for each of the three environments, we ran ES five times with different initialization and random seeds up to 16,000 ES steps (which is twice as long as Ha [64] gives agents in the same domain with ES). Such ES-optimized agents are consistently stuck at local minima in these environment. The maximum scores out of the five ES runs for the environments illustrated in Figure 2a, 2b, and 2c are 17.9, 39.6, and 13.6, respectively, far below the success threshold of 230 that POET exceeds in each case. In effect, to obtain positive scores, these agents learn to move forward, but also to freeze before challenging obstacles, which helps them avoid the penalty of -100 for falling. This behavior is a local optimum: the agents could in principle learn to overcome the obstacles, but instead converge on playing it safe by not moving. Note that ES had previously been shown to be a competent approach [66] on the original Bipedal Walker Hardcore environment in OpenAI Gym, which, as illustrated later in Table 2, consists of similar, but much less difficult obstacles. The implication is that the challenges generated and solved by POET are significantly harder and ES alone is unable to solve them.



(a) ES-/POET-generated agents attempting gaps



(b) ES-/POET-generated agents on rough surfaces

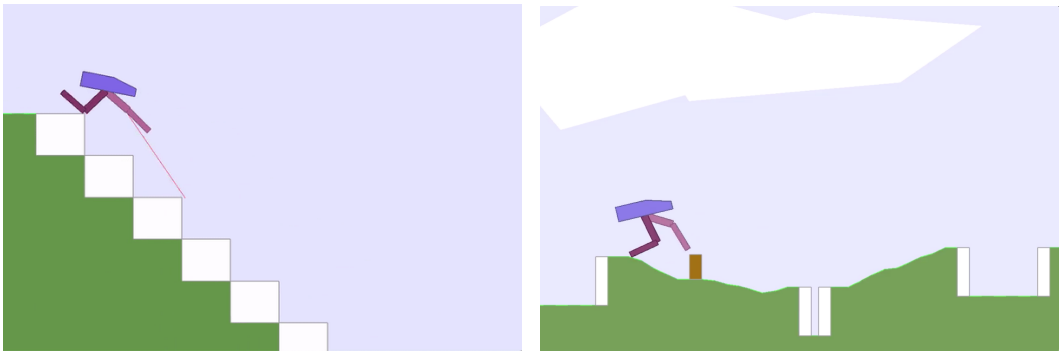


(c) ES-/POET-generated agents attempting stumps

Figure 2: **POET creates challenging environments and corresponding solutions that cannot be obtained by optimizing randomly initialized agents by ES.** As illustrated in the top row of (a) and the left panels of (b) and (c), agents directly optimized by ES converge on degenerate behaviors that give up early in the course. In contrast, POET not only creates these challenging environments, but also learns agents that overcome the obstacles to navigate effectively, as shown in the bottom row of (a) and right panels of (b) and (c).

For example, in Figure 2a, the sequence in the top row illustrates the agent choosing to stop to avoid jumping over a dangerously wide gap: it slowly sticks one of its feet out to reach the bottom of the first wide gap and then maintains its balance without any further movement until the time limit of the episode is reached. Videos of this and other agents from this paper can be found at `eng.uber.com/poet-open-ended-deep-learning`. In contrast, POET not only *creates* such challenging environments, but also in this case learns a clever behavior to overcome wide gaps and reach the finish line (second row of Figure 2a).

Figure 3 illustrates two more interesting examples of challenging environments and the highly adapted capabilities of their paired agents. In the first, the agent navigates a very steep staircase of oversized steps, which ES alone cannot solve. In the second, an experiment in heterogeneous environments (with both gaps and stumps available) yields a single obstacle course of varying gap widths and stump heights where the POET agent succeeds and ES fails. The maximum scores out of the five ES runs for the environments illustrated in Figure 3a and 3b are 24.0 and 19.2, respectively, again far below the success theshold for POET of 230. Figure 4 illustrates the dramatic failure of ES to match the performance of POET in these relatively simple environments that POET generated and solved.



(a) A downstairs course with a very large step height (b) A rough walking course with gaps and stumps

Figure 3: **Agents exhibit specialized skills in challenging environments created by POET**. These include the agent navigating (a) very large steps and (b) a rough walking course with mixed gaps of small and large widths, and stumps of various heights.

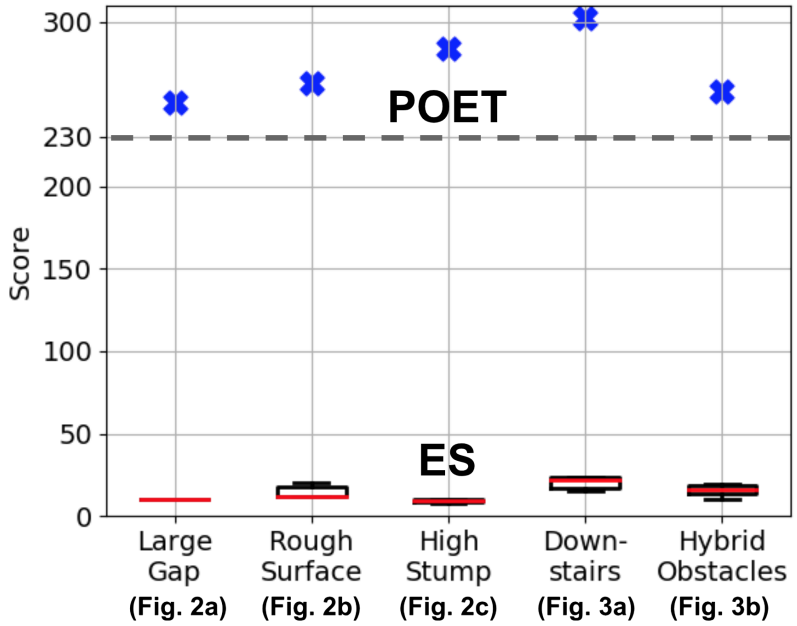


Figure 4: **ES on its own cannot reproduce results achieved by POET.** Each column compares a box plot representing five ES runs to the performance achieved by POET in the single-obstacle-type problems (except for the last column) generated by POET. Recall that a score of 230 (the dotted line) is the threshold for success. Though they are hard to see in these plots because of the narrow ES distributions, the red lines are medians, which are surrounded by a box from the first quartile to the third, which are further surrounded by lines touching the minimum and maximum. As the plot illustrates, ES on its own definitively fails in every case to come close to the challenges set and solved by POET.

An interesting aspect of open-ended algorithms is that statistical comparisons can be challenging because such algorithms are not trying to perform well on a specific preconceived target, and thus some amount of post-hoc analysis is often necessary. That property is what motivates an analysis driven by choosing environments produced and solved by POET, and then checking whether direct optimization can solve these environments. However, from a statistical comparison standpoint it raises the challenge that because each environment created by POET is unique, we only have one score for POET to compare against a set of direct-optimization ES runs. For that reason, we use a *single-sample t-test* that assumes the POET distribution is centered on the single POET score and measures whether the direct optimization distribution's mean is statistically significantly lower. As the very poor results for ES would suggest, based on single-sample t-tests for all five environments in figures 2 and 3 that are created by POET, the scores of agents optimized by ES alone are very unlikely to be from a distribution near the POET-level solutions ($p < 0.01$).

have to check

One interpretation of POET's ability to create agents that can solve challenging problems is that it is in effect an automatic curriculum builder. Building a proper curriculum is critical for learning to master tasks that are challenging to learn from scratch due to a lack of informative gradient. However, building an effective curriculum given a target task is itself often a major challenge. Because newer environments in POET are created through mutations of older environments and because POET only accepts new environments that are not too easy not too hard for current agents, POET implicitly builds a curriculum for learning each environment it creates. The overall effect is that it is building many overlapping curricula simultaneously, and continually checking whether skills learned in one branch might transfer to another.

A natural question then is whether the environments created and solved by POET can also be solved by an explicit, *direct-path curriculum-building control* algorithm. To test this approach, we first collect a sample of environments generated and solved by POET, and then apply the direct-path control to each one separately to see if it can reach the same capabilities on its own. In this control, the agent is progressively trained on a sequence of environments of increasing difficulty that move towards the target environment. This kind of incremental curriculum is intuitive and variants of it appear in the literature when a task is too hard to learn directly [40, 67–70]. The sequence of environments start with an environment of only flat ground without any obstacles (which is easy enough for any randomly initialized agent to quickly learn to complete). Then each of the subsequent environments are constructed by slightly modifying the current environment. More specifically, to get a new environment, each obstacle parameter of the current environment has an equal chance of staying the same value or *increasing* by the corresponding mutation step value in Table 1 until that obstacle parameter reaches that of the target environment.

have to check

In this direct-path curriculum-building control, the agent moves from its current environment to the next one when the agent's score in the current environment reaches the reproduction eligibility threshold for POET, i.e. the same condition for when an environment reproduces in POET. The control algorithm optimizes the agent with ES (just as in POET). It stops when the target environment is reached and solved, or when a computational budget is exhausted. To be fair, each run of the control algorithm is given the same computational budget (measured in total number of ES steps) spent by POET to solve the environment, which includes all the ES steps taken in the entire sequence of environments along the direct line of ancestors (taking into account transfers) leading to the target. This experiment is interesting because if the direct-path control cannot reach the same level as the targets, it means that a direct-path curriculum alone is not sufficient to produce the behaviors discovered by POET, supporting the importance of multiple simultaneous paths and transfers between them.

The direct-path curriculum-building control is tested against a set of environments generated and solved by POET that encompass a range of difficulties. For these experiments, the environments have three obstacle types enabled: gaps, roughness, and stumps. Unlike when comparing to ES alone, he we allow POET to generate environments with multiple obstacles at the same time because the curriculum-based approach should in principle be more powerful than ES alone. To provide a principled framework for choosing the set of generated environments to analyze, they are classified into three difficulty levels. The difficulty level of an environment is based on how many conditions it satisfies out of the three listed in Table 2. In particular, a *challenging* environment satisfies one of the three conditions; a *very challenging* environment satisfies two of the three; and satisfying all three makes an environment *extremely challenging*. It is important to note that these conditions all merit the word "challenging" because they all are much more demanding than the corresponding values

used in the original Hardcore version of Bipedal Walker in OpenAI Gym [71] (denoted as *reference* in Table 2, which was also used by Ha [64]. More specifically, they are 1.2, 2.0, and 4.5 times the corresponding reference values from the original OpenAI Gym environment, respectively.

	Top Value in range of Stump Height	Top Value in range of Gap Width	Roughness
This Work	≥ 2.4	≥ 6.0	≥ 4.5
Reference	2.0	3.0	1.0

Table 2: **Difficulty level criteria.** The difficulty level of an environment is based on how many conditions it satisfies out of the three listed here. The *reference* in the second row shows the corresponding top-of-range values used in the original Hardcore version of Bipedal Walker in OpenAI Gym [71]. The difficulty of the environments produced and solved by POET are much higher than the reference values.

In this experiment, each of three runs of POET takes up to 25,200 POET iterations with a population size of 20 active environments, while the number of sample points for each ES step is 512. These runs each take about 10 days to complete on 256 CPU cores. The mean (with 95% confidence intervals) POET iterations spent on solving challenging, very challenging, and extremely challenging environments starting from the iteration when they were first created are 638 ± 133, $1{,}180 \pm 343$, and $2{,}178 \pm 368$, respectively. Here, one POET iteration refers to creating new environments, optimizing current paired agents, and attempting transfers (i.e. lines 4-22 in Algorithm 2). The more challenging environments clearly take more effort to solve.

Figure 5 compares the POET environments and the direct-path curriculum-building control algorithm through a series of *rose plots*, wherein each rose plot compares the configuration of an environment solved by POET (red pentagons) with the closest that the direct-path control could come to that configuration (blue pentagons). For each red pentagon there are five such blue pentagons, each representing one of five separate attempts by the control to achieve the red pentagon target. The five vertices of each pentagon indicate roughness, the lower and upper bounds of the range of the gap width, and those of the stump height, respectively. Each column in figure 5 consists of six representative samples (the red pentagons) of environments that a single run of POET up to 25,200 iterations created and solved. As the rows descend from top to bottom, the difficulty level decreases. In particular, the top two rows are extremely challenging targets, the next two are very challenging, and the bottom two are challenging (all are randomly sampled from targets generated and solved at each difficulty level in each run). Qualitatively, it is clear from figure 5 that attempts by the direct-path control consistently fail to reach the same level as POET-generated (and solved) levels.

To more precisely quantify these results, we define the normalized distance between any two environments, E_A and E_B as $\frac{1}{\beta}||\frac{e(E_A)-e(E_B)}{e(E_{\text{Max}})}||_2$, where $e(E)$ is the genetic encoding vector of E, and E_{Max} is a hypothetical environment (for normalization purpose) with all genetic encoding values are maxed out. (The maximum values are roughness $= 8$, Gap_lower $=$ Gap_upper $= 8$, Stump_lower $=$ Stump_upper $= 3$.) The constant $\beta = \sqrt{5}$ is simply for normalization so that the distance is normalized to 1 when $E_A = E_{\text{Max}}$ and E_B is a purely flat environment (roughness zero) without any obstacles, which therefore contains an all-zero genetic encoding vector. Based on this distance measure, we can calculate the median values and confidence intervals of distances between target environments (created and solved by POET) and the corresponding closest-to-target environments that the control algorithms can solve. These values are calculated for all the environments of different challenge levels across all three runs shown in figure 5.

The results, summarized in figure 6, demonstrate that POET creates and solves environments that the control algorithm fails to solve at very and extremely challenging difficulty levels, while the curriculum-based control algorithm can often (though not always) solve environments at the lowest challenge level. One implication of these results is that the direct-path curriculum-building control is valid in the sense that it does perform reasonably well at solving minimally challenging scenarios. However, the very and extremely challenging environments that POET invents reach significantly beyond what the direct-path curriculum can match. There are a couple ways to characterize this significance level more formally. These analyses are based on the *distance between the closest environment reached by the control and the POET-generated target*. First, we can examine whether

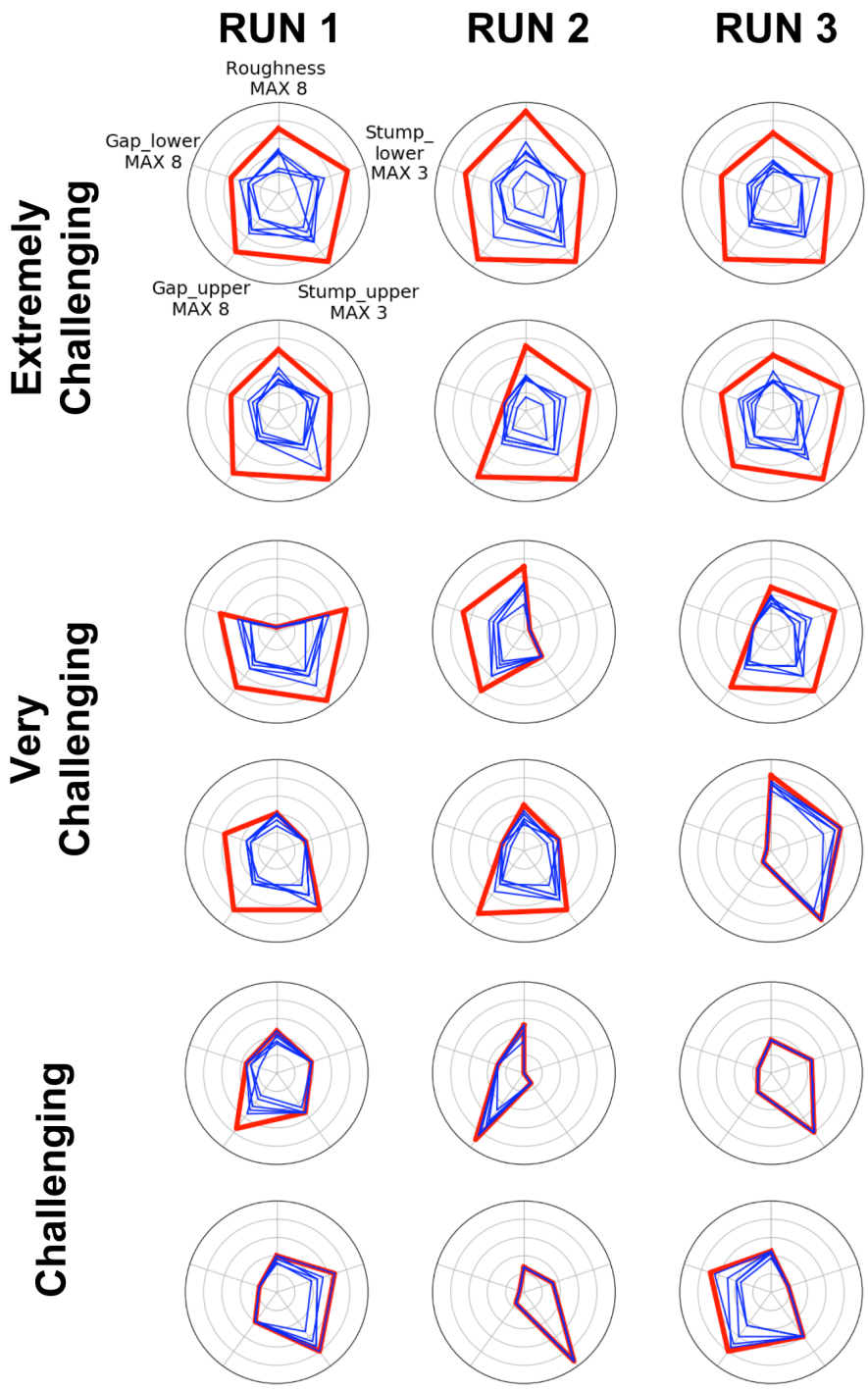


Figure 5: **POET versus direct-path curriculum-building controls.** Each rose plot depicts one environment that POET created and solved (red pentagon). For each, the five blue pentagons indicate what happens in control runs when the red pentagon is the target. Each blue pentagon is the closest-to-target environment solved by one of the five independent runs of the control algorithm. The five vertices of each pentagon indicate roughness (*Roughness*), the bottom and top values of the range of the gap width of all the gaps (*Gap_lower* and *Gap_upper*), and the bottom and top values for the height of stumps (*Stump_lower* and *Stump_upper*) in the given solved environment. The value after MAX in the key is the maximum value at the outermost circle for each type of obstacle. Each column contains sample solved environments from a single independent run of POET.

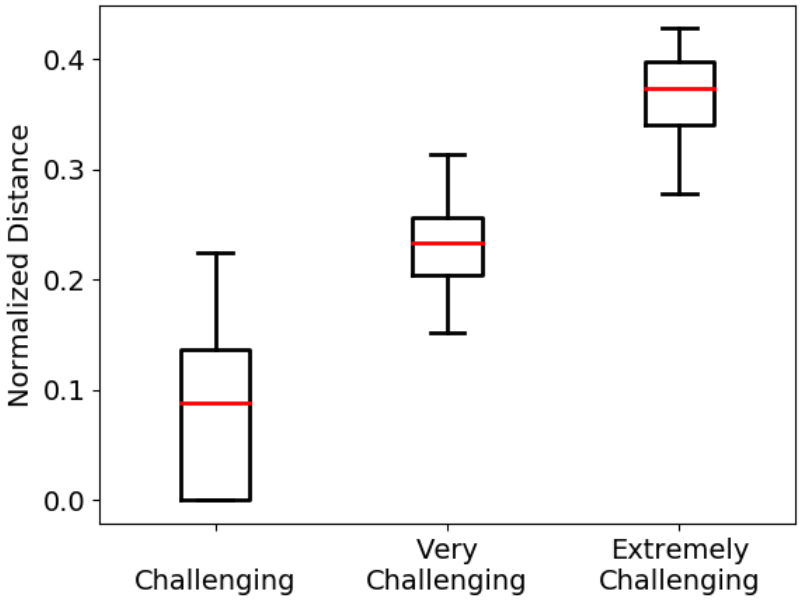
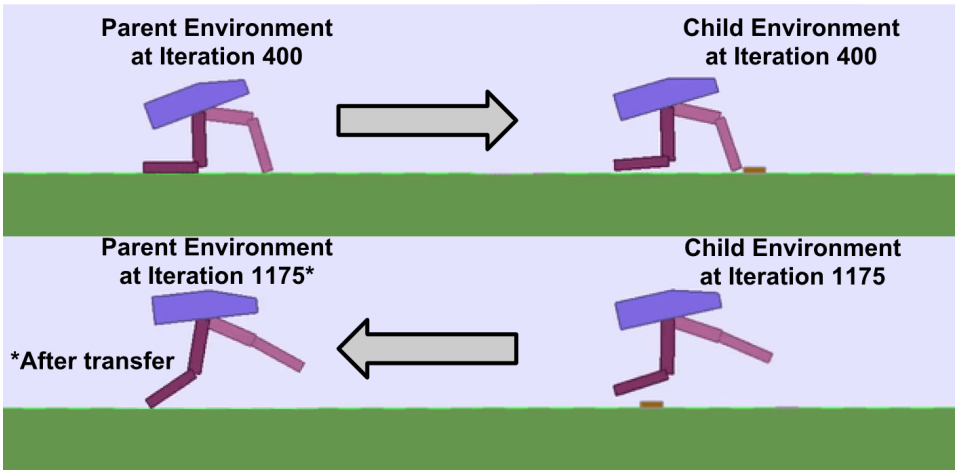


Figure 6: **Distances between targets and results of direct-path curriculum-building control runs.** Here, levels *challenging*, *very challenging*, and *extremely challenging* correspond to the bottom two, middle two and top two rows of red pentagons in Figure 5, respectively. The red lines are medians, which are surrounded by a box from the first quartile to the third, which are further surrounded by lines touching the minimum and maximum. These results highlight how difficult it becomes for the direct-path control to match challenges invented and solved by POET as the challenge level increases.

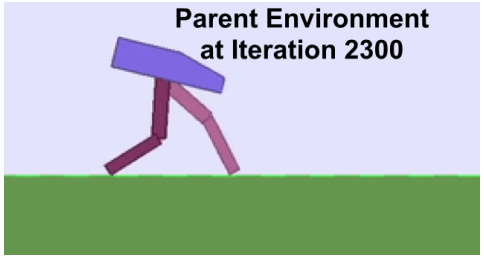
that distance is significantly greater the higher the challenge level. Indeed, Mann-Whitney U tests show that the difference between such distances between challenging and very challenging environments is indeed very significant ($p < 0.01$), as is the difference between such distances between very challenging and extremely challenging. This test gives a quantitative confirmation of the intuition (also supported by figure 6) that these challenge levels are indeed meaningful, and the higher they go, the more out of reach they become for the control. Second, a single-sample t-test (which was previously used when comparing POET to the ES-alone control) can also here provide confidence that the distances from the targets are indeed far in an absolute sense: indeed, even at the lowest challenge level (and for all challenge levels), the one-sample t-test measures high significance ($p < 0.01$) between the distribution of distances from the target and the target level.

In aggregate, the results from the direct-path curriculum-building control help to show quantitatively the advantage of POET over conventional curriculum-building. In effect, the ability to follow multiple chains of environments in the same run and transfer skills among them pushes the frontier of skills farther than a single-chain curriculum can push, hinting at the limitations of preconceived curricula in general. Thus, while POET cannot guarantee reaching a particular preconceived target, these results suggest that in some environment spaces POET's unique ability to generate multiple different challenges and solve them may actually still provide a more promising path towards solving some preconceived target challenges (i.e. offering a powerful alternative to more conventional optimization), putting aside the additional interesting benefits of an algorithm that invents, explores, and solves new types of challenges automatically.

A fundamental problem of a pre-conceived direct-path curriculum (like the control algorithm above) is the potential lack of necessary *stepping stones*. In particular, skills learned in one environment can be useful and critical for learning in another environment. Because there is no way to predict where and when stepping stones emerge, the need arises to conduct transfer experiments (which POET implements) from differing environments or problems [44]. In conventional curriculum-building



(a) Transfer from agent in parent environment to child environment and vice versa



(b) The walking gait of agent in parent environment at Iteration 2,300

Figure 7: **Synergistic two-way transfer between parent and child environments.** At iteration 400, a transfer from parent environment yields a child agent now learning in a stumpy environment, shown in the top row of (a). The child agent eventually learns to stand up and jump over the stumps, and at iteration 1,175 that skill is transferred *back* to the parent environment, depicted in the bottom row of (a). This transfer from the child environment back to the parent helps the parent agent learn a more optimal walking gait compared to its original one. Given the same amount of computation, the agent with the original walking gaits reaches a score of 309, while the one with the more optimal walking gait as illustrated in (b) reaches a score of 349.

algorithms, the "transfer" only happens once, i.e. at the time when the new environment is created, and is unidirectional, i.e. from the agent paired with the parent environment (*parent agent*) to the *child environment* with the hope that the parent agent helps jump-start learning in the child environment.

In contrast, POET maintains parallel paths of environment evolution and conducts transfer experiments periodically that give every active agent multiple chances to transfer its learned skills to others, including the transfer of a child agent back to its parent environment, which continually creates and preserves opportunities to harness unanticipated stepping stones. Figure 7 presents an interesting example from POET where such transfers help a parent environment acquire a better solution: The parent environment includes only flat ground and the parent agent learns to move forward without fully standing up (top left), which works in this simple environment, but represents a local optimum in walking behavior because standing would be better. At iteration 400, the parent environment mutates and generates a child environment with small stumps, and the child agent inherits the walking gait from the parent agent.

At first, it can move forward in the new stumpy environment, but it often stumbles because of its crouching posture (top right). Later, through several hundred further iterations of ES, the child agent eventually learns to stand up and jump over the stumps. The interesting moment is iteration 1,175, when that skill is transferred back to the *parent* environment (middle row). For the first time, the parent environment (which is completely flat) now contains an agent who stands up while walking. Without the backward transfer, the agent in the simple flat environment would be stuck at its original

gait (which is in effect a local optimum) forever, which was confirmed by separately turning off transfer and running the kneeling (local optimum) agent in the original (parent) environment for 3,000 iterations, which did not result in any substantive change. Interestingly, after transfer back to the parent environment, the transferred agent then continued to optimize the standing gait to the parent (stump-free) environment and eventually obtained a much better walking gait that moves faster and costs less energy due to less friction (bottom row). More specifically, given 3,000 iterations, the agent with the original kneeling walking gait reaches a score of 309, while the one with the more optimal walking gait illustrated in figure 7b reaches a score of 349.

To give a holistic view of the success of transfer throughout the entire system, we count the number of *replacement attempts* during the course of a run, which means the number of times an environment took a group of incoming transfer attempts from all the other active environments. The total number of such replacement attempts in RUN 1, RUN 2, and RUN 3 (labelled in Figure 5) are 18,894, 19,014, and 18,798, respectively, out of which, 53.62%, 49.26%, 48.89%, respectively, are successful replacements (meaning one of the tested agents is better than the current niche champion). Note that each replacement attempt here encompasses both the direct transfer and proposal transfer attempts. These statistics show how pervasively transfer permeates (and often adds value to) the parallel paths explored by POET.

While transfer is pervasive, that does not in itself prove it is essential. To demonstrate the value of transfer, a control is needed: We relaunched another three POET runs, but with all the transfers *disabled* (which we call *POET without transfer*). In this variant, POET runs as usual, but simply never tries to transfer paired solutions from one environment to another. We can then calculate the *coverage* of the environments that are created and solved by POET and the control, respectively, following a similar metric as defined in Lehman and Stanley [41]: We first uniformly sample 1,000 *challenging*, 1,000 *very challenging*, and 1,000 *extremely challenging* environments. For each of the total 3,000 sampled environments, the distance to the nearest one of the *challenging*, *very challenging*, or *extremely challenging* environments created and solved by POET (and by the control, respectively) is calculated. Note that the better covered the environment space is, the lower the sum of all such nearest distances will be. The result is that the coverage of environments created and solved by POET is significantly greater than by POET without transfer ($p < 2.2e{-}16$ based on Mann-Whitney U test), meaning POET with transfer explores more of the space of environments, including creating (and solving) more difficult environments. In an even more dramatic statistic showing the essential role of transfer in open-ended search, in POET without transfer, *no extremely challenging environments are solved at all (Fig. 8).*

Finally, one of the primary hypothesized benefits of POET is its ability to produce a broad diversity of different problems with functional solutions in a single run. The environments (depicted as red pentagons) shown in each column of Figure 5 are created and solved in a *single* run of POET. Each such column exhibits diversity in the values and/or value ranges in roughness, gap width of gaps, and height of stumps. Take the environments in the middle column (labelled "RUN 2") for example. In figure 9, each image of a section of an environment from top to bottom corresponds to the same row of rose plot for RUN 2 in Figure 5. For instance, the topmost environment has narrow ranges and high values in gap width and stump height, and high value in roughness; in stark contrast the plot at the bottom depicts a less challenging environment with a wide range of stump heights, but low gap width and roughness. This diversity of environments also implies diverse experiences for the agents paired with them, who in turn thereby learn diverse walking gaits. This diversity then supplies the stepping stones that fuel the mutual transfer mechanism of POET. For a more objective statement of POET's ability to cover much of the possibility space through its diversity, all three POET runs created and solved sufficiently diverse environments to cover all three challenge levels.

5 Discussion, Future Work, and Conclusion

The promise of open-ended computation is fascinating for its potential to enable systems that become more powerful and interesting the longer they run. There is always the possibility that the feats we observe in the system today will be overshadowed by the achievements of tomorrow. In these unfolding odysseys there is an echo of the natural world. More than just the story of a single intelligent lifetime, they evoke the history of invention, or of natural evolution over the eons of Earth. These are processes that produce not just a single positive result, but an ongoing cacophony of surprises – unplanned advances rolling ahead in parallel without any final destination.

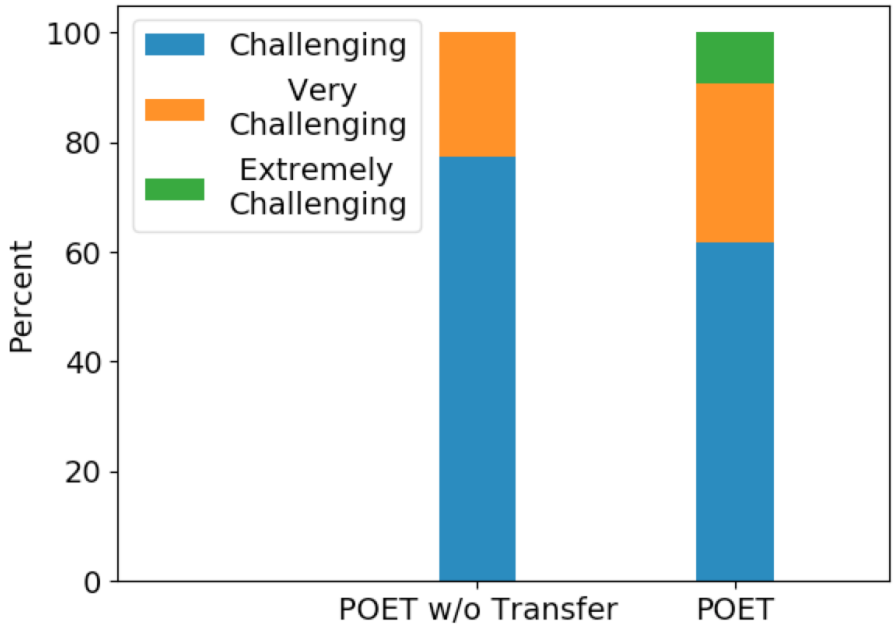


Figure 8: **Percentage of environments of different difficult levels created and solved by POET and the control runs of POET without transfer.** In the control without transfer, no *extremely challenging* environments are generated.

POET is an attempt to move further down the road towards these kinds of systems. While the road remains long, the rewards for machine learning of beginning to capture the character of open-ended processes is potentially high. First, as the results show, there is the opportunity to discover capabilities that could not be learned in any other way, even through a carefully crafted curriculum targeted at the desired result. In addition, a diversity of such results can be generated in a single run, and the problems and solutions can both increase in complexity over time. Furthermore, the implicit result is that POET is self-generating multiple curricula simultaneously, all while leveraging the results of some as stepping stones to progress in others.

The experiment in this article in 2-D walking establishes the potential of POET, but POET becomes more interesting the more unbounded its problem space becomes. The present problem space is a 2-D course of obstacles that can be generated within distributions defined by the genome describing the environment. This space is limited by the maximal ranges of those distributions. For example, there is a maximum possible gap width and stump height, which means that the system in effect can eventually "max out" the difficulty. While sufficiently broad to demonstrate POET's functionality, in the future much more flexible or even unbounded encodings can enable POET to traverse a far richer problem space. For example, an indirect encoding like a compositional pattern-producing network (CPPN) [72] can generate arbitrarily complex patterns that can be e.g. converted into levels or obstacle courses. Such an encoding would allow POET to diverge across a much richer landscape of possibilities.

An additional constraint that exists in this initial work is that the body of the agent is fixed, ultimately limiting the sort of obstacles it can overcome (e.g. how big of a gap it can jump over). Here too a more powerful and expressive encoding of morphologies, including those like CPPNs that are based on developmental biology, could allow us to co-evolve the morphology and body of the agent along with its brain in addition to the environment it is solving. Previous research has shown that using such indirect encodings to evolve morphologies can improve performance and create a wide diversity of interesting solutions [73]. Even within the 2-D Bipedal Walker domain from this paper, research has shown that allowing the morphology to be optimized can improve performance and provide a diversity of interesting solutions [64].

Furthermore, the CPPN indirect encoding can also encode the neural network controllers in an algorithm called HyperNEAT [74]. Work evolving robot controllers with HyperNEAT has shown

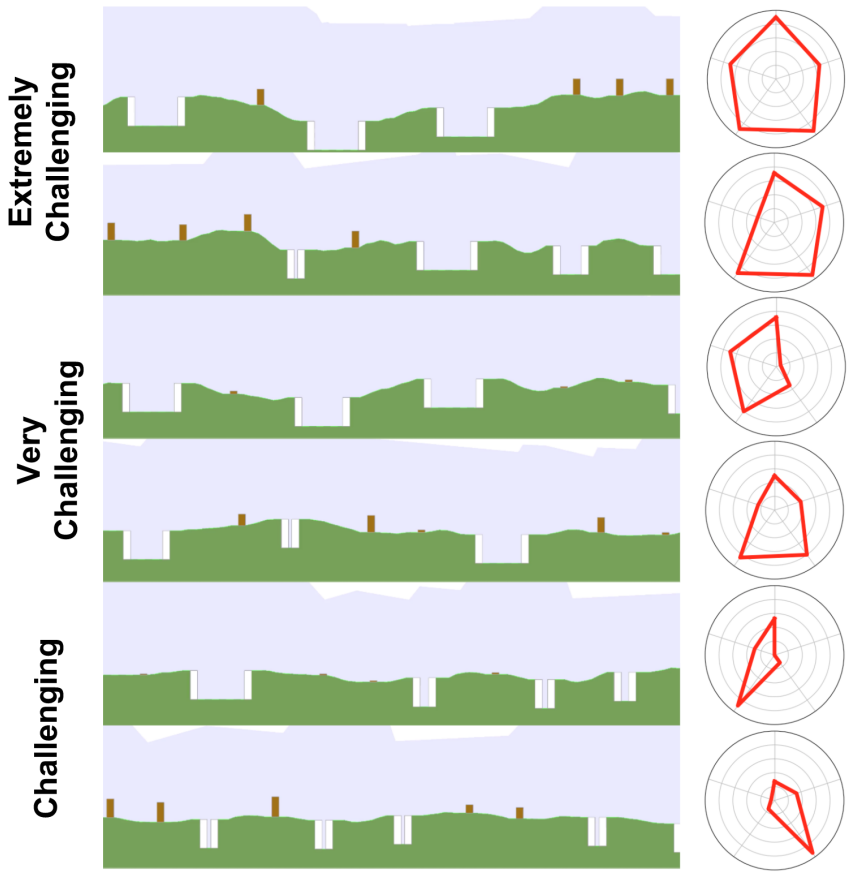


Figure 9: **Sub-sections of environments from the middle column (RUN 2) of figure 5.** These environments exhibit diversity in the levels and ranges of different obstacles. For example, the topmost environment has all large gaps and high stumps with a high amount of roughness; the second from the bottom exhibits a wide variety in gap width, but with trivial stumps and low roughness. In contract, the environment at the very bottom exhibits a wide range of stump heights, but with low gap width and low roughness.

it can produce neural networks with regular geometric patterns in neural connectivity that exploit regularities in the environment to produce regular behaviors that look natural and improve performance [75, 76, 76, 77]. POET allows us to create the interesting combination of using an indirect encoding like a CPPN to encode the environment, the body of the agent, and the neural network controller of that agent, allowing each of the three to benefit from and exploit the regularities being produced by the others.

While ES is the base optimization algorithm in this paper for training solvers for various tasks under POET, including the main transfer mechanism, POET can be instantiated with any RL or optimization algorithm. Trust Region Policy Optimization (TRPO) [78], Proximal Policy Optimization (PPO) [79], genetic algorithms, other variants of ES, and many other such algorithms are all viable alternatives. This plug-and-play aspect of the overall framework presents an intriguing set of future opportunities to hybridize the divergent search across environments with the most appropriate inner-loop learning algorithms, opening up the investigation of open-endedness itself to diverse research communities.

POET could also substantially drive progress in the field of meta-learning, wherein neural networks are exposed to many different problems and get better over time at learning how to solve new challenges (i.e. they learn to learn). Meta-learning requires access to a distribution of different tasks, and that traditionally requires a human to specify this task distribution, which is costly and may not be the right or best distribution on which to learn to learn. Gupta et al. [80] note both that the performance of meta-learning algorithms critically depends on the distribution of tasks they meta-train on, and that automatically producing an effective distribution of tasks for meta-learning

algorithms would represent a substantial advance. That work took an important initial step in that direction by automatically creating different reward functions within one fixed environment [80]. POET takes the important further step of creating entirely new environments (or challenges, more abstractly). While the version of POET in this initial work does not also also create a unique reward function for each environment, doing so is a natural extension that we leave for future work. One possibility is that the reward function could be encoded in addition to the rest of the environment and optimized similarly in parallel.

Another promising avenue for exploration in POET is its interaction with generalization versus specialization. Interestingly, the environments in POET themselves can foster generalization by including multiple challenges in the same sequence. However, alternate versions of POET are also conceivable that take a more explicit approach to optimizing for generality; for example, compound environments could be created whose scores are the aggregate of scores in their constituent environments (borrowing a further aspect of the CMOEA algorithm [45, 46]).

Finally, it is exciting to consider for the future the rich potential for surprise in all the possible domains where POET might be applied. For example, 3-D parkour was explored by Heess et al. [81] in environments created by humans, but POET could invent its own creative parkour challenges and their solutions. The soft robots evolved by Cheney et al. [73] would also be fascinating to combine with ever-unfolding new obstacle courses. POET also offers practical opportunities in domains like autonomous driving, where through generating increasingly challenging and diverse scenarios it could uncover important edge cases and policies to solve them. Perhaps more exotic opportunities could be conceived, such as protein folding for specific biological challenges invented by the system, or searching for new kinds of chemical processes that solve unique problems. The scope is broad for imagination and creativity in the application of POET.

Acknowledgments

We thank all of the members of Uber AI Labs, in particular Joost Huizinga and Zoubin Ghahramani for helpful discussions. We also thank Alex Gajewski (from his internship at Uber AI Labs) for help with implementation and useful discussions. Finally, we thank Justin Pinkul, Mike Deats, Cody Yancey, Joel Snow, Leon Rosenshein and the entire OpusStack Team inside Uber for providing our computing platform and for technical support.

References

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, and M. Bernstein et al. Imagenet large scale visual recognition challenge. *arXiv preprint arXiv:1409.0575*, 2014.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

[5] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. 1998.

[6] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[8] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[9] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[10] Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. *arXiv preprint arXiv:1809.04474*, 2018.

[11] Steven Kapturowski, Georg Ostrovski, Will Dabney, John Quan, and Remi Munos. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=r1lyTjAqYX`.

[12] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Montezuma's revenge solved by go-explore, a new algorithm for hard-exploration problems (sets records on pitfall, too). `https://eng.uber.com/go-explore/`, 2018.

[13] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[14] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[15] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[16] Sébastien Forestier, Yoan Mollard, and Pierre-Yves Oudeyer. Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*, 2017.

[17] Jürgen Schmidhuber. Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in psychology*, 4:313, 2013.

[18] Russell K Standish. Open-ended artificial evolution. *International Journal of Computational Intelligence and Applications*, 3(02):167–175, 2003.

[19] W. B. Langdon. Pfeiffer – A distributed open-ended evolutionary system. In Bruce Edmonds, Nigel Gilbert, Steven Gustafson, David Hales, and Natalio Krasnogor, editors, *AISB'05: Proceedings of the Joint Symposium on Socially Inspired Computing (METAS 2005)*, pages 7–13, 2005. URL `http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/wbl_metas2005.pdf`.

[20] Mark Bedau. The arrow of complexity hypothesis (abstract). In Seth Bullock, Jason Noble, Richard Watson, and Mark Bedau, editors, *Proceedings of the Eleventh International Conference on Artificial Life (Alife XI)*, page 750, Cambridge, MA, 2008. MIT Press. URL `http://www.alifexi.org/papers/ALIFExi-abstracts-0010.pdf`.

[21] Kenneth O. Stanley, Joel Lehman, and Lisa Soros. Open-endedness: The last grand challenge you've never heard of. *O'Reilly Online*, 2017. URL `https://www.oreilly.com/ideas/open-endedness-the-last-grand-challenge-youve-never-heard-of`.

[22] Tim Taylor, Mark Bedau, Alastair Channon, and David Ackley et al. Open-ended evolution: Perspectives from the oee workshop in york. *Artificial life*, 22(3):408–423, 2016.

[23] Joel Lehman and Kenneth O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In Seth Bullock, Jason Noble, Richard Watson, and Mark Bedau, editors, *Proceedings of the Eleventh International Conference on Artificial Life (Alife XI)*, Cambridge, MA, 2008. MIT Press. URL `http://eplex.cs.ucf.edu/papers/lehman_alife08.pdf`.

[24] Lars Graening, Nikola Aulig, and Markus Olhofer. Towards directed open-ended search by a novelty guided evolution strategy. In Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature – PPSN XI*, volume 6239 of *Lecture Notes in Computer Science*, pages 71–80. Springer, 2010. ISBN 978-3-642-15870-4.

[25] L.B. Soros and Kenneth O Stanley. Identifying necessary conditions for open-ended evolution through the artificial life world of chromaria. In *ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, pages 793–800, 2014.

[26] L.B. Soros, Nick Cheney, and Kenneth O Stanley. How the strictness of the minimal criterion impacts open-ended evolution. In *ALIFE 15: The Fifteenth International Conference on the Synthesis and Simulation of Living Systems*, pages 208–215, 2016.

[27] Jonathan C. Brant and Kenneth O. Stanley. Minimal criterion coevolution: A new approach to open-ended search. In *Proceedings of the 2017 on Genetic and Evolutionary Computation Conference (GECCO)*, pages 67–74, 2017.

[28] Thomas S Ray. An approach to the synthesis of life. In *Artificial Life II*, pages 371–408. Addison-Wesley, 1991.

[29] S.G. Ficici and J.B. Pollack. Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states. *Artificial life VI*, page 238, 1998.

[30] R. Paul Wiegand, William C. Liles, and Kenneth A. De Jong. An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, GECCO'01, pages 1235–1242, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-774-9. URL `http://dl.acm.org/citation.cfm?id=2955239.2955458`.

[31] Elena Popovici, Anthony Bucci, R. Paul Wiegand, and Edwin D. De Jong. *Coevolutionary Principles*, pages 987–1033. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-540-92910-9. doi: 10.1007/978-3-540-92910-9_31. URL `http://dx.doi.org/10.1007/978-3-540-92910-9_31`.

[32] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*, 2018.

[33] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[34] Nick Moran and Jordan B. Pollack. Coevolutionary neural population models. *CoRR*, abs/1804.04187, 2018. URL `http://arxiv.org/abs/1804.04187`.

[35] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning*, pages 1514–1523, 2018.

[36] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 482–495, 2017.

[37] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *arXiv preprint arXiv:1707.00183*, 2017.

[38] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011.

[39] Noor Shaker, Julian Togelius, and Mark J Nelson. *Procedural content generation in games*. Springer, 2016.

[40] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *arXiv preprint arXiv:1806.10729*, 2018.

[41] Joel Lehman and Kenneth O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 211–218, 2011.

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

[43] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521:503–507, 2015. doi: 10.1038/nature14422.

[44] Anh Nguyen, Jason Yosinski, and Jeff Clune. Understanding innovation engines: Automated creativity and improved stochastic optimization via deep learning. *Evolutionary Computation*, 24(3):545–572, 2016.

[45] Joost Huizinga, Jean-Baptiste Mouret, and Jeff Clune. Does aligning phenotypic and genotypic modularity improve the evolution of neural networks? In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference (GECCO)*, pages 125–132, 2016.

[46] Joost Huizinga and Jeff Clune. Evolving multimodal robot behavior via many stepping stones with the combinatorial multi-objective evolutionary algorithm. *arXiv preprint arXiv:1807.03392*, 2018.

[47] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[48] Joel Lehman and Kenneth O. Stanley. Novelty search and the problem with objectives. In *Genetic Programming Theory and Practice IX (GPTP 2011)*, 2011.

[49] Justin K Pugh, Lisa B. Soros, and Kenneth O. Stanley. Quality diversity: A new frontier for evolutionary computation. 3(40), 2016. ISSN 2296-9144.

[50] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[51] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

[52] Nikolay Savinov, Anton Raichuk, Raphael Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.0227*, 2018.

[53] Joel Lehman and Kenneth O. Stanley. Revising the evolutionary computation abstraction: minimal criteria novelty search. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 103–110, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0072-8. doi: http://doi.acm.org/10.1145/1830483.1830503. URL `http://doi.acm.org/10.1145/1830483.1830503`.

[54] Ingo Rechenberg. Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie*, pages 83–114. 1978.

[55] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[56] Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *Evolutionary Computation, 2008.*, pages 3381–3387, 2008.

[57] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

[58] Joel Lehman, Jay Chen, Jeff Clune, and Kenneth O. Stanley. Es is more than just a traditional finite-difference approximator. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 450–457, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5618-3. doi: 10.1145/3205455.3205474. URL `http://doi.acm.org/10.1145/3205455.3205474`.

[59] Xingwen Zhang, Jeff Clune, and Kenneth O. Stanley. On the relationship between the openai evolution strategy and stochastic gradient descent. *arXiv preprint arXiv:1712.06564*, 2017.

[60] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, pages 1928–1937, 2016.

[61] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Advances in Neural Information Processing Systems (NeurIPS) 31*, pages 5032–5043. 2018.

[62] Kenneth O Stanley and Joel Lehman. Why greatness cannot be planned. 2015.

[63] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[64] David Ha. Reinforcement learning for improving agent design. *arXiv preprint arXiv:1810.03779*, 2018.

[65] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[66] David Ha. Evolving stable strategies. `http://blog.otoro.net/`, 2017.

[67] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997. URL `gomez:ab97`.

[68] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[69] Andrej Karpathy and Michiel Van De Panne. Curriculum learning for motor skills. In *Canadian Conference on Artificial Intelligence*, pages 325–330. Springer, 2012.

[70] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[71] O. Klimov. Bipedalwalkerhardcore-v2. `https://gym.openai.com`, 2016.

[72] Kenneth O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems*, 8(2):131–162, 2007.

[73] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2013)*, New York, NY, 2013. ACM Press.

[74] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009. URL `http://eplex.cs.ucf.edu/publications/2009/stanley.alife09.html`.

[75] Jeff Clune, Kenneth O. Stanley, Robert T. Pennock, and Charles Ofria. On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, 2011.

[76] Jeff Clune, Benjamin E. Beckmann, Charles Ofria, and Robert T. Pennock. Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009) Special Session on Evolutionary Robotics*, Piscataway, NJ, USA, 2009. IEEE Press.

[77] Jason Yosinski, Jeff Clune, Diana Hidalgo, Sarah Nguyen, Juan Cristobal Zagal, and Hod Lipson. Evolving robot gaits in hardware: the hyperneat generative encoding vs. parameter optimization. In *Proceedings of the 20th European Conference on Artificial Life*, August 2011.

[78] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[79] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[80] Abhishek Gupta, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Unsupervised meta-learning for reinforcement learning. *arXiv*, 2018. URL `http://arxiv.org/abs/1806.04640`.

[81] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. *CoRR*, abs/1707.02286, 2017. URL `http://arxiv.org/abs/1707.02286`.

[82] Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011. URL `http://www.mitpressjournals.org/doi/pdf/10.1162/EVCO_a_00025`.

6 Supplemental Information

The first section in this supplement details the procedure in POET for generating and choosing new environments, which is followed by an explanation of how POET attempts to transfers agents from one environment to another.

6.1 Producing New Environments

New environments are produced by copying the current environments and making random changes to them (in the language of evolutionary algorithms, the environments *reproduce* through *mutation*). Algorithm 3 formalizes the single mutation step, given a list of environment-agent pairs. The primary logic in this process is: (1) Each environment evaluates its paired agent. When the reward obtained by the agent satisfies a predefined reproduction eligibility condition for the environment (denoted as ELIGIBLE_TO_REPRODUCE()), the environment is added to the list of parent environments. (2) Once the list of eligible parents is complete, mutations of the parent environments return a list of child environments (through the function ENV_REPRODUCE()). Each child environment's paired agent is initialized as a clone of its parent environment's paired agent. (3) Each child environment $E^{\text{child}}(\cdot)$, together with its paired agent θ^{child}, is checked against a predefined minimal criterion (reminiscent of the minimal criterion in the MCC algorithm [27]) and only those satisfying the minimal criterion are kept (denoted as MC_SATISFIED()), ensuring that potential new environments are not too trivial and not too hard (which would lead to little progress). (4) All remaining child environments are then ranked based on their novelty (detailed shortly), which is calculated with respect to the current active population of environments and an *archive* of all previously active environments (i.e. those that were allowed to enter the population previously). (5) A transfer attempt is then performed to see whether an agent from a different environment could lead to the best performance in a new child environment. The logic of transfer is described in the next section (Section 6.2). Each pair is then checked against the minimal criterion again in case some pairs no longer satisfy the minimal criterion after the transfer (e.g. if an environment is exposed as too easy).

To limit the usage of computational resources, (6) POET caps the number of child environments that can be created during one iteration of POET's main loop. The number of new environments that can be admitted is `max_admitted` and the maximum number of active environments at any time is `capacity`. If adding new environments exceeds the system's capacity, the oldest environments and their paired agents in the system are removed to make room (also as in the MCC algorithm).

The *encoding* of an environment in POET is its underlying description in the system, such as a vector of numbers that specify the distribution of obstacles in an obstacle course. As mentioned in step (4) above, POET calculates the novelty of the child environments. This novelty calculation is based on the encoding. Child environments are ranked based on their novelty. For readers familiar with the *novelty search* algorithm [82], it is important to note here that the encoding of an environment is in effect a genetic representation, which means that in POET in effect novelty is a *genetic* diversity measure. That way, POET does not in principle require a behavior characterization (BC) [82] to be devised for environments, though doing so is also not precluded.

Measuring novelty introduces an explicit pressure for diversity among the environments by encouraging newly admitted environments to have notably different encodings than those previously admitted. Given an environment E with an encoding denoted as $e(E)$, and a list L containing the encodings of all the environments currently in the population plus those previously admitted but no longer active (which are stored in an *archive*), the novelty of E, denoted as $N(E, L)$, is then computed by selecting the k-nearest neighbors of $e(E)$ from L and computing the average distance between them:

$$N(e(E), L) = \frac{1}{|S|} \sum_{j \in S} ||e(E) - e(E_j)||_2$$
$$S = \text{kNN}(e(E), L)$$
$$= \{e(E_1), e(E_2), ..., e(E_k)\}$$

Note that in this work the distance between genetic encodings of environments is calculated with an $L2$-norm and the number of nearest neighbors k in the calculation of novelty is 5.

Algorithm 3 MUTATE_ENVS

Input: list of environment(E)-agent(A) pairs `EA_list`, each entry is a pair $(E(\cdot), \theta)$
Parameters: maximum number of children per reproduction `max_children`, maximum number of children admitted per reproduction `max_admitted`, maximum number of active environments `capacity`
Initialize: Set `parent_list` to empty
$M = \text{len}(\texttt{EA\_list})$
for $m = 1$ **to** M **do**
  $E^m(\cdot), \theta^m$ = `EA_list`[m]
  if ELIGIBLE_TO_REPRODUCE$(E^m(\cdot), \theta^m)$ **then**
    add $(E^m(\cdot), \theta^m)$ to `parent_list`
  end if
end for
`child_list` = ENV_REPRODUCE(`parent_list`, `max_children`)
`child_list` = MC_SATISFIED(`child_list`)
`child_list` = RANK_BY_NOVELTY(`child_list`)
`admitted` $= 0$
for $E^{\text{child}}(\cdot), \theta^{\text{child}} \in$ `child_list` **do**
  θ^{child} = EVALUATE_CANDIDATES$(\theta^1, \theta^2, \ldots, \theta^M, E^{\text{child}}(\cdot), \alpha, \sigma)$
  if MC_SATISFIED$([E^{\text{child}}(\cdot), \theta^{\text{child}}])$ **then**
    add $(E^{\text{child}}(\cdot), \theta^{\text{child}})$ to `EA_list`
    `admitted` = `admitted` $+ 1$
    if `admitted` $>=$ `max_admitted` **then** break **end if**
  end if
end for
$M = \text{len}(\texttt{EA\_list})$
if $M >$ `capacity` **then**
  `num_removals` $= M -$ `capacity`
  REMOVE_OLDEST(`EA_list`, `num_removals`)
end if
Return: `EA_list`

6.2 ES-Based Transfer

POET adopts a transfer mechanism that allows one agent (the *emigrant*) to attempt to transfer its learned capabilities to a *target environment*, possibly replacing the current target environment's agent. This transfer mechanism reflects the idea that skills gained when interacting with one environment might serve as a stepping stone to improved performance when interacting with another environment. We do not know at any given time which environment's mastery might boost another's, but we can always perform *experiments* to see whether such a transfer might improve performance in the target environment. As results support, the richness of simultaneous transfer attempts in a divergent coevolving system can lead to a succession of breakthroughs that would not be possible through a simple monotonic succession of improvements on a single environment.

Algorithm 4 illustrates the function EVALUATE_CANDIDATES() that facilitates this transfer process. Given a list of M input candidate agents, denoted as $\theta^1, \theta^2, \ldots, \theta^M$, and a target environment $E(\cdot)$, this function first concatenates the list with *proposals*, which are calculated by optimizing each of $\theta^1, \theta^2, \ldots, \theta^M$ in $E(\cdot)$ with one ES step. Then each agent in the augmented list (the original M agents and each of their M proposals) is evaluated in $E(\cdot)$ and the one with the highest reward is returned. As illustrated in Algorithm 2, the agent returned from this function will replace the agent currently paired with $E(\cdot)$ if the former performs better than the latter in $E(\cdot)$. We refer to the replacement as *transfer*, and there are two types: the situation when the replacing agent is from the input candidate list (i.e. without an optimization step on $E(\cdot)$) is referred to as a *direct transfer*, while if the replacing agent is one of the proposals, it is called a *proposal transfer*.

Algorithm 4 EVALUATE_CANDIDATES

Input: candidate agents denoted by their policy parameter vectors θ^1, θ^2, ..., θ^M, target environment $E(\cdot)$, learning rate α, noise standard deviation σ
Initialize: Set list C empty
for $m = 1$ **to** M **do**
    Add θ^m to C
    Add (θ^m+ ES_STEP($\theta^m, E(\cdot), \alpha, \sigma$)) to C
end for
Return: $\text{argmax}_{\theta \in C}\, E(\theta)$