Innocent Until Proven Guilty: Reducing Robot Shaping from Polynomial to Linear Time

Josh C. Bongard

***Abstract*—In evolutionary algorithms, much time is spent evaluating inferior phenotypes that produce no offspring. A common heuristic to address this inefficiency is to stop evaluations early if they hold little promise of attaining high fitness. However, the form of this heuristic is typically dependent on the fitness function used, and there is a danger of prematurely stopping evaluation of a phenotype that may have recovered in the remainder of the evaluation period. Here a stopping method is introduced that gradually reduces fitness over the phenotype's evaluation, rather than accumulating fitness. This method is independent of the fitness function used, only stops those phenotypes that are guaranteed to become inferior to the current offspring-producing phenotypes, and realizes significant time savings across several evolutionary robotics tasks. It was found that for many tasks, time complexity was reduced from polynomial to sublinear time, and time savings increased with the number of training instances used to evaluate a phenotype as well as with task difficulty.**

***Index Terms*—Early stopping, evolutionary robotics.**

I. Introduction

ONE OF THE main criticisms raised against stochastic optimization in general and evolutionary algorithms in particular is that they are computationally inefficient. Most phenotypes turn out to be inferior to the current set of offspring-producing phenotypes. This is particularly true in rugged fitness landscapes, in which phenotypes occupying narrow fitness peaks mostly produce inferior offspring phenotypes that fall on nearby slopes; evolutionary algorithms are designed to perform well in rugged landscapes because other, mostly deterministic optimization methods become trapped on the many local optima. However, in many applications of evolutionary algorithms, the evaluation of a single phenotype takes a significant amount of time, to the point where heuristics are often employed to stop the evaluation of failing phenotypes early.

In some cases, a domain-specific heuristic is employed to terminate the evaluation of phenotypes early if there is no chance that they will recover fitness later in the evaluation. In evolutionary robotics [1], [2], for instance, in which the controllers (e.g., [3]), or the morphologies and controllers (e.g., [4]–[6]) of simulated robots are optimized, a single evaluation can require significant computation time. In this domain, phenotype evaluation is often terminated early if a legged robot falls down and is not equipped to right itself [7], [8] or fails to move during some interval [4], [9].

However, for many complex tasks it is non-trivial to determine whether a phenotype may have recovered fitness in the remaining evaluation time: for example in some situations an immobile robot with a recurrent neural network controller may spontaneously begin moving again. In this paper, an early stopping method is introduced that is domain independent and only stops phenotypes guaranteed to have remained inferior even if they had been evaluated fully. This method involves gradually reducing fitness over evaluation time from some theoretical maximum, rather than accumulating it. Once fitness falls below the worst of the current offspring-producing individuals in the population, its evaluation can therefore safely be terminated.

A related technique in evolutionary robotics employed to reduce computation time is robot shaping [7], [10]–[15], in which initial robots are evaluated in one task environment, and their descendants are evaluated in a growing number of task environments. However, although this may save time compared to evaluating all robots in all task environments, it alters the time complexity of the evolutionary algorithm. Assuming that the number of generations required to evolve a robot that behaves successfully in each new task environment is constant, the time to discover a robot that can behave successfully in k task environments increases at least polynomially with k, if learning task k can influence the robot's ability to maintain the first, second, ... and $k-1$th task. For example once a robot learns one task, that robot's progeny must be evaluated twice: once to assess its ability to perform the new task, and a second time to assess its ability to still perform the first task. It is important to note that the algorithm scales polynomially from the point of view of *cumulative* time: the time to learn two behaviors includes the time to evaluate the first generations of robots against the first task plus the time required to evaluate their progeny against both tasks. Shaping is employed in the work presented here, and it is shown that early stopping accrues increased time savings as the number of task environments in which robots are evaluated increases, and in some cases reduces the search from polynomial to sub-linear time.

Besides computational inefficiency, evolutionary algorithms often suffer from premature convergence. Several broad approaches to guarding against premature convergence exist in the literature. Niching methods [16]–[19] attempt to restrict

Manuscript received October 27, 2009; revised March 12, 2010, August 13, 2010, and November 4, 2010; accepted November 12, 2010. This work was supported by the National Science Foundation, under Grant CAREER-0953837.

The author is with the Department of Computer Science, University of Vermont, Burlington, VT 05405 USA (e-mail: josh.bongard@uvm.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2010.2096540

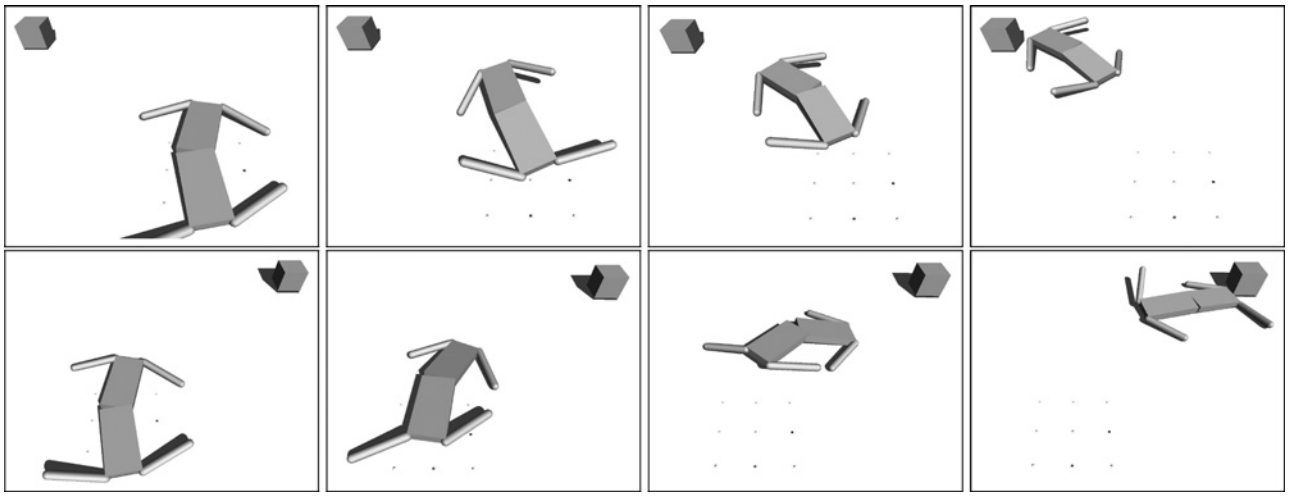

Fig. 1. *A sample evolved locomoting robot.* A typical robot that has evolved to successfully approach objects placed at 12 different locations in an arc in front of it. The upper and lower rows show its behavior when the object is placed at the extreme left-hand and right-hand endpoints of that arc, respectively.

mating to maintain diversity, while novelty-based metrics explicitly or implicitly reward phenotypes not only for gains in fitness, but also for exhibiting uniqueness compared to other members of the population [20]–[22]. Meta-algorithms [23], [24] evolve the evolutionary mechanisms of the algorithm itself, so that convergence in a population is a trait at the meta-level that can be selected against. Recently, several techniques for reducing the probability of becoming trapped at local optima have been introduced [25]–[28]. However, in all of these approaches, the resulting evolutionary algorithms run for much longer: they continue to search even after discovering local optima. These approaches would therefore benefit greatly from early stopping methods that accelerate the search process itself.

The next section describes the conditions necessary to apply the early stopping method to any evolutionary algorithm. Section III introduces the robot tasks, evolutionary algorithm and early stopping method itself. Sections IV–VI provide results, discussion, and concluding remarks, respectively.

II. Necessary Conditions for Employing Early Stopping

Before introducing the details of early stopping, an evolutionary algorithm may make use of early stopping if the following conditions hold.

Multiobjective optimization must be employed, and only those candidate phenotypes that are non-dominated can produce offspring. This is necessary because if a candidate phenotype becomes dominated before its evaluation is finished, evaluation of that phenotype can be terminated early because it is guaranteed that it will not produce offspring.

The fitness function must be strictly non-increasing. At the outset of a phenotype's evaluation its fitness is set to a maximum value, and at each time step of its evaluation fitness stays the same or decreases. This ensures that once a candidate phenotype becomes dominated during evaluation, it cannot become non-dominated again as a result of its fitness increasing later during evaluation.

Fitness may be summed, averaged, or the minimum value may be taken over an evaluation period in which a fitness value is computed at each time step. It should be assumed that if k of n total time steps have been evaluated so far then fitness will be maximal for time steps $k+1, k+2, \ldots, n$. This ensures that if fitness is summed, averaged or the minimum value is taken over the entire interval the statistic will either stay the same or decrease as evaluation continues.

III. Methods

This section describes the two simulated robots used in this paper, as well as the evolutionary algorithm used to evolve behaviors for them. The evolutionary algorithm incorporates two existing mechanisms known to increase the probability of discovering good phenotypes: shaping [7], [10]–[15], and multiobjective optimization [29]. The section concludes by introducing a new mechanism, early stopping, which greatly increases the speed of evolutionary computation.

A. The Robots

To investigate the generality of the performance impact of early stopping, two robots are employed in this paper. A quadrupedal robot was evolved to locomote toward objects placed at different positions (Fig. 1), and an anthropomorphic arm was evolved to manipulate objects of different shape, size, and position (Fig. 2). All of the robots were evolved in the open dynamics engine physics engine. The simulator uses an Euler method with discrete timesteps, and during each time step of a robot evaluation the robot's sensors are updated; the artificial neural network controlling the robot is updated once; outputs from the networks are used to determine the desired angle of the joints; and torque proportional to the difference between the desired and actual angle at each joint is applied to the objects connected by that joint. All of the internal and external forces acting on each object are then added and used to calculate the new position and velocity of the object.

1) *Quadruped:* The quadruped is composed of six body parts connected by five actuated, two degree-of-freedom

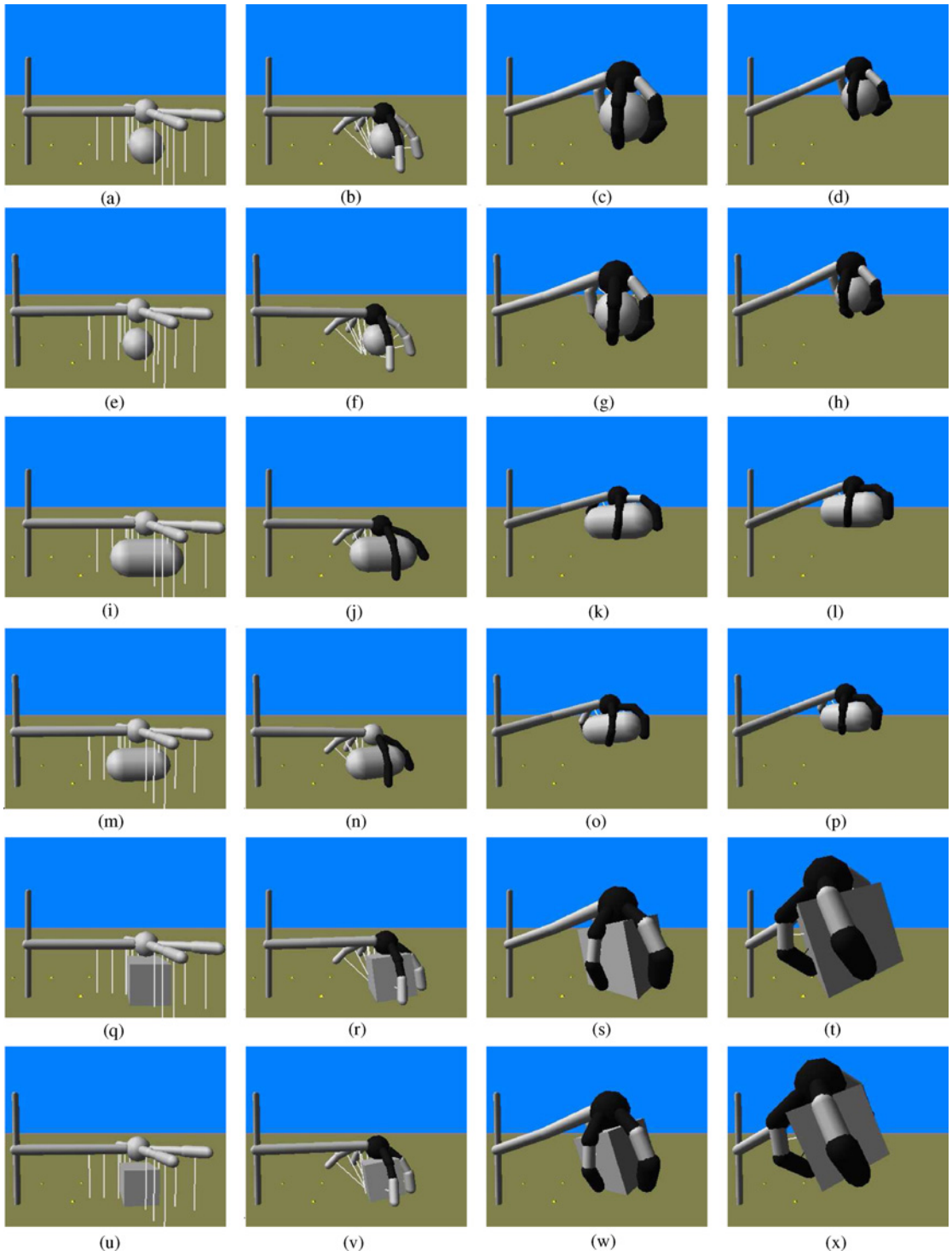

Fig. 2. *A sample evolved arm robot.* This robot can successfully grasp, lift, and actively distinguish between spheres of different sizes [(a)–(d) radius = 35 cm, (e)–(h) r = 31.5 cm], cylinders of difference sizes [(i)–(l) r = 35 cm, length = 70 cm, (m)–(p) r = 31.5 cm, l = 63 cm), and cubes of different sizes [(q)–(t) l, w, h = 70 cm, (u)–(x) l, w, h = 63 cm). Black and white finger segments indicate segments in which the tactile sensor is on or off, respectively. White lines denote the range sensors. Videos of this and other robots can be found on the author's website.

(DOF) rotational joints. Each DOF is composed of a rotational joint and an actuator: the actuator causes the two body parts to rotate relative to one another through some pre-defined plane; the joint acts as the fulcrum. The upper and lower $2 \times 1 \times 0.25$ m trunks are connected by a joint which rotates the parts through the sagittal and coronal planes. The joint rotating through the coronal plane can sweep through $[-50°, 50°]$, the joint rotating through the sagittal plane can only sweep through $[-10°, 10°]$. The four 0.1×1 m legs are attached to their respective trunks by joints that also rotate through the sagittal and coronal planes: rotation through the coronal plane is restricted to $[-50°, 50°]$ and rotation through the sagittal plane is restricted to $[-20°, 20°]$. These joint ranges ensure that the robot keeps its center of mass close to the ground plane and precludes the danger of tipping over. Further, the wide range of horizontal motion facilitates turning during motion.

The robot is equipped with a continuous time recurrent neural network (CTRNN) [30], a common neural network architecture in evolutionary robotics. It is also equipped with seven sensors: four tactile sensors in each leg, and three distance sensors embedded in the left and right shoulders as well as at the robot's midpoint. The tactile sensors return 1 if the leg is in contact with the ground plane or target object, and -1 otherwise. The distance sensor returns a value commensurate with the distance of the sensor from the target object. If the distance sensor is greater than 10m from the target object it returns 0; if it is co-located with the object's center (an impossibility) it returns a 1; and for intermediate distances it returns a value in $[0, 1]$. The CTRNN contains ten neurons that output values to the ten motors which in turn control the five, two DOF joints. Motors receive a value in $[0, 1]$ from the CTRNN; this value is then scaled to a value in the joint's range and treated as a desired angle. Torque is then applied to the joint proportional to the difference between the joint's current and desired angle. At each time step of the simulation the positions, orientations, rotational and linear velocities of each object are updated based on the internal (joint torques) and external (gravitational, inertial, and frictional) forces acting on them. The value of each motor neuron is also updated each simulation time step according to

$$\tau_i y_i' = -y_i + \sum_{j=1}^{m} w_{ji}\sigma(y_j - \theta_j) + \sum_{j=1}^{s_d+s_t} n_{ji} r_j \qquad (1)$$

where τ_i is the time constant associated with neuron i, y_i is the value of neuron i, m is the number of motor neurons ($m = 10$), w_{ji} is the weight of the synapse connecting neuron j to neuron i, $\sigma(x) = 1/(1 + e^{-x})$ is an activation function that brings the value of neuron i back into $[0, 1]$, θ_j is the bias of neuron j, s_d is the number of distance sensors, s_t is the number of tactile sensors, n_{ji} is the weight of the synapse connecting sensor j to neuron i, and r_j is the value of sensor j. In this formulation, each sensor may have a direct effect on every motor neuron. However this effect may be minimized or eliminated by low values for r, or by behaviors that cause a motor neuron to saturate to extremal values. τ_i can range between $[0.0001, 1.0]$, w_{ji} in $[-16, 16]$, θ_i in $[-4, 4]$, and n_{ji} in $[-16, 16]$.

Objects are placed at various locations distal to the robot and it is evolved to locomote toward them. Twelve different placements are used here. The object can be placed at any one of 12 evenly-spaced points on an arc extending in front of the robot; the arc originates at $[10\sin(-\pi/4), 0, 10\cos(-\pi/4)]$ and terminates at $[10\sin(\pi/4), 0, 10\cos(\pi/4)]$ (the robot's center is placed at the center of the simulation's coordinate frame $[0, 0, 0]$). The placement points are therefore

$$\begin{aligned}
&[10\sin(\tfrac{-\pi}{4} + \tfrac{0\pi}{2*11}), 0, 10\cos(\tfrac{-\pi}{4} + \tfrac{0\pi}{2*11})] \\
&= [10\sin(\tfrac{-\pi}{4}), 0, 10\cos(\tfrac{-\pi}{4})] \\
&= [-7.071, 0, 7.071], \\
&[17\sin(\tfrac{-\pi}{4} + \tfrac{1\pi}{2*11}), 0, 10\cos(\tfrac{-\pi}{4} + \tfrac{1\pi}{2*11})] \quad = \quad \ldots = \ldots, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \ldots \quad = \quad \ldots = \ldots, \\
&[10\sin(\tfrac{-\pi}{4} + \tfrac{11\pi}{2*11}), 0, 10\cos(\tfrac{-\pi}{4} + \tfrac{11\pi}{2*11})] \\
&= [10\sin(\tfrac{\pi}{4}), 0, 10\cos(\tfrac{\pi}{4})] = [7.071, 0, 7.071]
\end{aligned}$$

and ensure that the object is always placed 10 m from the robot, and that some placements select for leftward, rightward, or forward locomotion.

2) *The Arm:* In addition to the quadruped, an anthropomorphic robot arm was also employed here as such robots have been studied in evolutionary robotics in the past [31]–[33], and object manipulation allows for the evolution of more diverse behaviors than simple legged locomotion. The robot arm is comprised of a trunk, upper and lower arm, wrist, hand and four fingers containing three phalanges each (Fig. 2). The robot is therefore made up of $4 + 4 \times 3 = 16$ body parts. Like the quadruped, each joint is actuated by a one degree of freedom rotational joint, with the exception of the shoulder joint, a two-DOF actuated rotational joint, and the wrist joint, a three-DOF actuated rotational joint. This gives a total of $6 + 4 \times 3 = 18$ DOFs. If we define the robot's long axis to be the vector that passes from its shoulder through its hand, then the shoulder becomes its posterior point and its hand its anterior point. The sagittal plane is then the vertical plane that cuts through the length of the arm, the coronal plane the horizontal plane that cuts through the length of the arm, and the transverse plane a horizontal plane perpendicular to the sagittal and coronal planes. The arm may be presented with six different objects placed at two different positions. Details regarding the physical characteristics of the objects as well as the robot's body parts and joints are provided in Table I.

In previous work [34], it was found that evolving aspects of the robot's morphology along with its control improves the probability of evolving successful robots. For this reason, along with the controller's parameters the length of each phalange is evolved to any value in the range [0.01 m, 0.6 m], the radius of each phalange is evolved to any value in the range [0.015 m, 0.3 m], and the relative spacings between any pair of fingers i and $i+1$ may change to any angle in $[-\pi, \pi]$.

Each robot is equipped with three binary tactile sensors per finger, one for each phalange, and an additional one in the hand. Each phalange is also equipped with a range sensor (indicated by the white lines in Fig. 2). A range sensor

TABLE I
PHYSICAL PARAMETERS OF THE TARGET OBJECTS AND THE ANTHROPOMORPHIC ROBOT ARM

Part	Length	Height	Width	Mass
Large sphere [S]	0.35 m			1 kg
Large cylinder [Y]	0.35 m	0.7 m		1 kg
Large cube [C]	0.7 m*	0.7 m	0.7 m	1 kg
Small sphere [s]	0.315 m			1 kg
Small cylinder [y]	0.315 m	0.63 m		1 kg
Small cube [c]	0.63 m*	0.63 m	0.63 m	1 kg
Trunk [Tr]	0.05 m	2.0 m		1 kg
Upper arm [Ua]	0.1 m	1.0 m		1 kg
Fore arm [Fa]	0.1 m	1.0 m		1 kg
Hand [Ha]	0.25 m			1 kg
Proximal phalanges [Pp]	0.075 m	0.3 m		1 kg
Intermediate phalanges [Ip]	0.075 m	0.3 m		1 kg
Distal phalanges [Dp]	0.075 m	0.3 m		1 kg
Joint	Min	Max	Plane of rotation	
Shoulder [Tr][Ua]	−30°	30°	Sagittal and coronal	
Elbow [Ua][Fa]	−30°	30°	Coronal	
Wrist [Fa][Ha]	−30°	30°	Sagittal, coronal and transverse	
Metacarpophalangeal [Ha][Pp]	−90°	90°	Relative to finger	
Proximal interphalangeal [Pp][Ip]	−60°	60°	Relative to finger	
Distal interphalangeal [Ip][Dp]	−60°	60°	Relative to finger	

* = length.

returns a value between zero and one commensurate with the distance of the ray emitted by the sensor. This differs from the adirectional distance sensor described for the quadruped above, which can be thought of as a sound or light sensor that responds proportionally to light or volume, respectively. The shoulder contains a proprioceptive sensor that measures the sagittal rotation of the arm: positive values indicate the arm is raised; values near zero indicate the arm is horizontal; and negative values indicate the arm is rotated downward.

As for the quadruped, the arm is controlled by a CTRNN where each of the 18 motor neurons and 3 hidden neurons ($m = 21$) are updated at each simulation time step using

$$\tau_i y_i' = -y_i + \sum_{j=1}^{m+3} w_{ji}\sigma(y_j - \theta_j) + \sum_{j=1}^{s_n+s_t} n_{ji} r_j \qquad (2)$$

where s_r is the number of distance sensors, s_t is the number of tactile sensors, and all other parameters and their allowable ranges are described for (1). Three additional hidden neurons are incorporated for allowing the robot to actively distinguish between different objects. This mechanism is described in more detail when the fitness functions are introduced in Section III-B.

The tactile sensors in both robots are noisy: at each time step, for each tactile sensor, the true value is flipped to the incorrect value with 1% probability. No noise was added to the other sensors or motors.

B. Fitness Functions

The fitness functions for evolving locomotion in the quadruped and object manipulation in the arm follow a similar form

$$f_1 = \frac{\sum_{i=1}^{e}\sum_{j=1}^{t} H(i, j, 1)}{et}, \ldots, f_k = \frac{\sum_{i=1}^{e}\sum_{j=1}^{t} H(i, j, k)}{et} \qquad (3)$$

where $f_1, \ldots, f_k$ are k separate objectives that the robot must maximize, e is the number of environments in which a robot is evaluated, t is the maximum allowed time steps per evaluation, and $H(i, j, k)$ indicates a function corresponding to objective k that transforms the values of the robot's sensors at time step j into a value in $[0, 1]$ when the robot is evaluated in environment i. This formulation ensures that the fitness of the robot is only expressed as a function of its sensor values so that if such experiments were performed by physical robots no external instruments would be required to assess the robot's fitness. Also, the normalization ensures that fitness objectives range from zero for worst fitness to 1 for the theoretical maximum fitness.

Either robot may be evaluated in one or more environments, up to a maximum of 12 in this paper. For the quadruped, each task environment contains an object placed in a different location. The robot arm may be presented with six different objects (Table I) placed at two different positions each: either directly below and slightly to the left of the hand, or below and slightly to the right.

The fitness objectives for the quadruped are therefore defined using $H(i, j, 1) = r_{1,i,j}$, $H(i, j, 2) = r_{2,i,j}$, and $H(i, j, 3) = r_{3,i,j}$, where $r_{d,i,j}$ indicates the value of the dth distance sensor at time step j when evaluated in environment i. This fitness function has the effect of selecting for behaviors which minimize the distances between the distance sensors and the object, thereby bringing the robot into proximity of the target object; each robot is evaluated for 1000 time steps in each of 12 environments.

The first fitness objective for the robot arm selects for grasping and is defined as

$$H(i, j, 1) = G(i, j) = \frac{\sum_{n=1}^{s_n} r_{n,i,j}}{s_n} \qquad (4)$$

where $r_{n,i,j}$ indicates the value of the nth range sensor during the jth time step when the robot arm is evaluated in environment i. Fitness objective G therefore selects for grasping: it selects for strategies that maximize the mean values of the $s_n = 12$ range sensors, thereby selecting for the closing of the fingers around the target object as tightly and rapidly as possible.

The second fitness objective for the robot arm selects for active categorical perception and is defined as

$$H(i,j,2) = A(i,j) = \sum_{n=1}^{12} \begin{cases} 1 - \sqrt{\sum_{h=1}^{3}(y_{h,i,j} - y_{h,n,j})^2}/\sqrt{3} & : \quad S(i) = S(n) \\ \sqrt{\sum_{h=1}^{3}(y_{h,i,j} - y_{h,n,j})^2}/\sqrt{3} & : \quad \text{otherwise} \end{cases} \Big/ 12 \tag{5}$$

where $y_{h,i,j}$ is the value of the hth hidden neuron at time step j when evaluated in environment i, and $S(i)$ returns the shape of the object in environment i. This fitness function is adapted from [33], and selects for behaviors that generate similar time signatures across hidden neurons when the robot interacts with objects of the same shape, but diverging hidden neuron time signatures when presented with objects of different shape. This is accomplished by recording the hidden neuron values during each time step and during each evaluation of a robot, minimizing the Euclidean distance between hidden neuron values across evaluations using objects of the same shape, and maximizing this distance between evaluations using objects of different shape.

The third fitness objective for the robot arm selects for lifting and is defined as

$$H(i,j,3) = L(i,j) \quad = \quad \left(\left(\sum_{t=1}^{s_t} r_{t,i,j} \right) > 0 \right) r_{p,i,j} \tag{6}$$

where $r_{t,i,j}$ indicates the value of the tth tactile sensor during the jth time step when the robot arm is evaluated in environment i, and $r_{p,i,j}$ indicates the angle reported by the proprioceptive sensor in the shoulder at time step j during evaluation in environment i. Fitness objective L therefore selects for lifting: it selects for behaviors that keep at least one phalange and/or the palm in contact with the target object while lifting the arm by rotating upward through the shoulder. The robot can evolve a control policy that allows it to lift without grasping the object firmly: for example in many runs in which only lifting was selected for, the robot would touch the object with the tips of two or three fingers and then lift it.

Multiobjective optimization [29] is used to evolve robots capable of meeting all three of these criteria. Within the evolving population, evaluated phenotypes are divided into dominated and non-dominated phenotypes. A phenotype m is dominated if there exists another evaluated phenotype n in the population that has a higher value across all of the fitness objectives. Only non-dominated phenotypes are allowed to produce offspring; dominated phenotypes are replaced by those offspring.

C. Shaping

As has been shown previously in other domains within [35], [36] and outside of [37], [38] evolutionary algorithms, gradually expanding the set of training instances against which phenotypes in a population are evaluated can increase the probability of finding a good phenotype compared with simply evaluating every phenotype against an entire training set. This principle is known as robot shaping [7], [10]–[15] in robotics, and is derived from the concept of scaffolding in developmental psychology [39] and active learning [37], [40], [41] in machine learning.

Shaping is employed here by initiating each evolutionary run by randomly choosing an environment and evolving the population until a phenotype is found that successfully accomplishes the task in that environment. Thus tasks are defined here as environments in which the robot must act appropriately: we do not consider tasks that can be performed in parallel. Success is defined as a phenotype obtaining a value for each of its fitness objectives above a defined threshold. For the quadruped, the thresholds are set to $f_1 = 0.5$, $f_2 = 0.4$, $f_3 = 0.5$: f_2 is set slightly lower because this distance sensor, embedded at the midpoint of the robot, cannot approach the object as close as the sensors embedded in the shoulders. The thresholds for the robot arm were set to $f_1, f_2, f_3 = 0.85$. Once success is achieved, another environment is added to the training set at random, and evolution continues: phenotypes are now evaluated in both environments. This process continues until a maximum time limit of 30 CPU hours is reached or a phenotype is found that succeeds in all 12 environments.

The success threshold values were empirically established, but it was found that other settings did not affect the results reported below significantly (results not shown). Lower values increase the number of runs that successfully finish before the time limit, and reduce the mean time to completion for those that do; higher values reduce the number of independent runs that finish before time, and increase the time to do so for those that do.

D. Early Stopping

It is common knowledge that the majority of phenotypes produced by evolutionary algorithms as a result of mutation and/or crossover of their underlying genotypes perform significantly worse than their parent phenotypes. This majority can be tuned, but typically a majority of worse performers is desired. For phenotypes that take time to evaluate it would be useful to have a mechanism for terminating poorly-performing phenotypes early. However, this mechanism should not preclude the possibility of prematurely stopping phenotypes that may make up for early underperformance in the latter part of the evaluation. For example, a robot arm that drops an object at the outset may suddenly snatch it up near the end of its evaluation and actually perform better than a parent that began grasping early but never attained a firm grip.

This paper presents an early stopping method that meets both of these criteria: it terminates phenotypes early that would be unable to outperform their parents even if they were run to completion. This is accomplished by starting each phenotype

Algorithm 1

procedure *EarlyStopping*()

```
1:  for i = 0 to 11 do
2:      for k = 0 to 2 do
3:          f(i, k) ← max(k)
4:      end for
5:  end for
6:  i ← 0
7:  d ← 0
8:  while (i < 12) & (!d) do
9:      j ← 0
10:     while (j < 1000) & (!d) do
11:         H(i, j, k) ← evaluate(i, j)
12:         for k = 0 to 2 do
13:             f(i, k) ← f(i, k) − H(i,j,k)/1000
14:         end for
15:         d ← dominated(mean(f))
16:         j ← j + 1
17:     end while
18:     i ← i + 1
19: end while
```

with the maximum obtainable fitness and subtracting value at each time step, rather than beginning with zero fitness and accumulating value at each time step. More specifically, this involves setting all fitness objectives f_i to their maximum possible value and ensuring that the associated H_i function returns negative values at each time step, rather than starting with $f_i = 0$ and H_i returning positive values [see (3)].

This paper reports the results of using two types of early stopping. The first stops the evaluation of a phenotype before the 1000 time steps elapse, referred to as *within stopping*. The other may stop a phenotype between evaluations in different task environments, referred to as *between stopping*. Both mechanisms may be used in isolation or in concert, but rely on the same principle. Pseudocode is given in Algorithm 1.

Algorithmically, this involves computing the fitness for a phenotype as follows: the first block (1–5) initializes each of the k fitness objectives for all i environments, and the second block (6–19) evaluates a single phenotype, but stops evaluation early if the current phenotype becomes dominated by another already-evaluated phenotype in the population. **max**(k) returns the maximum threshold value for fitness objective k, the outer while loop (8–19) evaluates the phenotype in different environments, the inner while loop (10–17) evaluates the phenotype time step by time step in the current environment, **evaluate**(i, j) evaluates the robot in environment i at time step j, the inner for loop (12–14) gradually degrades the fitness objectives of the robot away from their original maximal values, **mean**(f) returns a vector of length three that contains the mean values of each column, thereby reporting the performance of the robot on each objective up to the current time step, and **dominated**(...) indicates whether the current robot is dominated by some other already-evaluated robot in the population.

When a robot is evaluated against a training set of two or more environments, it is evaluated in them in order of recency: it is evaluated against the object most recently added to the training set; if it is not yet dominated (line 8), it is evaluated against the object second-most recently added to the training set; and so on. This sequence is employed because it is likely that a robot will perform most poorly against the most recent object, and therefore have the highest chance of becoming dominated at this point, precluding its evaluation against the remaining objects in the training set.

E. Genetic Algorithm

When early stopping is employed, the time to evaluate different robots varies greatly: phenotypes that reach the non-dominated Pareto front are evaluated for each time step in all of the current training environments, while dominated phenotypes are only evaluated for part of the time in a subset of these environments. For this reason, it is desirable to use a steady state genetic algorithm and distribute evaluation over computational nodes working in parallel. While one node evaluates what is to become a non-dominated phenotype, a second node may evaluate several dominated phenotypes in sequence without having to wait for the first node to complete.

Each run begins with a population of 200 randomly-generated genomes, and is distributed across four computational nodes: the first executes the GA while the remaining three evaluate robots in parallel. Once assigned a robot, a computational node evaluates the robot in the training environments in sequence: because a robot may not be evaluated in all of the training environments, it would be computationally inefficient to further parallelize the algorithm by evaluating each robot in different environments simultaneously.

While the first node waits for a robot's fitness to be returned by the computational nodes, a non-dominated and dominated phenotypes are chosen, both at random: a mutated copy is made of the non-dominated phenotype and overwrites the dominated phenotype. Mutation involves mutating each CTRNN parameter (and morphological parameter for the robot arm) with a probability of 0.05. If a parameter is mutated, the value is replaced with a new random value chosen with a gaussian distribution such that the mean of the distribution is equal to the original value and the variance is equal to the legal range for that parameter. If the new value falls outside the legal range, it is thresholded to its minimum or maximum value.

When a computational node returns a non-dominated phenotype, the population's Pareto front is recalculated. If a node returns a phenotype that succeeded in all of the current i task environments, a new task environment is chosen at random, the current non-dominated phenotypes are re-evaluated against the $i + 1$ environments, the Pareto front is recalculated, and evolution then continues. If 30 h of CPU time elapse on the first node, or a phenotype is found that succeeds in all 12 environments, the run terminates.

IV. Results

One hundred independent runs were conducted with the quadruped robot, and $7 \times 100 = 700$ runs were conducted with the robot arm. Within each of the seven sets of 100 runs with the robot arm, one or more of the fitness objectives G,

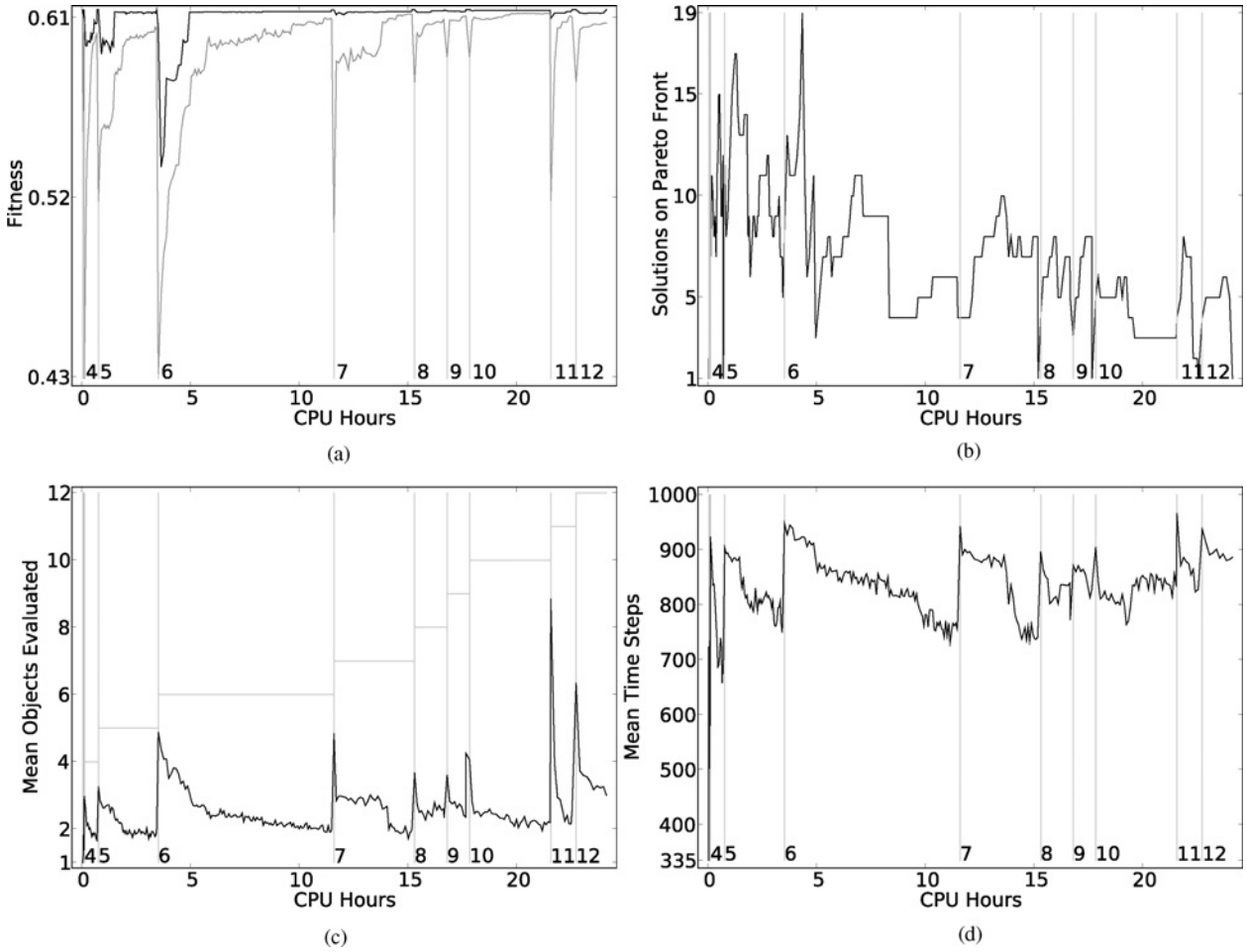

Fig. 3. *Typical evolutionary trajectory when selecting for object manipulation.* (a) Fitness change. (b) Change in front membership. (c) Average objects evaluated per genome. (d) Average time steps evaluated per genome. A sample run drawn from experiments in which the robot arm was evolved to grasp, actively distinguish between and lift 12 objects. (a) reports the best (black line) and mean (gray line) fitness in the population over the length of the run. The three objectives were collapsed to a single fitness value ($f = G \times A \times L$) for the sake of clarity: this run was drawn from a multiobjective run in which grasping, active perception, and lifting served as the three objectives to maximize. Vertical lines indicate the time at which a phenotype was found that could successfully manipulate $n - 1$ objects, triggering the inclusion of an nth object (numbers in the figure panel) in the training set. (b) reports the number of phenotypes occupying the Pareto front. (c) reports the mean number of objects evaluated for each phenotype. The horizontal gray lines indicate how many objects would be evaluated if early stopping between object evaluations were not used. (d) reports the mean number of time steps (out of 1000) evaluated per phenotype.

A or L were used. In the sets of runs in which only one fitness objective was used, the run collapsed to a uni-objective optimization, with only one phenotype in the population occupying the Pareto front at any one time. In the set of runs employing two or more fitness objectives, the run was multiobjective with one or more points occupying the Pareto front.

Fig. 3 reports the evolutionary trajectory of a successful run for the robot arm in which all three fitness objectives were selected for. A successful run is defined as a run in which a phenotype is found that succeeds in all of the task environments before the maximum time limit is reached. A robot that could successfully grasp, actively distinguish between and lift all 12 objects was found after almost 24 h of CPU time had elapsed. It can be seen that robots able to manipulate the first five objects were discovered relatively rapidly, but the sixth object presented a significant challenge [Fig. 3(a)]. Despite this, the best phenotype in the population (black line) was just shy of the success thresholds ($G, A, L = 0.85$) for over five CPU hours. Indeed from five CPU hours onward, there was always at least one phenotype very close to the success threshold.

Fig. 3(b) reports the membership size of the Pareto front for the same run. As can be seen, membership gradually decreases over time as phenotypes cluster ever nearer to the success threshold. When a new object is added the front enlarges as different strategies for manipulating the new object proliferate, but the front membership then decreases again as a few phenotypes converge on and eventually meet the success thresholds.

Fig. 3(c) reports the mean number of environments in which each robot was evaluated (black line). Values less than the total number of environments in the current training set (gray horizontal lines) indicate the workings of between stopping. It can be seen that the average number of environments in which the robots are tested spikes just after a new object is added to the training set. This can be explained by the fact that robots are not yet adept at manipulating the recently-

introduced object, so therefore a newly-generated robot has a high probability of outperforming members of the current Pareto front. If it is still non-dominated after evaluation against the new object, there is a high likelihood that it has retained or only slightly adapted its behavior for manipulating the older objects in the training set, and so will most likely be evaluated on many of the older objects. As time passes the population improves on its ability to manipulate the new object as well as the old ones. If a deleterious mutation is introduced into a newly-generated robot, it is more likely to undermine its ability to manipulate the new object than the older objects (data not shown), so the newly-generated robot is often stopped during evaluation of the new object. Thus, some time after a new object has been added, the mean number of evaluations per genome drops drastically and remains there until a new object is added to the training set.

Most notably, the mean number of environment evaluations does not seem to rise with the number of environments in the training set. A similar pattern is observed in Fig. 3(d), which reports the mean number of time steps evaluated per robot, per environment. Values less than 1000 reflect the use of within stopping. However, there is no consistent rise in the fraction of time employed for evaluating a robot, with perhaps the exception of a seeming rise from 15 CPU hours onward.

A. Performance Impact of Early Stopping

Fig. 4 reports the relative difficulty of the eight tasks. As can be seen, when both active categorical perception and lifting are selected for, less of the runs within those regimes finish in the allotted time, and for that do, they take significantly longer than the mean time to completion for the other tasks.

Fig. 5 reports the empirical time complexity for the algorithm when both within and between stopping are employed (dark triangles) for all eight of these tasks. This is computed by extracting all successful runs from the 100 runs that were conducted and averaging the times across these runs in which phenotypes were found for environment set k, where in this paper $k = 1, 2 \ldots 12$. Rather than running additional algorithm variants in which either within, between or both within and between stopping is disabled, the time those runs would have taken if between shaping was disabled (circles), within shaping was disabled (light triangles), or between and within shaping was disabled (squares) were calculated by adding in the time saved by early stopping. Time savings obtained by between stopping can be visualized as the area between the horizontal grey lines and the black line in Fig. 3(c). Time savings obtained by within stopping can be visualized as the area between the panel's top and the black line in Fig. 3(d).

Nonlinear regression was used to fit a second-order polynomial to each set of times (dotted lines). Clearly, for many of the algorithm variants the times to success increased at least polynomially: The mean time between discoveries of robots successful in k and then $k+1$ environments takes longer than the time between discoveries of robots successful in $k-1$ and then k environments. Table II reports the actual regression models obtained when fitting the times for the four variants across all eight tasks.

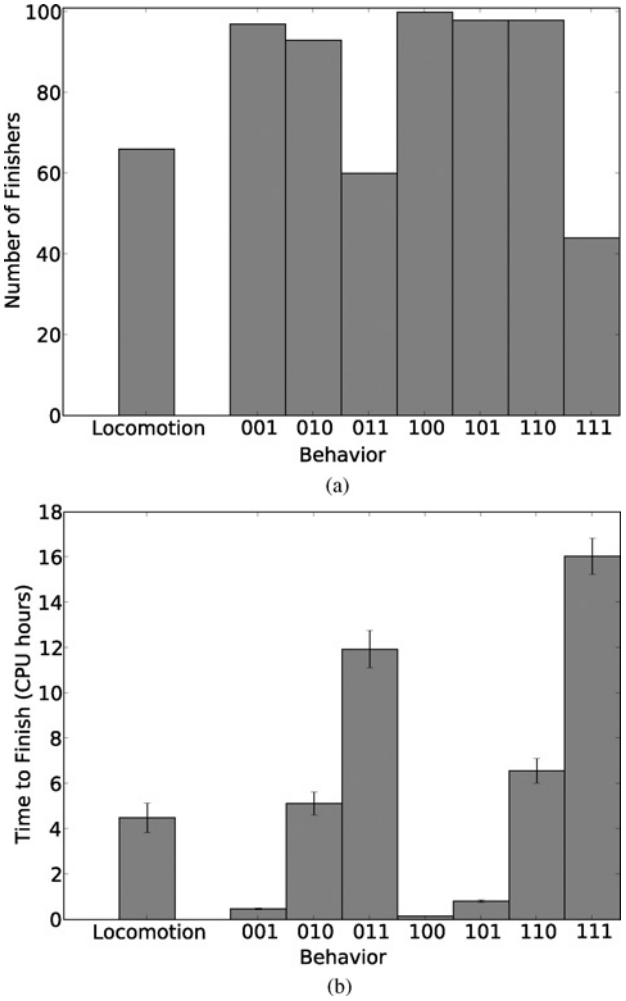

Fig. 4. *Relative difficulty of the locomotion task (left bar) and the seven manipulation tasks (right cluster).* The binary labels indicate which of the three manipulation components was selected for in that experiment group (e.g., 110 indicates grasping was selected for, active categorical perception was selected for, but lifting was not selected for). (a) How many of the 100 runs completed successfully in the allotted time. (b) From among those runs that completed, the mean time required for completion. Error bars report standard error of the mean.

As can be seen in the table, for the two algorithm variants in which between stopping is disabled (BS = 0) the time to discover a successful phenotype for each new environment scales polynomially or greater: each of the nonlinear terms is positive. However, when between stopping is used for the locomotion task the time complexity reduces from polynomial $(0.004x^2, 0.0031x^2)$ to sub-linear time $(-0.0004x^2, -0.0002x^2)$. This can also be seen in Fig. 5(a): the curves without between stopping (squares, light triangles) are convex while those with between stopping are concave (circles, dark triangles). The sublinearity of these algorithms variants are supported by the goodness of fit indicated by their high r^2 values ($r^2 = 0.993, 0.994$). The lifting task also achieved sublinear time complexity when between stopping was employed (second left-hand column in Table II; $r^2 = 0.994, 0.998$). For combined grasping and lifting, sublinear time complexity was achieved when between stopping was employed but within stopping was

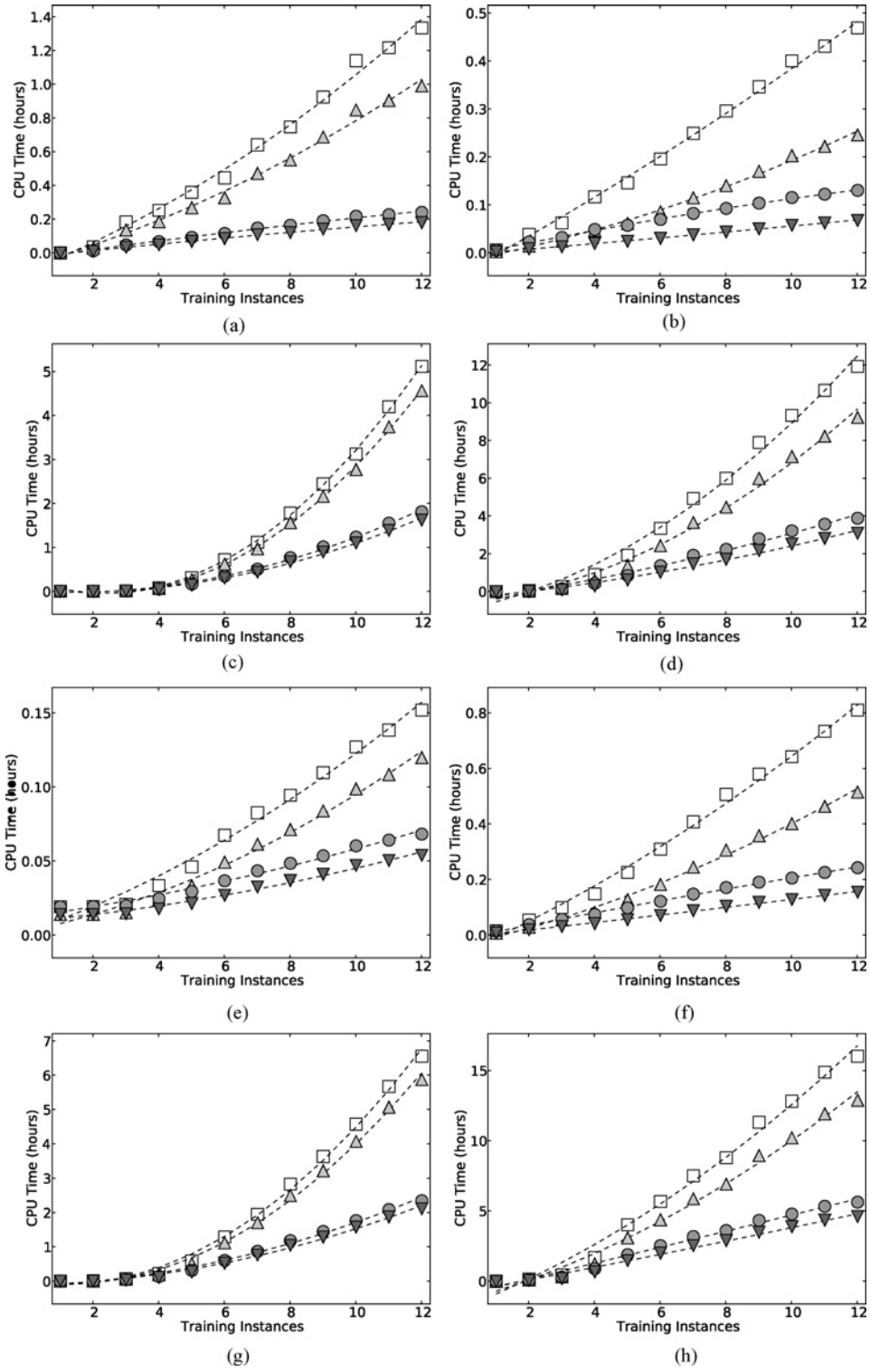

Fig. 5. *Relative timings for the four regimes.* (a) Locomotion. (b) Manipulation: G = 0, A = 0, L = 1. (c) Manipulation: G = 0, A = 1, L = 0. (d) Manipulation: G = 0, A = 1, L = 1. (e) Manipulation: G = 1, A = 0, L = 0. (f) Manipulation: G = 1, A = 0, L = 1. (g) Manipulation: G = 1, A = 1, L = 0. (h) Manipulation: G = 1, A = 1, L = 1. Each panel reports the speed to completion for the eight behaviors selected for [(a) locomotion, (b)–(h) object manipulation variants]. Each marker indicates the mean time required for successful runs to conquer that training set size. Black triangles denote the actual time taken when phenotypes are stopped early within evaluations ("within stopping") and between evaluations ("between stopping"). Circles report the predicted time if evaluations were stopped early between evaluations but not within evaluations. Gray triangles report the predicted time if evaluations were stopped within evaluations but not between evaluations. Squares report the predicted time if neither early stopping condition was used. The dotted line represents the fit regression line using a second-order polynomial.

TABLE II
NONLINEAR REGRESSION MODELS OBTAINED FROM THE TIME TO SUCCESS FOR ALL EIGHT TASKS

		Locomotion	Manipulation						
			G = 0	0	0	1	1	1	1
			A = 0	1	1	0	0	1	1
			L = 1	0	1	0	1	0	1
Nonlinear regression models									
WS = 0	BS = 0	$0.004x^2$ $+0.0767x$ -0.1054	$0.0005x^2$ $+0.0375x$ -0.0413	$0.0554x^2$ $-0.2573x$ $+0.2561$	$0.0659x^2$ $+0.3291x$ -0.9247	$0.0004x^2$ $+0.0077x$ $+0.0025$	$0.0018x^2$ $+0.053x$ -0.0619	$0.0573x^2$ $-0.1196x$ -0.0348	$0.0541x^2$ $+0.904x$ -1.8437
WS = 1	BS = 0	$0.0031x^2$ $+0.0556x$ -0.0755	$0.0008x^2$ $+0.0128x$ -0.0158	$0.0502x^2$ $-0.2406x$ $+0.2485$	$0.0566x^2$ $+0.1803x$ -0.6159	$0.0004x^2$ $+0.0047x$ $+0.0028$	$0.0016x^2$ $+0.0274x$ -0.0347	$0.0525x^2$ $-0.1237x$ -0.0075	$0.0494x^2$ $+0.6484x$ -1.3855
WS = 0	BS = 1	$-0.0004x^2$ $+0.0281x$ -0.0335	$-0.0002x^2$ $+0.0145x$ -0.0067	$0.0162x^2$ $-0.0382x$ $+0.0044$	$0.0118x^2$ $+0.2399x$ -0.4831	$0.0001x^2$ $+0.0031x$ $+0.0126$	$-0.0002x^2$ $+0.0234x$ -0.0094	$0.0154x^2$ $+0.0307x$ -0.1224	$0.0006x^2$ $+0.5631x$ -0.9591
WS = 1	BS = 1	$-0.0002x^2$ $+0.0205x$ -0.0243	$0.0x^2$ $+0.0055x$ -0.0027	$0.0148x^2$ $-0.0394x$ $+0.0134$	$0.0115x^2$ $+0.1633x$ -0.3537	$0.0002x^2$ $+0.002x$ $+0.0094$	$0.0001x^2$ $+0.013x$ -0.0067	$0.0144x^2$ $+0.0206x$ -0.102	$0.0027x^2$ $+0.4333x$ -0.7634
r^2 values from linear regression									
WS = 0	BS = 0	0.985	0.995	0.881	0.964	0.978	0.99	0.924	0.98
WS = 1	BS = 0	0.984	0.985	0.878	0.958	0.973	0.985	0.921	0.977
WS = 0	BS = 1	0.993	0.994	0.921	0.979	0.978	0.997	0.954	0.986
WS = 1	BS = 1	0.994	0.998	0.917	0.975	0.973	0.997	0.951	0.986

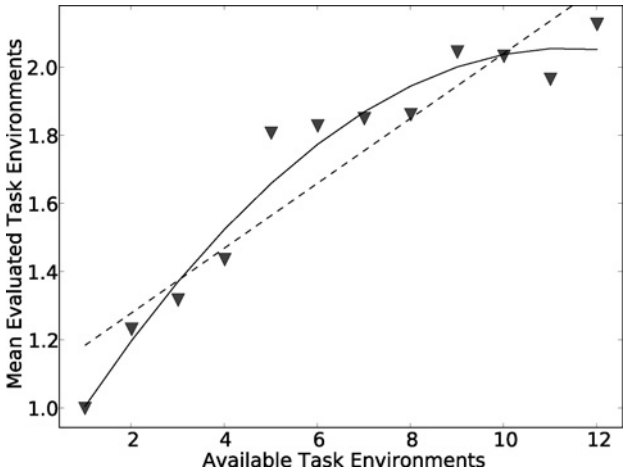

Fig. 6. *Mean objects evaluated per genome.* The mean time to successfully conquer a set of training instances was averaged over all eight behaviors when both within and between early stopping was employed. The solid line represents the fit regression line using a second-order polynomial ($r^2 = 0.96$). The dotted line represents the fit linear regression line ($r^2 = 0.87$).

disabled ($-0.0002x^2, r^2 = 0.997$). For the remaining tasks it can be observed that the nonlinear terms are typically at least an order of magnitude lower when between stopping is employed, compared to when it is disabled.

These results suggest that between stopping not only reduces the mean time required to evaluate phenotypes in an evolutionary algorithm, but that the ratio of time saved to the total time required to evaluate a phenotype increases with total evaluation time. As shaping is employed here, this means that the ratio of time saved to the total evaluation time increases along with the number of task environments in which the robot is evaluated. This observation is supported by Fig. 6, which reports the mean number of task environments in which robots are evaluated, over all eight tasks, as a function of the total number of task environments available for evaluation. The dotted line indicates the linear regression model and assumes that the ratio of available (horizontal axis) to evaluated (vertical axis) time remains the same as the number of available task environments increases. This model does not fit ($r^2 = 0.87$) as well as a second-order regression model (dashed line, $r^2 = 0.96$) which indicates that the ratio of available to evaluated task environments increases with the number of available task environments.

Finally, there is evidence that early stopping reduces the nonlinearity of the algorithm's time complexity most for difficult tasks. Using the fraction of runs that successfully finish in time as a measure of task difficulty, Fig. 4(a) indicates that the task requiring object grasping, active categorical perception and lifting is the most difficult task ("111"), followed by combined active categorical perception and lifting ("011"), followed by locomotion.

For this most difficult task when neither form of stopping is employed, the nonlinear contribution to the time complexity is $(0.0541[x^2]/(0.904[x]+0.0541[x^2])) \times 100 = 5.64\%$, while the nonlinear contribution when both forms of stopping are employed is only $(0.0027[x^2]/(0.4333[x] + 0.0027[x^2])) \times 100 = 0.62\%$ (right-hand column in Table II). The nonlinearity contribution therefore decreases by $5.64/0.62 \times 100 = 909.67\%$ when stopping is employed. For the second-most complex task the nonlinearity contribution decreases by 253.59% when stopping is employed (fifth column in Table II). For the third-most complex task (the locomotion task), the nonlinearity contribution decreases by 513.01% when stopping is employed (second column in Table II).

In contrast, for easy tasks such as just object grasping ("100") or grasping and active categorical perception ("110"), the change in the nonlinear contribution to the time complexity is 54.32% and 78.73%, respectively.

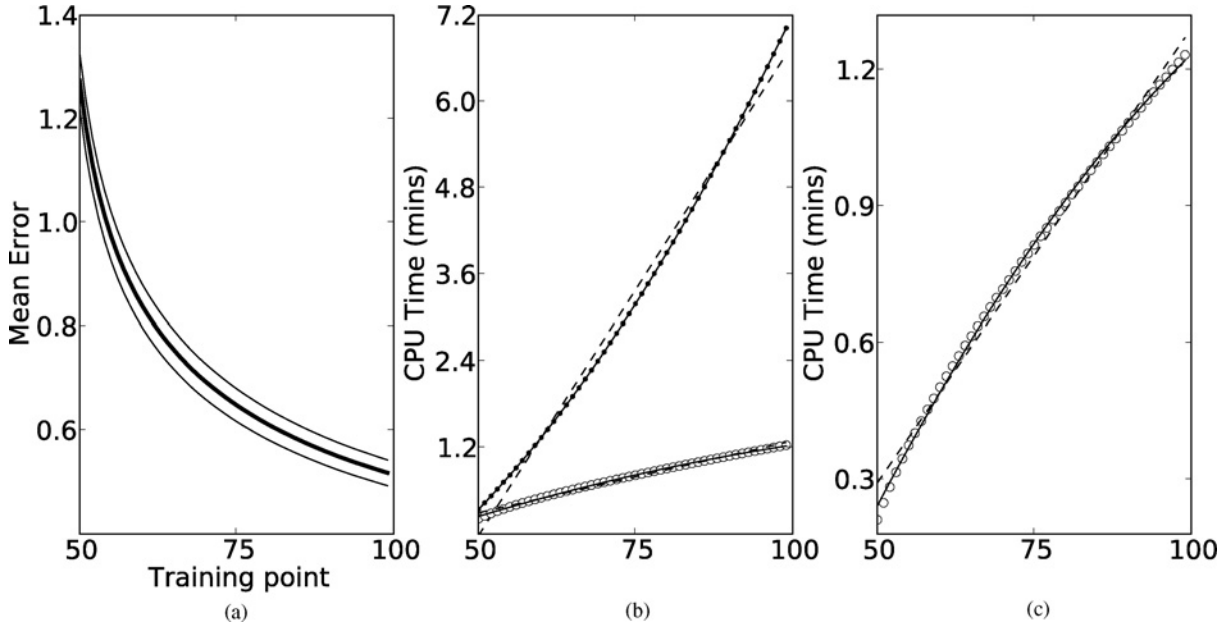

Fig. 7. *Effect of early stopping on symbolic regression.* (a) Genetic programming gradually reduces the error of expression trees with a maximum depth of four over 50 generations. Initially 50 training points are added to the training set, after each subsequent generation, a new training point is added. (Thick line indicates mean averaged over 100 independent runs, thin lines indicate one unit of standard error of the mean.) (b) Dots indicate the mean cumulative time (in CPU minutes) that it took each generation to complete. Circles indicate the mean cumulative time for the generations to complete when early stopping is employed. The dotted line indicates the linear regression line for each of the two experimental regimes. The solid line indicates the regression line using second regression. (c) Cumulative times for only the regime using early stopping.

V. Discussion

Here, a method has been introduced for greatly accelerating evolutionary algorithms by reducing a theoretical maximum fitness value over the length of an evaluation, rather than accumulating fitness over evaluation time. When employing shaping [7], [10]–[15] or active learning [37], [40], [41] in which the number of training instances against which an evolved phenotype must be evaluated increases over the run, stopping an evaluation early may happen during the evaluation of a single training instance (within stopping), or between training instances (between stopping).

It was found that both within and between stopping accelerate an evolutionary run, but between shaping in particular realizes the greatest time savings. Indeed for three of the eight evolutionary robotics tasks tested here, between stopping reduces the running time of the algorithm from polynomial to sub-linear time [Fig. 5(a), (b), (f)]. Sub-linear time complexity is possible because the ratio of saved time to total available evaluation time increases with the number of training instances when between stopping is employed.

This increasing time saving as evaluation time increases can be explained by observing that when a new training instance is included in the training set early in an evolutionary run, it has a high probability of presenting the current population of phenotypes with a novel and therefore difficult situation, thereby greatly reducing their fitness. This can be observed in Fig. 3, when the addition of the fifth and sixth objects causes a drop in fitness [Fig. 3(a)] and a sudden increase in the size of the Pareto front's membership [Fig. 3(b)]. A drop in fitness increases the likelihood that a new phenotype must be evaluated for longer, against more training instances, to determine whether it will eventually perform worse than its parent. This degrades the time savings of between stopping, as shown in Fig. 3(c): after the introduction of the sixth object, phenotypes were evaluated against a relatively large number of objects, slowing search temporarily until much more fit phenotypes were discovered around the fifth CPU hour.

As a run continues however, phenotypes must generalize to solve the larger number of training instances. This generalization ensures that the introduction of a new training instance later in the run is likely to cause a much less drastic drop in fitness. This can be seen in the very minute fitness drops for objects seven to 12 in Fig. 3(a). Correspondingly, as phenotypes retain their high fitness even after the introduction of a new training instance, early stopping can rapidly determine the inferiority of a new phenotype: as fitness is subtracted rather than added in early stopping, it does not require much time for new, inferior phenotypes to have their fitness reduced below the level of the non-dominated phenotypes in the population. This phenomenon can be observed as the very narrow spikes in mean evaluated objects appearing shortly after the introduction of the seventh to twelfth objects in Fig. 3(c). The existence of the spikes indicate that although the previously successful phenotype may not behave well in the presence of a new training instance, the population rapidly evolves a phenotype that can. Therefore, early stopping provides more of a time saving benefit as a run proceeds, and explains why not only does early stopping accelerate an evolutionary run, but can, in some cases, reduce its running time from polynomial to sub-linear time.

Finally, evidence was provided that the speed benefit of early stopping increases not only with phenotype evaluation time, but also with the difficulty of the task at hand. For three tasks that rarely finished in the maximum allotted time, early stopping greatly reduced the nonlinear component of

those runs' time complexity. For easy tasks in which the majority finished in time, there was relatively little reduction in the nonlinear components of their time complexities. It is hypothesized (although not yet shown) that the fitness landscapes for these easier tasks are smoother. This means that for any given parent, the mean fitness of its children is higher than an equivalent parent's offspring in a rugged landscape. In a smooth fitness landscape, children would therefore have to be evaluated for longer to determine whether they are inferior to their parents, compared to children in a rugged fitness landscape. Therefore early stopping would be of less benefit in a smooth fitness landscape.

Early stopping was demonstrated on a series of evolutionary robotics tasks, and therefore raises the question as to the generality of this approach. To address this, early stopping was applied to a standard genetic programming application domain, symbolic regression.

A. Early Stopping for Symbolic Regression

Symbolic regression is one of the most popular application domains for genetic programming, and the tree structures of GP genomes lend themselves well to encoding arbitrary mathematical expressions. Early stopping was used to accelerate symbolic regression in the following manner.

Four hundred independent runs of symbolic were performed: 100 runs were performed against a hidden target expression tree with length four, five, six and seven, respectively. At the outset of each run, the target expression tree was constructed by selecting either an operator or an operand with equal probability; if the new node lay at the target tree's maximum depth, only an operand was added. Valid operators were addition, sine and cosine; valid operands were the independent variable $\mathbf{x}$ or a constant value in $[-1, 1]$. Both the dependent variable $\mathbf{x}$ and the dependent variable $\mathbf{y}$ are vectors of length 1000. Vectors are used instead of scalars because when early stopping greatly reduces the evaluation time of a single expression tree, if each node only performs a scalar calculation, the mechanics of the evolutionary algorithm can actually take much longer than evaluating new members of the population. This overhead makes it more difficult to determine the true running time of the algorithm. However, this is not seen as a problem as it is argued here that early stopping is of value for domains in which evaluating phenotypes takes a considerable amount of time, which is the case for vector-based symbolic regression.

Fifty training points $\mathbf{x_1}, \ldots \mathbf{x_{50}}$ were then created in which each element in $\mathbf{x_i}$ is randomly selected from $[-1, 1]$. Each point is evaluated by the target function $t(\mathbf{x})$, yielding $\mathbf{y_1^{(t)}} = \mathbf{t}(\mathbf{x_1}), \ldots \mathbf{y_{50}^{(t)}} = \mathbf{t}(\mathbf{x_{50}})$. A population of 2000 expression trees were then created in the same manner as the target expression tree, and each was evaluated against the 50 training points. The error of each expression tree i was then computed using

$$e_i = \sum_{j=1}^{50} \frac{\sum_{k=1}^{1000} (y_{j,k}^{(t)} - y_{j,k}^{(i)})^2}{1000}$$

where $y_{j,k}^{(t)}$ is the kth element of the result of the target expression tree's evaluation of the jth training point, and $y_{j,k}^{(i)}$ is the kth element of the result of the ith expression tree's evaluation of the jth training point.

Diversity in the population was maintained by employing deterministic crowding [16], a popular diversity-maintenance method. After all current expressions are evaluated, they are randomly grouped into 1000 pairs. Each pair is copied, and each of the resulting two expression trees are mutated and crossed. Mutation involved selecting a node at random, and replacing it with an operator or operand at random; the node's branches are either deleted or randomly grown to match the mutated node's arity. Probabilistic functional crossover [42] was also performed after the two trees were mutated.

Deterministic crowding requires a metric d that can compute the distance between two phenotypes i and j. This metric is formulated here as $\sum_{k=1}^{50} \sum_{m=1}^{1000} |y_{k,m}^{(i)} - y_{k,m}^{(j)}|$. Once the two child expression trees have been aligned with their more similar parent, if a child tree incurs less error than its parent, it replaces it; otherwise, it is discarded.

After one generation elapses, a new training point $\mathbf{x_{51}}$ is created at random and added to the training set. The existing phenotypes are evaluated on this new point, and their errors are adjusted. The 2000 new child phenotypes are evaluated on all 51 training points, and deterministic crowding is again performed. This process continues for 50 generations such that in the last generation the final cohort of expression trees are evaluated against 100 training points.

An additional 400 independent runs were then performed using the same 400 target expression trees from the control experiments. In these runs however, early stopping is employed. Since two children only compete against their two parents, as they are evaluated against the points in the training set, evaluation of a child can be stopped once its error grows beyond that of both parents. In such a case no matter which parent a child is aligned with, it will not replace it even if it is evaluated on the remaining training points.

Fig. 7 reports the running times of these two experiment variants for the experiments in which the target tree had a depth of seven. As can be seen, running time is more than linear when early stopping is not employed [Fig. 7(b)]. When second-order regression is applied to these data, the regression model is $1.49 \times 10^{-5}x^2 - 1.19 \times 10^{-5}x - 0.03$ with $r^2 = 0.9999998$. When linear regression is applied, the fit is not as good, yielding $r^2 = 0.992$. Conversely, when early stopping is employed, running time is sublinear. When second-order regression is applied, the regression model is $-2.06 \times 10^{-6}x^2 + 6.36 \times 10^{-4}x - 0.02$ with $r^2 = 0.9991$. When linear regression is applied, the fit is not as good, yielding $r^2 = 0.992$.

This supports the claim that in some cases, early stopping can not only provide a significant time savings, but can also reduce the running time of an evolutionary algorithm from polynomial or exponential time complexity to sublinear time complexity. Moreover, this phenomenon is not limited to the problem domain or type of evolutionary algorithm used.

VI. Conclusion

One of the major criticisms raised against stochastic optimization methods in general and evolutionary algorithms

in particular is that they are computationally inefficient: a vast majority of time is spent evaluating inferior phenotypes. However, the exponential increase in computing power is often cited as reducing the relevance of this criticism: there is, or will shortly be sufficient computing power to tackle increasingly complex tasks. This argument though fails to take into consideration how evolutionary algorithms scale with task complexity. This paper provided some insight into this issue by demonstrating that for a series of evolutionary robotics tasks in which the robots are gradually exposed to an increasing number of task environments, the running time increases at least polynomially with the number of task environments.

Here it was demonstrated that by using an early stopping method that prematurely stops inferior phenotypes, the speed of evolving successful phenotypes for all of the tasks tested could be increased. Moreover, the time complexity of some of the tasks could be reduced from polynomial to sublinear time. This method ensures that only phenotypes guaranteed to become inferior to the current best set of phenotypes if they are evaluated fully are stopped early.

It was shown that the benefit of early stopping increases with the number of training instances that an evolved phenotype must be evaluated against, and also with the difficulty of the task. This suggests that this or other early stopping methods will become increasingly necessary in ever more difficult tasks in which phenotypes must be evaluated against many training instances.

Indeed in evolutionary robotics, in order to realize robots capable of executing several tasks in parallel or in sequence [32], and operating in noisy or nonstationary environments [3], [34] it becomes necessary to evaluate robots in hundreds and most likely thousands of task environments. Furthermore, this approach could be used to address the reality gap problem [3], [43], in which the same phenotype must be evaluated several times in different simulations to keep phenotypes from exploiting hidden inaccuracies in any one simulation.

Of course, there are other applications besides evolutionary robotics in which phenotype evaluation is computationally expensive. The application of early stopping to vector-based symbolic regression and observing the same reduction in running time complexity (Section V-A) suggests that this property of early stopping is domain independent. This application also demonstrates that it is not always necessary to calculate a maximum fitness (or minimum error) *a priori*: as long as the fitness or error calculation is guaranteed to increase (or decrease) monotonically, it is ensured that a prematurely-terminated phenotype could not have later improved and qualified for membership in the breeding pool.

There are many shaping or active learning methods in which individual evaluations are somehow separable, and therefore do not scale polynomially: for example if a phenotype must be evaluated k times and the evaluations are independent, they may be evaluated in parallel on k computational nodes. However, it should be noted that significant time savings were accrued through the use of early stopping on all tasks studied, regardless of whether the problem proved to scale polynomially without early stopping.

Another criticism leveled against evolutionary methods is that they prematurely converge. Meta-heuristics have recently been introduced that ensure that evolution continues to explore the search space, thereby increasing the probability of finding globally optimal phenotypes and thus increasing the consistency of a single evolutionary run [25]–[28]. However, such constant searching lengthens the time required for an evolutionary run. In future work, integration between these meta-heuristics and early stopping is planned to increase both the speed and consistency of evolutionary methods.

REFERENCES

[1] I. Harvey, P. Husbands, D. Cliff, A. Thompson, and N. Jakobi, "Evolutionary robotics: The Sussex approach," *Robot. Autonomous Syst.*, vol. 20, nos. 2–4, pp. 205–224, 1997.

[2] S. Nolfi and D. Floreano, *Evolutionary Robotics*. Boston, MA: MIT Press, 2000.

[3] J. Bongard, V. Zykov, and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, vol. 314, pp. 1118–1121, Nov. 2006.

[4] K. Sims, "Evolving 3-D morphology and behavior by competition," *Artif. Life IV*, vol. 1, no. 4, pp. 28–39, 1994.

[5] H. Lipson and J. B. Pollack, "Automatic design and manufacture of artificial lifeforms," *Nature*, vol. 406, pp. 974–978, Aug. 2000.

[6] J. C. Bongard, "Evolving modular genetic regulatory networks," in *Proc. IEEE CEC*, May 2002, pp. 1872–1877.

[7] T. Reil and P. Husbands, "Evolution of central pattern generators for bipedal walking in a real-time physics environment," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 159–168, Apr. 2002.

[8] J. C. Bongard and C. Paul, "Making evolution an offer it can't refuse: Morphology and the extradimensional bypass," in *Proc. 6th Eur. Conf. Artif. Life*, 2001, pp. 401–412.

[9] J. Bongard, "Behavior chaining: Incremental behavioral integration for evolutionary robotics," in *Proc. 11th Artif. Life*, 2008, pp. 64–71 [Online]. Available: http://www.alifexi.org/papers/ALIFExi_pp064-071.pdf

[10] S. P. Singh, "Transfer of learning across sequential tasks," *Mach. Learning*, vol. 8, no. 3, pp. 323–339, 1992.

[11] M. Dorigo and M. Colombetti, "Robot shaping: Developing autonomous agents through learning," *Artif. Intell.*, vol. 71, no. 2, pp. 321–370, 1994.

[12] L. Saksida, S. Raymond, and D. Touretzky, "Shaping robot behavior using principles from instrumental conditioning," *Robot. Autonomous Syst.*, vol. 22, nos. 3–4, pp. 231–250, 1997.

[13] J. Pratt, C. Chew, A. Torres, P. Dilworth, and G. Pratt, "Virtual model control: An intuitive approach for bipedal locomotion," *Int. J. Robot. Res.*, vol. 20, no. 2, pp. 129–143, 2001.

[14] M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini, "Developmental robotics: A survey," *Connection Sci.*, vol. 15, no. 4, pp. 151–190, 2003.

[15] T. Ziemke, N. Bergfeldt, G. Buason, T. Susi, and H. Svensson, "Evolving cognitive scaffolding and environment adaptation: A new research direction for evolutionary robotics," *Connection Sci.*, vol. 16, no. 4, pp. 339–350, 2004.

[16] S. Mahfoud, "Niching methods for genetic algorithms," Ph.D. dissertation, Dept. Industr. Enterprise Syst. Engineer. (IESE), Univ. Illinois at Urbana-Champaign, Urbana, 1995.

[17] E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs Paralleles, Reseaux Syst. Repartis*, vol. 10, no. 2, pp. 141–171, 1998.

[18] D. Whitley, S. Rana, and R. Heckendorn, "The island model genetic algorithm: On separability, population size and convergence," *J. Comput. Inform. Technol.*, vol. 7, no. 1, pp. 33–48, 1999.

[19] E. Alba and B. Dorronsoro, *Cellular Genetic Algorithms*. Berlin, Germany: Springer-Verlag, 2008.

[20] D. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Proc. 2nd Int. Conf. Genet. Algorithms Their Applicat. Table Contents*, 1987, pp. 41–49.

[21] P. Darwen and X. Yao, "Every niching method has its niche: Fitness sharing and implicit sharing compared," in *Lecture Notes in Computer Science*, vol. 1141. New York: Springer, 1996, pp. 398–407.

[22] J. Hu, E. Goodman, K. Seo, Z. Fan, and R. Rosenberg, "The hierarchical fair competition (hfc) framework for sustainable evolutionary algorithms," *Evol. Comput.*, vol. 13, no. 2, pp. 241–277, 2005.

[23] B. Freisleben, "Meta-evolutionary approaches," in *Evolutionary Computation: Advanced Algorithms and Operators*. Bristol, U.K.: Institute of Physics Publishing, 2000, p. 212.

[24] J. Moore, C. Greene, P. Andrews, and B. White, "Does complexity matter? Artificial evolution, computational evolution and the genetic analysis of epistasis in common human diseases," in *Genetic Programming Theory and Practice VI*. New York: Springer, 2008, p. 125.
[25] R. Ursem, "Diversity-guided evolutionary algorithms," in *Lecture Notes in Computer Science*, vol. 2439. New York: Springer, 2003, pp. 462–474.
[26] G. S. Hornby, "ALPS: The age-layered structure for reducing the problem of premature convergence," in *Proc. 8th Annu. Conf. Genet. Evol. Comput.*, 2006, pp. 815–822.
[27] J. Lehman and K. Stanley, "Exploiting open-endedness to solve problems through the search for novelty," in *Proc. 11th Int. Conf ALIFE*, vol. 54. 2008, pp. 329–336.
[28] G. S. Hornby, "Steady-state ALPS for real-valued problems," in *Proc. 11th Annu. Conf. Genet. Evol. Comput.*, 2009, pp. 795–802.
[29] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. New York: Wiley, 2001.
[30] R. Beer, "Parameter space structure of continuous-time recurrent neural networks," *Neural Comput.*, vol. 18, no. 12, pp. 3009–3051, 2006.
[31] G. Gomez and P. Eggenberger, "Evolutionary synthesis of grasping through self-exploratory movements of a robotic hand," in *Proc. IEEE CEC*, Sep. 2007, pp. 3418–3425.
[32] Y. Yamashita and J. Tani, "Emergence of functional hierarchy in a multiple timescale neural network model: A humanoid robot experiment," *PLoS Comput. Biol.*, vol. 4, no. 11, p. e1000220, Nov. 2008.
[33] E. Tuci, G. Massera, and S. Nolfi, "Active categorical perception in an evolved anthropomorphic robotic arm," in *Proc. IEEE CEC*, May 2009, pp. 31–38.
[34] J. C. Bongard, "Accelerating self-modeling in cooperative robot teams," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 321–332, Apr. 2009.
[35] W. Hillis, "Co-evolving parasites improve simulated evolution as an optimization procedure," *Physica D*, vol. 42, nos. 1–3, pp. 228–234, 1990.
[36] J. Bongard and H. Lipson, "Nonlinear system identification using coevolution of models and tests," *IEEE Trans. Evol. Comput.*, vol. 9, no. 4, pp. 361–384, Aug. 2005.
[37] H. S. Seung, M. Opper, and H. Sompolinsky, "Query by committee," in *Proc. 5th Workshop Comput. Learning Theory*, 1992, pp. 287–294.
[38] J. Bongard and H. Lipson, "Active coevolutionary learning of deterministic finite automata," *J. Mach. Learning Res.*, vol. 6, pp. 1651–1678, Oct. 2005.
[39] D. Wood, J. Bruner, and G. Ross, "The role of tutoring in problem solving," *J. Child Psychol Psychiatry*, vol. 17, no. 2, pp. 89–100, 1976.
[40] D. Cohn, L. Atlas, and R. Ladner, "Improving generalization with active learning," *Mach. Learning*, vol. 15, no. 2, pp. 201–221, 1994.
[41] Y. Baram, R. El-Yaniv, and K. Luz, "Online choice of active learning algorithms," *J. Mach. Learning Res.*, vol. 5, pp. 255–291, Mar. 2004.
[42] J. C. Bongard, "A probabilistic functional crossover operator for genetic programming," in *Proc. 12th Annu. Conf. Genet. Comput.*, 2009, pp. 312–318.
[43] N. Jakobi, "Evolutionary robotics and the radical envelope of noise hypothesis," *Adaptive Behavior*, vol. 6, no. 1, pp. 131–174, 1997.

Josh C. Bongard received the B.S. (Honors) degree in computer science from McMaster University, Hamilton, ON, Canada, in 1997, the M.S. degree in evolutionary and adaptive systems from the School of Cognitive and Computing Sciences, University of Sussex, Sussex, U.K., in 1999, and the Ph.D. degree from the Artificial Intelligence Laboratory, University of Zurich, Zurich, Switzerland, for research in the field of evolutionary robotics.

He joined the faculty of the Department of Computer Science, University of Vermont, Burlington, in 2006. Prior to this appointment, he was a Post-Doctoral Researcher with the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY.